Katzenpost administration guide

Katzenpost administration guide

Table of Contents

Introducing Katzenpost, a modern mixnet ... 1
Quickstart guide ... 2
Components and configuration of the Katzenpost mixnet ... 3
    Understanding the Katzenpost components ... 3
        Directory authorities (dirauths) ... 5
        Mix nodes ... 5
        Gateway nodes ... 5
        Service nodes ... 5
        Clients ... 5
    Configuring Katzenpost ... 6
        Configuring directory authorities ... 6
        Configuring mix nodes ... 18
        Configuring gateway nodes ... 29
        Configuring service nodes ... 41
Using the Katzenpost Docker test network ... 57
    Requirements ... 57
    Preparing to run the container image ... 57
    Operating the test mixnet ... 58
        Starting and monitoring the mixnet ... 58
        Testing the mixnet ... 59
        Shutting down the mixnet ... 60
        Uninstalling and cleaning up ... 60
    Network topology and components ... 61
        The Docker file tree ... 63
    ... 63
Tuning the Katzenpost mixnet ... 66
Using Katzenpost from behind a NAT device ... 67
    Using Tor ... 67
    Using `BindAddresses` with `Addresses` ... 68
Appendix: Configuration files from the Docker test mixnet ... 69
    Directory authority ... 69
    Mix node ... 71
    Gateway node ... 72
    Service node ... 74
Operating the Katzenpost mixnet ... 77
    CLI usage for directory authorities ... 77
    CLI usage for servers ... 77
Appendix: Using *gensphinx* ... 78

List of Figures

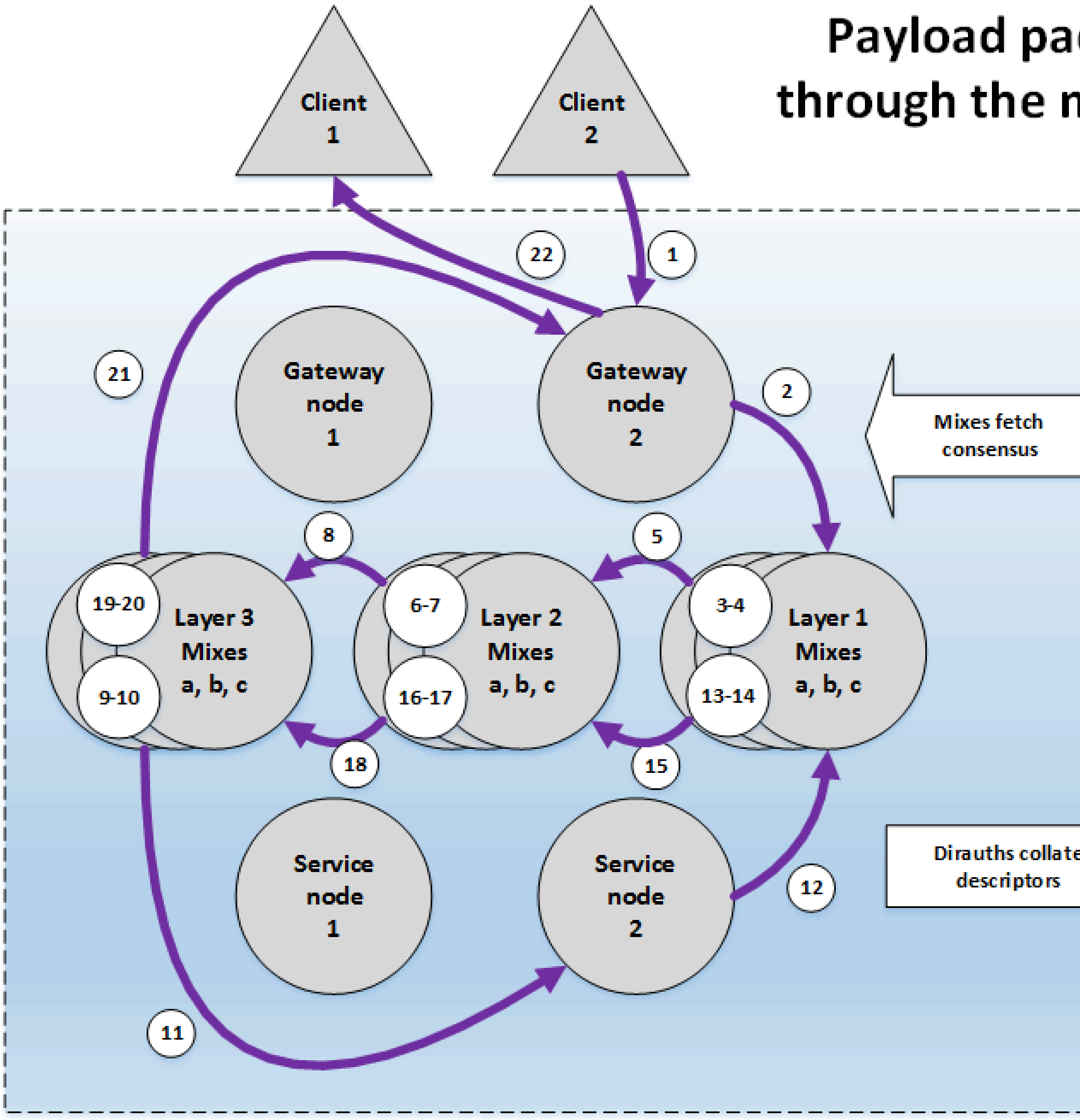
1. The pictured element types correspond to discrete client and server programs that Katzen-
post requires to function. 4
1. Test network topology 62

List of Tables

1. Katzenpost clients 6
2. **Directory authority (dirauth) configuration sections** 6
3. Mix node configuration sections 18
4. Gateway node configuration sections 29
5. Mix node configuration sections 41
1. Table 1: Makefile targets 58
2. Table 2: Test mixnet hosts 63
1. Lan hosting options 67

Introducing Katzenpost, a modern mixnet

To do



Quickstart guide

To do

Components and configuration of the Katzenpost mixnet

This section of the Katzenpost technical documentation provides an introduction to the software components that make up Katzenpost and guidance on how to configure each component. The intended reader is a system administrator who wants to implement a working, production Katzenpost network.

For information about the theory and design of this software, see Introducing Katzenpost, a modern mixnet [https://katzenpost.network/docs/admin_guide/introduction.html]. For a quickly deployable, non-production test network (primarily for use by developers), Using the Katzenpost Docker test network [https://katzenpost.network/docs/admin_guide/docker.html].

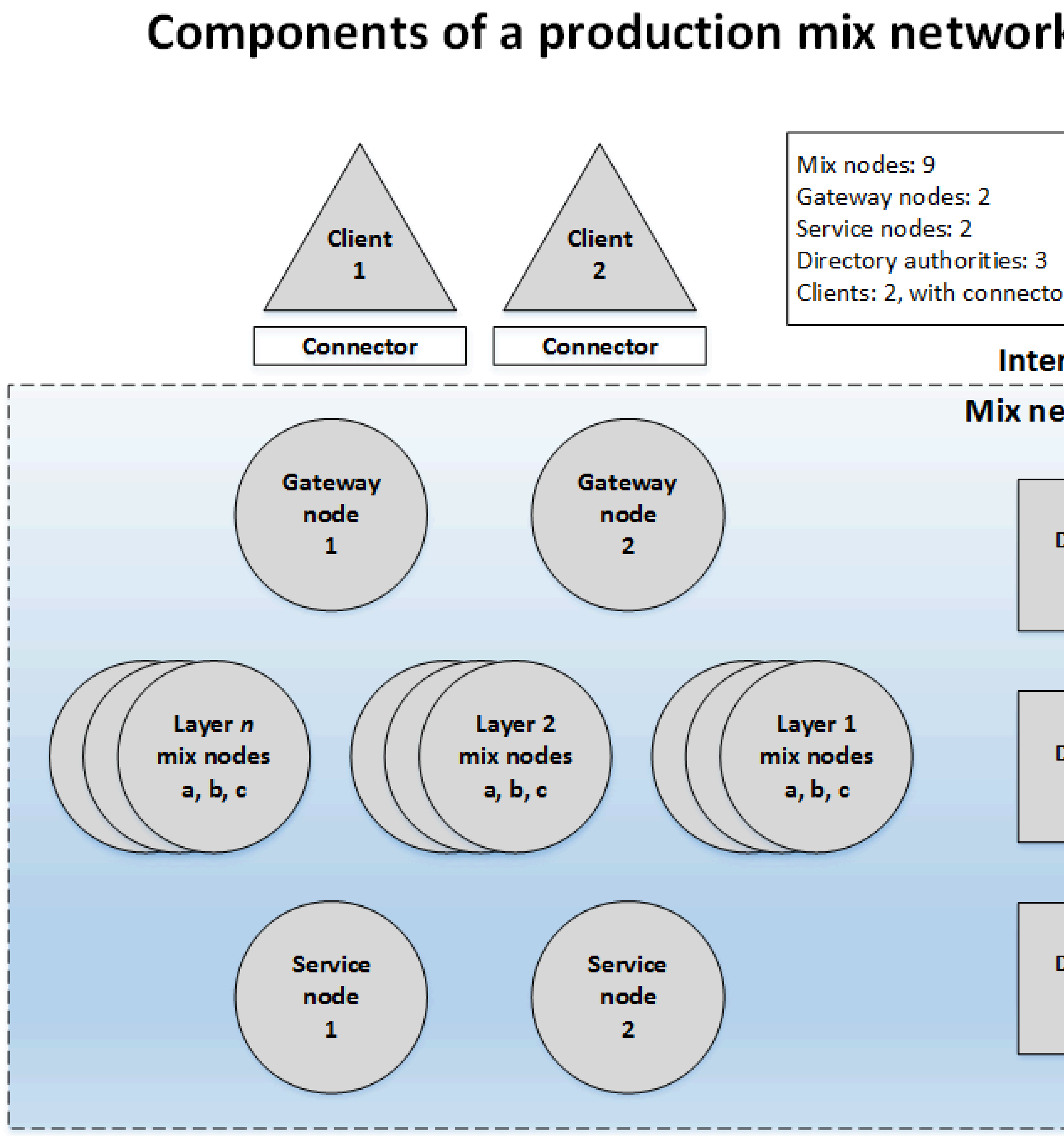
Understanding the Katzenpost components

The core of Katzenpost consists of two program executables, dirauth and server. Running the **dirauth** commmand runs a *directory authority* node, or *dirauth*, that functions as part of the mixnet's public-key infrastructure (PKI). Running the **server** runs either a *mix* node, a *gateway* node, or a *service* node, depending on the configuration. Configuration settings are provided in an associated `katzenpost-authority.toml` or `katzenpost.toml` file respectively.

In addition to the server components, Katzenpost also supports connections to client applications hosted externally to the mix network and communicating with it through gateway nodes.

A model mix network is shown in Figure 1.

Figure 1. The pictured element types correspond to discrete client and server programs that Katzenpost requires to function.



The mix network contains an *n*-layer topology of mix-nodes, with three nodes per layer in this example. Sphinx packets traverse the network in one direction only. The gateway nodes allow clients to interact with the mix network. The service nodes provide mix network services that mix network clients can interact with. All messages sent by clients are handed to a *connector* daemon hosted on the client system, passed across the Internet to a gateway, and then relayed to a service node by way of the nine mix nodes. The service node sends its reply back across the mix-node layers to a gateway, which transmits it across the Internet to be received by the targeted client. The mix, gateway, and service nodes send *mix descriptors* to the dirauths and retrieve a *consensus document* from them, described below.

In addition to the server components, Katzenpost supports connections to client applications hosted externally to the mix network and communicating with it through gateway nodes and, in some cases, a client connector.

Directory authorities (dirauths)

Dirauths compose the decentralized public key infrastructure (PKI) that serves as the root of security for the entire mix network. Clients, mix nodes, gateways nodes, and service nodes rely on the PKI/dirauth system to maintain and sign an up-to-date consensus document, providing a view of the network including connection information and public cryptographic key materials and signatures.

Every 20 minutes (the current value for an *epoch*), each mix, gateway, and service node signs a mix descriptor and uploads it to the dirauths. The dirauths then vote on a new consensus document. If consensus is reached, each dirauth signs the document. Clients and nodes download the document as needed and verify the signatures. Consensus fails when 1/2 + 1 nodes fail, which yields greater fault tolerance than, for example, Byzantine Fault Tolerance, which fails when 1/3 + 1 of the nodes fail.

The PKI signature scheme is fully configurable by the dirauths. Our recommendation is to use a hybrid signature scheme consisting of classical Ed25519 and the post-quantum, stateless, hash-based signature scheme known as Sphincs+ (with the parameters: "sphincs-shake-256f"), which is designated in Katzenpost configurations as "Ed25519 Sphincs+". Examples are provided below.

Mix nodes

The mix node is the fundamental building block of the mix network.

Katzenpost mix nodes are arranged in a layered topology to achieve the best levels of anonymity and ease of analysis while being flexible enough to scale with traffic demands.

Gateway nodes

Gateway nodes provide external client access to the mix network. Because gateways are uniquely positioned to identify clients, they are designed to have as little information about client behavior as possible. Gateways are randomly selected and have no persistent relationship with clients and no knowledge of whether a client's packets are decoys or not. When client traffic through a gateway is slow, the node additionally generates decoy traffic.

Service nodes

Service nodes provide functionality requested by clients. They are logically positioned at the deepest point of the mix network, with incoming queries and outgoing replies both needing to traverse all *n* layers of mix nodes. A service node's functionality may involve storing messages, publishing information outside of the mixnet, interfacing with a blockchain node, and so on. Service nodes also process decoy packets.

Clients

Client applications should be designed so that the following conditions are met:

- Separate service requests from a client are unlinkable. Repeating the same request may be lead to linkability.
- Service nodes and clients have no persistent relationship.
- Cleints generate a stream of packets addressed to random or pseudorandom services regardless of whether a real service request is being made. Most of these packets will be decoy traffic.
- Traffic from a client to a service node must be correctly coupled with decoy traffic. This can mean that the service node is chosen independently from traffic history, or that the transmitted packet replaces a decoy packet that was meant to go to the desired service.

Katzenpost currently includes several client applications. All applications make extensive use of Sphinx single-use reply blocks (SURBs), which enable service nodes to send replies without knowing the location of the client. Newer clients require a connection through the client *connector*, which provides multiplexing and privilege separation with a consequent reduction in processing overhead. These clients also implement the Pigeonhole storage and BACAP protocols detailed in **Place-holder for research paper link**.

The following client applications are available.

Table 1. Katzenpost clients

Name	Needs connector	Description	Code
Ping	no	The mix network equivalent of an ICMP ping utility, used for network testing.	GitHub: ping [https://github.com/katzenpost/katzenpost/tree/main/ping]
Katzen	no	A text chat client with file-transfer support.	GitHub: katzen [https://github.com/katzenpost/katzen]
Status	yes	An HTML page containing status information about the mix network.	GitHub: status [https://github.com/katzenpost/status]
Worldmap	yes	An HTML page with a world map showing geographic locations of mix network nodes.	GitHub: worldmap [https://github.com/katzenpost/worldmap]

Configuring Katzenpost

This section documents the configuration parameters for each type of Katzenpost server node. Each node has its own configuration file in TOML [https://toml.io/en/v1.0.0] format.

Configuring directory authorities

The following configuration is drawn from the reference implementation in `katzenpost/docker/dirauth_mixnet/auth1/authority.toml`. In a real-world mixnet, the component hosts would not be sharing a single IP address. For more information about the test mixnet, see **Using the Katzenpost Docker test network [https://katzenpost.network/docs/admin_guide/docker.html]**.

Table 2. Directory authority (dirauth) configuration sections

Dirauth: Server section
Dirauth: `Authorities` section
Dirauth: Logging section
Dirauth: Parameters section
Dirauth: Debug section
Dirauth: Mixes sections
Dirauth: GatewayNodes section
Dirauth: ServiceNodes sections
Dirauth: Topology section
Dirauth: SphinxGeometry section

Dirauth: Server section

The `Server` section configures mandatory basic parameters for each directory authority.

```
[Server]
    Identifier = "auth1"
    WireKEMScheme = "xwing"
    PKISignatureScheme = "Ed25519 Sphincs+"
    Addresses = ["tcp://127.0.0.1:30001"]
    DataDir = "/dirauth_mixnet/auth1"
```

- **Identifier**

  Specifies the human-readable identifier for a node, and must be unique per mixnet. The identifier can be an FQDN but does not have to be.

  Type: string

  Required: Yes

- **WireKEMScheme**

  Specifies the key encapsulation mechanism (KEM) scheme for the PQ Noise [https://eprint.iacr.org/2022/539]-based wire protocol (link layer) that nodes use to communicate with each other. PQ Noise is a post-quantum variation of the Noise protocol framework [https://noiseprotocol.org/], which algebraically transforms ECDH handshake patterns into KEM encapsulate/decapsulate operations.

  This configuration option supports the optional use of hybrid post-quantum cryptography to strengthen security. The following KEM schemes are supported:

  - **Classical:** "x25519", "x448"

    Note

    X25519 and X448 are actually non-interactive key-exchanges (NIKEs), not KEMs. Katzenpost uses a hashed ElGamal cryptographic construction to convert them from NIKEs to KEMs.

  - **Post-quantum:** "mlkem768","sntrup4591761", "frodo640shake", "mceliece348864", "mceliece348864f", "mceliece460896", "mceliece460896f", "mceliece6688128", "mceliece6688128f", "mceliece6960119", "mceliece6960119f", "mceliece8192128", "mceliece8192128f", "CTIDH511", "CTIDH512", "CTIDH1024", "CTIDH2048",

  - **Hybrid post-quantum:** "xwing", "Kyber768-X25519", "MLKEM768-X25519", "MLKEM768-X448", "FrodoKEM-640-SHAKE-X448", "sntrup4591761-X448", "mceliece348864-X25519", "mceliece348864f-X25519", "mceliece460896-X25519", "mceliece460896f-X25519", "mceliece6688128-X25519", "mceliece6688128f-X25519", "mceliece6960119-X25519", "mceliece6960119f-X25519", "mceliece8192128-X25519", "mceliece8192128f-X25519", "CTIDH512-X25519", "CTIDH512-X25519"

  Type: string

  Required: Yes

- **PKISignatureScheme**

  Specifies the cryptographic signature scheme which will be used by all components of the mix network when interacting with the PKI system. Mix nodes sign their descriptors using this signature scheme, and dirauth nodes similarly sign PKI documents using the same scheme.

The following signature schemes are supported: "ed25519", "ed448", "Ed25519 Sphincs+", "Ed448-Sphincs+", "Ed25519-Dilithium2", "Ed448-Dilithium3"

Type: string

Required: Yes

- **Addresses**

  Specifies a list of one or more address URLs in a format that contains the transport protocol, IP address, and port number that the node will bind to for incoming connections. Katzenpost supports URLs with that start with either "tcp://" or "quic://" such as: ["tcp://192.168.1.1:30001"] and ["quic://192.168.1.1:40001"].

  Type: []string

  Required: Yes

- **DataDir**

  Specifies the absolute path to a node's state directory. This is where `persistence.db` is written to disk and where a node stores its cryptographic key materials when started with the "-g" command-line option.

  Type: string

  Required: Yes

Dirauth: `Authorities` section

An `Authorities` section is configured for each peer authority. We recommend using TOML's style [https://quickref.me/toml.html] for multi-line quotations for key materials.

```
[[Authorities]]
    Identifier = "auth1"
    IdentityPublicKey = """
-----BEGIN ED25519 PUBLIC KEY-----
dYpXpbozjFfqhR45ZC2q97SOOsXMANdHaEdXrP42CJk=
-----END ED25519 PUBLIC KEY-----
"""
    PKISignatureScheme = "Ed25519"
    LinkPublicKey = """
-----BEGIN XWING PUBLIC KEY-----
ooQBPYNdmfwnxXmvnljPA2mG5gWgurfHhbY87DMRY2tbMeZpinJ5BlSiIecprnmm
QqxcS9o36IS62SVMlOUkw+XEZGVvc9wJqHpgEgVJRAs1PCR8cUAdM6QIYLWt/lkf
SPKDCtZ3GiSIOzMuaglo2tarIPEv1AY7r9B0xXOgSKMkGyBkCfw1VBZf46MM26NL
...
gHtNyQJnXski52O03JpZRIhR40pFOhAAcMMAZDpMTVoxlcdR6WA4SlBiSceeJBgY
Yp9PlGhCimx9am99TrdLoLCdTHB6oowt8tss3POpIOxaSlguyeym/sBhkUrnXOgN
ldMtDsvvc9KUfE4I0+c+XQ==
-----END XWING PUBLIC KEY-----
    """
    WireKEMScheme = "xwing"
    Addresses = ["tcp://127.0.0.1:30001"]
```

- **Identifier**

  Specifies the human-readable identifier for the node which must be unique per mixnet. The identifier can be an FQDN but does not have to be.

Type: string

Required: Yes

- **IdentityPublicKey**

  String containing the node's public identity key in PEM format. `IdentityPublicKey` is the node's permanent identifier and is used to verify cryptographic signatures produced by its private identity key.

  Type: string

  Required: Yes

- **PKISignatureScheme**

  Specifies the cryptographic signature scheme used by all directory authority nodes. `PKISignatureScheme` must match the scheme specified in the `Server` section of the configuration.

  Type: string

  Required: Yes

- **LinkPublicKey**

  String containing the peer's public link-layer key in PEM format. `LinkPublicKey` must match the specified `WireKEMScheme`.

  Type: string

  Required: Yes

- **WireKEMScheme**

  Specifies the key encapsulation mechanism (KEM) scheme for the PQ Noise [https://eprint.iacr.org/2022/539]-based wire protocol (link layer) that nodes use to communicate with each other. PQ Noise is a post-quantum variation of the Noise protocol framework [https://noiseprotocol.org/], which algebraically transforms ECDH handshake patterns into KEM encapsulate/decapsulate operations.

  This configuration option supports the optional use of hybrid post-quantum cryptography to strengthen security. The following KEM schemes are supported:

  - **Classical:** "x25519", "x448"

    Note

    X25519 and X448 are actually non-interactive key-exchanges (NIKEs), not KEMs. Katzenpost uses a hashed ElGamal cryptographic construction to convert them from NIKEs to KEMs.

  - **Post-quantum:** "mlkem768","sntrup4591761", "frodo640shake", "mceliece348864", "mceliece348864f", "mceliece460896", "mceliece460896f", "mceliece6688128", "mceliece6688128f", "mceliece6960119", "mceliece6960119f", "mceliece8192128", "mceliece8192128f", "CTIDH511", "CTIDH512", "CTIDH1024", "CTIDH2048",

  - **Hybrid post-quantum:** "xwing", "Kyber768-X25519", "MLKEM768-X25519", "MLKEM768-X448", "FrodoKEM-640-SHAKE-X448", "sntrup4591761-X448", "mceliece348864-X25519", "mceliece348864f-X25519", "mceliece460896-X25519", "mceliece460896f-X25519", "mceliece6688128-X25519", "mceliece6688128f-X25519", "mceliece6960119-

X25519", "mceliece6960119f-X25519", "mceliece8192128-X25519", "mceliece8192128f-X25519", "CTIDH512-X25519", "CTIDH512-X25519"

Type: string

Required: Yes

- **Addresses**

  Specifies a list of one or more address URLs in a format that contains the transport protocol, IP address, and port number that the node will bind to for incoming connections. Katzenpost supports URLs with that start with either "tcp://" or "quic://" such as: ["tcp://192.168.1.1:30001"] and ["quic://192.168.1.1:40001"].

  Type: []string

  Required: Yes

Dirauth: Logging section

The `Logging` configuration section controls logging behavior across Katzenpost.

```
[Logging]
                Disable = false
                File = "katzenpost.log"
                Level = "INFO"
```

- **Disable**

  If **true**, logging is disabled.

  Type: bool

  Required: No

- **File**

  Specifies the log file. If omitted, `stdout` is used.

  An absolute or relative file path can be specified. A relative path is relative to the DataDir specified in the `Server` section of the configuration.

  Type: string

  Required: No

- **Level**

  Supported logging level values are ERROR | WARNING | NOTICE |INFO | DEBUG.

  Type: string

  Required: No

  Warning

  The DEBUG log level is unsafe for production use.

Dirauth: Parameters section

The `Parameters` section contains the network parameters.

```
[Parameters]
    SendRatePerMinute = 0
    Mu = 0.005
    MuMaxDelay = 1000
    LambdaP = 0.001
    LambdaPMaxDelay = 1000
    LambdaL = 0.0005
    LambdaLMaxDelay = 1000
    LambdaD = 0.0005
    LambdaDMaxDelay = 3000
    LambdaM = 0.0005
    LambdaG = 0.0
    LambdaMMaxDelay = 100
    LambdaGMaxDelay = 100
```

- **SendRatePerMinute**

  Specifies the maximum allowed rate of packets per client per gateway node. Rate limiting is done on the gateway nodes.

  Type: uint64

  Required: Yes

- **Mu**

  Specifies the inverse of the mean of the exponential distribution from which the Sphinx packet per-hop mixing delay will be sampled.

  Type: float64

  Required: Yes

- **MuMaxDelay**

  Specifies the maximum Sphinx packet per-hop mixing delay in milliseconds.

  Type: uint64

  Required: Yes

- **LambdaP**

  Specifies the inverse of the mean of the exponential distribution that clients sample to determine the time interval between sending messages, whether actual messages from the FIFO egress queue or decoy messages if the queue is empty.

  Type: float64

  Required: Yes

- **LambdaPMaxDelay**

  Specifies the maximum send delay interval for LambdaP in milliseconds.

  Type: uint64

  Required: Yes

- **LambdaL**

Specifies the inverse of the mean of the exponential distribution that clients sample to determine the delay interval between loop decoys.

Type: float64

Required: Yes

- **LambdaLMaxDelay**

  Specifies the maximum send delay interval for LambdaL in milliseconds.

  Type: uint64

  Required: Yes

- **LambdaD**

  LambdaD is the inverse of the mean of the exponential distribution that clients sample to determine the delay interval between decoy drop messages.

  Type: float64

  Required: Yes

- **LambdaDMaxDelay**

  Specifies the maximum send interval in for LambdaD in milliseconds.

  Type: uint64

  Required: Yes

- **LambdaM**

  LambdaM is the inverse of the mean of the exponential distribution that mix nodes sample to determine the delay between mix loop decoys.

  Type: float64

  Required: Yes

- **LambdaG**

  LambdaG is the inverse of the mean of the exponential distribution that gateway nodes to select the delay between gateway node decoys.

  ### Warning

  Do not set this value manually in the TOML configuration file. The field is used internally by the dirauth server state machine.

  Type: float64

  Required: Yes

- **LambdaMMaxDelay**

  Specifies the maximum delay for LambdaM in milliseconds.

  Type: uint64

  Required: Yes

- **LambdaGMaxDelay**

  Specifies the maximum delay for LambdaG in milliseconds.

  Type: uint64

  Required: Yes

Dirauth: Debug section

```
[Debug]
    Layers = 3
    MinNodesPerLayer = 1
    GenerateOnly = false
```

- **Layers**

  Specifies the number of non-service-provider layers in the network topology.

  Type: int

  Required: Yes

- **MinNodesrPerLayer**

  Specifies the minimum number of nodes per layer required to form a valid consensus document.

  Type: int

  Required: Yes

- **GenerateOnly**

  If **true**, the server halts and cleans up the data directory immediately after long-term key generation.

  Type: bool

  Required: No

Dirauth: Mixes sections

The `Mixes` configuration sections list mix nodes that are known to the authority.

```
[[Mixes]]
    Identifier = "mix1"
    IdentityPublicKeyPem = "../mix1/identity.public.pem"

[[Mixes]]
    Identifier = "mix2"
    IdentityPublicKeyPem = "../mix2/identity.public.pem"

[[Mixes]]
    Identifier = "mix3"
    IdentityPublicKeyPem = "../mix3/identity.public.pem"
```

- **Identifier**

  Specifies the human-readable identifier for a mix node, and must be unique per mixnet. The identifier can be an FQDN but does not have to be.

  Type: string

Required: Yes

- **IdentityPublicKeyPem**

  Path and file name of a mix node's public identity signing key, also known as the identity key, in PEM format.

  Type: string

  Required: Yes

Dirauth: GatewayNodes section

The `GatewayNodes` sections list gateway nodes that are known to the authority.

```
[[GatewayNodes]]
    Identifier = "gateway1"
    IdentityPublicKeyPem = "../gateway1/identity.public.pem"
```

- **Identifier**

  Specifies the human-readable identifier for a gateway node, and must be unique per mixnet. Identifier can be an FQDN but does not have to be.

  Type: string

  Required: Yes

- **IdentityPublicKeyPem**

  Path and file name of a gateway node's public identity signing key, also known as the identity key, in PEM format.

  Type: string

  Required: Yes

Dirauth: ServiceNodes sections

The `ServiceNodes` sections list service nodes that are known to the authority.

```
[[ServiceNodes]]
    Identifier = "servicenode1"
    IdentityPublicKeyPem = "../servicenode1/identity.public.pem"
```

- **Identifier**

  Specifies the human-readable identifier for a service node, and must be unique per mixnet. Identifier can be an FQDN but does not have to be.

  Type: string

  Required: Yes

- **IdentityPublicKeyPem**

  Path and file name of a service node's public identity signing key, also known as the identity key, in PEM format.

  Type: string

  Required: Yes

Dirauth: Topology section

The `Topology` section defines the layers of the mix network and the mix nodes in each layer.

```
[Topology]

    [[Topology.Layers]]

        [[Topology.Layers.Nodes]]
            Identifier = "mix1"
            IdentityPublicKeyPem = "../mix1/identity.public.pem"

    [[Topology.Layers]]

        [[Topology.Layers.Nodes]]
            Identifier = "mix2"
            IdentityPublicKeyPem = "../mix2/identity.public.pem"

    [[Topology.Layers]]

        [[Topology.Layers.Nodes]]
            Identifier = "mix3"
            IdentityPublicKeyPem = "../mix3/identity.public.pem"
```

- **Identifier**

  Specifies the human-readable identifier for a node, and must be unique per mixnet. The identifier can be an FQDN but does not have to be.

  Type: string

- **IdentityPublicKeyPem**

  Path and file name of a mix node's public identity signing key, also known as the identity key, in PEM format.

  Type: string

  Required: Yes

Dirauth: SphinxGeometry section

Sphinx is an encrypted nested-packet format designed primarily for mixnets. The original Sphinx paper [https://www.freehaven.net/anonbib/cache/DBLP:conf/sp/DanezisG09.pdf] described a non-interactive key exchange (NIKE) employing classical encryption. The Katzenpost implementation strongly emphasizes configurability, supporting key encapsulation mechanisms (KEMs) as well as NIKEs, and enabling the use of either classical or hybrid post-quantum cryptography. Hybrid constructions offset the newness of post-quantum algorithms by offering heavily tested classical algorithms as a fallback.

Note

Sphinx, the nested-packet format, should not be confused with Sphincs or Sphincs+ [http://sphincs.org/index.html], which are post-quantum signature schemes.

Katzenpost Sphinx also relies on the following classical cryptographic primitives:

- CTR-AES256, a stream cipher
- HMAC-SHA256, a message authentication code (MAC) function
- HKDF-SHA256, a key derivation function (KDF)

- AEZv5, a strong pseudorandom permutation (SPRP)

All dirauths must be configured to use the same `SphinxGeometry` parameters. Any geometry not advertised by the PKI document will fail. Each dirauth publishes the hash of its `SphinxGeometry` parameters in the PKI document for validation by its peer dirauths.

The `SphinxGeometry` section defines parameters for the Sphinx encrypted nested-packet format used internally by Katzenpost.

Warning

The values in the `SphinxGeometry` configuration section must be programmatically generated by **gensphinx**. Many of the parameters are interdependent and cannot be individually modified. Do not modify the these values by hand.

The settings in this section are generated by the **gensphinx** utility, which computes the Sphinx geometry based on the following user-supplied directives:

- The number of mix node layers (not counting gateway and service nodes)
- The length of the application-usable packet payload
- The selected NIKE or KEM scheme

The output in TOML should then be pasted unchanged into the node's configuration file, as shown below. For more information, see Appendix: Using *gensphinx* [https://katzenpost.network/docs/admin_guide/gensphinx.html].

```
[SphinxGeometry]
                PacketLength = 3082
                NrHops = 5
                HeaderLength = 476
                RoutingInfoLength = 410
                PerHopRoutingInfoLength = 82
                SURBLength = 572
                SphinxPlaintextHeaderLength = 2
                PayloadTagLength = 32
                ForwardPayloadLength = 2574
                UserForwardPayloadLength = 2000
                NextNodeHopLength = 65
                SPRPKeyMaterialLength = 64
                NIKEName = "x25519"
                KEMName = ""
```

- **PacketLength**

  The length of a Sphinx packet in bytes.

  Type: int

  Required: Yes

- **NrHops**

  The number of hops a Sphinx packet takes through the mixnet. Because packet headers hold destination information for each hop, the size of the header increases linearly with the number of hops.

  Type: int

  Required: Yes

- **HeaderLength**

The total length of the Sphinx packet header in bytes.

Type: int

Required: Yes

- **RoutingInfoLength**

  The total length of the routing information portion of the Sphinx packet header.

  Type: int

  Required: Yes

- **PerHopRoutingInfoLength**

  The length of the per-hop routing information in the Sphinx packet header.

  Type: int

  Required: Yes

- **SURBLength**

  The length of a single-use reply block (SURB).

  Type: int

  Required: Yes

- **SphinxPlaintextHeaderLength**

  The length of the plaintext Sphinx packet header.

  Type: int

  Required: Yes

- **PayloadTagLength**

  The length of the payload tag.

  Type: int

  Required: Yes

- **ForwardPayloadLength**

  The total size of the payload.

  Type: int

  Required: Yes

- **UserForwardPayloadLength**

  The size of the usable payload.

  Type: int

  Required: Yes

- **NextNodeHopLength**

The `NextNodeHopLength` is derived from the largest routing-information block that we expect to encounter. Other packets have `NextNodeHop` + `NodeDelay` sections, or a `Recipient` section, both of which are shorter.

Type: int

Required: Yes

- **SPRPKeyMaterialLength**

  The length of the strong pseudo-random permutation (SPRP) key.

  Type: int

  Required: Yes

- **NIKEName**

  The name of the non-interactive key exchange (NIKE) scheme used by Sphinx packets.

  `NIKEName` and `KEMName` are mutually exclusive.

  Type: string

  Required: Yes

- **KEMName**

  The name of the key encapsulation mechanism (KEM) used by Sphinx packets.

  `NIKEName` and `KEMName` are mutually exclusive.

  Type: string

  Required: Yes

Configuring mix nodes

The following configuration is drawn from the reference implementation in `katzenpost/docker/dirauth_mixnet/mix1/katzenpost.toml`. In a real-world mixnet, the component hosts would not be sharing a single IP address. For more information about the test mixnet, see Using the Katzenpost Docker test network [https://katzenpost.network/docs/admin_guide/docker.html].

Table 3. Mix node configuration sections

Mix node: Server section
Mix node: Logging section
Mix node: PKI section
Mix node: Management section
Mix node: SphinxGeometry section
Mix node: Debug section

Mix node: Server section

The `Server` section configures mandatory basic parameters for each server node.

```
[Server]
  Identifier = "mix1"
  WireKEM = "xwing"
```

```
PKISignatureScheme = "Ed25519"
Addresses = ["127.0.0.1:30008"]
OnlyAdvertiseAltAddresses = false
MetricsAddress = "127.0.0.1:30009"
DataDir = "/dirauth_mixnet/mix1"
IsGatewayNode = false
IsServiceNode = false
[Server.AltAddresses]
```

- **Identifier**

  Specifies the human-readable identifier for a node, and must be unique per mixnet. The identifier can be an FQDN but does not have to be.

  Type: string

  Required: Yes

- **WireKEM**

  WireKEM specifies the key encapsulation mechanism (KEM) scheme for the PQ Noise [https://eprint.iacr.org/2022/539]-based wire protocol (link layer) that nodes use to communicate with each other. PQ Noise is a post-quantum variation of the Noise protocol framework [https://noiseprotocol.org/], which algebraically transforms ECDH handshake patterns into KEM encapsulate/decapsulate operations.

  This configuration option supports the optional use of hybrid post-quantum cryptography to strengthen security. The following KEM schemes are supported:

  - **Classical:** "x25519", "x448"

    ### Note

    X25519 and X448 are actually non-interactive key-exchanges (NIKEs), not KEMs. Katzenpost uses a hashed ElGamal cryptographic construction to convert them from NIKEs to KEMs.

  - **Post-quantum:** "mlkem768","sntrup4591761", "frodo640shake", "mceliece348864", "mceliece348864f", "mceliece460896", "mceliece460896f", "mceliece6688128", "mceliece6688128f", "mceliece6960119", "mceliece6960119f", "mceliece8192128", "mceliece8192128f", "CTIDH511", "CTIDH512", "CTIDH1024", "CTIDH2048",

  - **Hybrid post-quantum:** "xwing", "Kyber768-X25519", "MLKEM768-X25519", "MLKEM768-X448", "FrodoKEM-640-SHAKE-X448", "sntrup4591761-X448", "mceliece348864-X25519", "mceliece348864f-X25519", "mceliece460896-X25519", "mceliece460896f-X25519", "mceliece6688128-X25519", "mceliece6688128f-X25519", "mceliece6960119-X25519", "mceliece6960119f-X25519", "mceliece8192128-X25519", "mceliece8192128f-X25519", "CTIDH512-X25519", "CTIDH512-X25519"

  Type: string

  Required: Yes

- **PKISignatureScheme**

  Specifies the cryptographic signature scheme that will be used by all components of the mix network when interacting with the PKI system. Mix nodes sign their descriptors using this signature scheme, and dirauth nodes similarly sign PKI documents using the same scheme.

  The following signature schemes are supported:

  - **Classical:** "ed25519", "ed448"
  - **Hybrid post-quantum:** "Ed25519 Sphincs+", "Ed448-Sphincs+", "Ed25519-Dilithium2", "Ed448-Dilithium3"

  Type: string

  Required: Yes

- **Addresses**

  Specifies a list of one or more address URLs in a format that contains the transport protocol, IP address, and port number that the server will bind to for incoming connections. Katzenpost supports URLs with that start with either "tcp://" or "quic://" such as: ["tcp://192.168.1.1:30001"] and ["quic://192.168.1.1:40001"].

  Type: []string

  Required: Yes

- **BindAddresses**

  If **true**, allows setting of listener addresses that the server will bind to and accept connections on. These addresses are not advertised in the PKI.

  Type: bool

  Required: No

- **MetricsAddress**

  Specifies the address/port to bind the Prometheus metrics endpoint to.

  Type: string

  Required: No

- **DataDir**

  Specifies the absolute path to a node's state directory. This is where persistence.db is written to disk and where a node stores its cryptographic key materials when started with the "-g" commmand-line option.

  Type: string

  Required: Yes

- **IsGatewayNode**

  If **true**, the server is a gateway node.

  Type: bool

  Required: No

- **IsServiceNode**

  If **true**, the server is a service node.

  Type: bool

  Required: No

Mix node: Logging section

The `Logging` configuration section controls logging behavior across Katzenpost.

```
[Logging]
                Disable = false
                File = "katzenpost.log"
                Level = "INFO"
```

- **Disable**

  If **true**, logging is disabled.

  Type: bool

  Required: No

- **File**

  Specifies the log file. If omitted, `stdout` is used.

  An absolute or relative file path can be specified. A relative path is relative to the DataDir specified in the `Server` section of the configuration.

  Type: string

  Required: No

- **Level**

  Supported logging level values are ERROR | WARNING | NOTICE |INFO | DEBUG.

  Type: string

  Required: No

  Warning

  The DEBUG log level is unsafe for production use.

Mix node: PKI section

The `PKI` section contains the directory authority configuration for a mix, gateway, or service node.

```
[PKI]
[PKI.dirauth]

    [[PKI.dirauth.Authorities]]
        Identifier = "auth1"
        IdentityPublicKey = """-----BEGIN ED25519 PUBLIC KEY-----
tqN6tpOVotHWXKCszVn2kS7vAZjQpvJjQF3Qz/Qwhyg=
-----END ED25519 PUBLIC KEY-----
"""
        PKISignatureScheme = "Ed25519"
        LinkPublicKey = """-----BEGIN XWING PUBLIC KEY-----
JnJ8ztQEIjAkKJcpuZvJAdkWjBim/5G5d8yoosEQHeGJeeBqNPdm2AitUbpiQPcd
tNCo9DxuC9Ieqmsfw0YpV6AtOOsaInA6QnHDYcuBfZcQL5MU4+t2TzpBZQYlrSED
hPCKrAG+8GEUl6akseG371WQzEtPpEWWCJCJOiS/VDFZT7eKrldlumN6gfiB84sR
...
```

```
arFh/WKwYJUj+aGBsFYSqGdzC6MdY4x/YyFe2ze0MJEjThQE91y1d/LCQ3Sb7Ri+
u6PBi3JU2qzlPEejDKwK0t5tMNEAkq8iNrpRTdD/hS0gR+ZIN8Z9QKh7Xf94FWG2
H+r8OaqImQhgHabrWRDyLg==
-----END XWING PUBLIC KEY-----
"""
        WireKEMScheme = "xwing"
        Addresses = ["127.0.0.1:30001"]

    [[PKI.dirauth.Authorities]]
        Identifier = "auth2"
        IdentityPublicKey = """-----BEGIN ED25519 PUBLIC KEY-----
O51Ty2WLu4C1ETMa29s03bMXV72gnjJfTfwLV++LVBI=
-----END ED25519 PUBLIC KEY-----
"""
        PKISignatureScheme = "Ed25519"
        LinkPublicKey = """-----BEGIN XWING PUBLIC KEY-----
TtQkg2XKUnY602FFBaPJ+zpN0Twy20cwyyFxh7FNUjaXA9MAJXs0vUwFbJc6BjYv
f+olKnlIKFSmDvcF74U6w1F0ObugwTNKNxeYKPKhX4FiencUbRwkHoYHdtZdSctz
TKy08qKQyCAccqCRpdo6ZtYXPAU+2rthjYTOL7Zn+7SHUKCuJClcPnvEYjVcJxtZ
...
ubJIe5U4nMJbBkOqr7Kq6niaEkiLODa0tkpB8tKMYTMBdcYyHSXCzpo7U9sb6LAR
HktiTBDtRXviu2vbw7VRXhkMW2kjYZDtReQ5sAse04DvmD49zgTp1YxYW+wWFaL3
37X7/SNuLdHX4PHZXIWHBQ==
-----END XWING PUBLIC KEY-----
"""
        WireKEMScheme = "xwing"
        Addresses = ["127.0.0.1:30002"]

    [[PKI.dirauth.Authorities]]
        Identifier = "auth3"
        IdentityPublicKey = """-----BEGIN ED25519 PUBLIC KEY-----
zQvydRYJq3npeLcg1NqIf+SswEKE5wFmiwNsI9Z1whQ=
-----END ED25519 PUBLIC KEY-----
"""
        PKISignatureScheme = "Ed25519"
        LinkPublicKey = """
-----BEGIN XWING PUBLIC KEY-----
OYK9FiC53xwZ1VST3jDOO4tR+cUMSVRSekmigZMChSjDCPZbKut8TblxtlUfc/yi
Ugorz4NIvYPMWUt3QPwS2UWq8/HMWXNGPUiAevg12+oV+jOJXaJeCfY24UekJnSw
TNcdGaFZFSR0FocFcPBBnrK1M2B8w8eEUKQIsXRDM3x/8aRIuDif+ve8rSwpgKeh
...
OdVD3yw7OOS8uPZLORGQFyJbHtVmFPVvwja4G/o2gntAoHUZ2LiJJakpVhhlSyrI
yuzvwwFtZVfWtNb5gAKZCyg0aduR3qgd7MPerRF+YopZk3OCRpC02YxfUZrHv398
FZWJFK0R8iU52CEUxVpXTA==
-----END XWING PUBLIC KEY-----
"""
        WireKEMScheme = "xwing"
        Addresses = ["127.0.0.1:30003"]
```

- **Identifier**

  Specifies the human-readable identifier for a node, which must be unique per mixnet. The identifier can be an FQDN but does not have to be.

  Type: string

  Required: Yes

- **IdentityPublicKey**

String containing the node's public identity key in PEM format. `IdentityPublicKey` is the node's permanent identifier and is used to verify cryptographic signatures produced by its private identity key.

Type: string

Required: Yes

- **PKISignatureScheme**

  Specifies the cryptographic signature scheme that will be used by all components of the mix network when interacting with the PKI system. Mix nodes sign their descriptors using this signature scheme, and dirauth nodes similarly sign PKI documents using the same scheme.

  Type: string

  Required: Yes

- **LinkPublicKey**

  String containing the peer's public link-layer key in PEM format. `LinkPublicKey` must match the specified `WireKEMScheme`.

  Type: string

  Required: Yes

- **WireKEMScheme**

  The name of the wire protocol key-encapsulation mechanism (KEM) to use.

  Type: string

  Required: Yes

- **Addresses**

  Specifies a list of one or more address URLs in a format that contains the transport protocol, IP address, and port number that the server will bind to for incoming connections. Katzenpost supports URLs that start with either "tcp://" or "quic://" such as: ["tcp://192.168.1.1:30001"] and ["quic://192.168.1.1:40001"].

  Type: []string

  Required: Yes

Mix node: Management section

The `Management` section specifies connectivity information for the Katzenpost control protocol which can be used to make run-time configuration changes. A configuration resembles the following:

```
[Management]
   Enable = false
   Path = "/dirauth_mixnet/mix1/management_sock"
```

- **Enable**

  If **true**, the management interface is enabled.

  Type: bool

Required: No

- **Path**

  Specifies the path to the management interface socket. If left empty, then `management_sock` is located in the configuration's defined `DataDir`>.

  Type: string

  Required: No

Mix node: SphinxGeometry section

The `SphinxGeometry` section defines parameters for the Sphinx encrypted nested-packet format used internally by Katzenpost.

Warning

The values in the `SphinxGeometry` configuration section must be programmatically generated by **gensphinx**. Many of the parameters are interdependent and cannot be individually modified. Do not modify the these values by hand.

The settings in this section are generated by the **gensphinx** utility, which computes the Sphinx geometry based on the following user-supplied directives:

- The number of mix node layers (not counting gateway and service nodes)
- The length of the application-usable packet payload
- The selected NIKE or KEM scheme

The output in TOML should then be pasted unchanged into the node's configuration file, as shown below. For more information, see Appendix: Using *gensphinx* [https://katzenpost.network/docs/admin_guide/gensphinx.html].

```
[SphinxGeometry]
                PacketLength = 3082
                NrHops = 5
                HeaderLength = 476
                RoutingInfoLength = 410
                PerHopRoutingInfoLength = 82
                SURBLength = 572
                SphinxPlaintextHeaderLength = 2
                PayloadTagLength = 32
                ForwardPayloadLength = 2574
                UserForwardPayloadLength = 2000
                NextNodeHopLength = 65
                SPRPKeyMaterialLength = 64
                NIKEName = "x25519"
                KEMName = ""
```

- **PacketLength**

  The length of a Sphinx packet in bytes.

  Type: int

  Required: Yes

- **NrHops**

The number of hops a Sphinx packet takes through the mixnet. Because packet headers hold destination information for each hop, the size of the header increases linearly with the number of hops.

Type: int

Required: Yes

- **HeaderLength**

  The total length of the Sphinx packet header in bytes.

  Type: int

  Required: Yes

- **RoutingInfoLength**

  The total length of the routing information portion of the Sphinx packet header.

  Type: int

  Required: Yes

- **PerHopRoutingInfoLength**

  The length of the per-hop routing information in the Sphinx packet header.

  Type: int

  Required: Yes

- **SURBLength**

  The length of a single-use reply block (SURB).

  Type: int

  Required: Yes

- **SphinxPlaintextHeaderLength**

  The length of the plaintext Sphinx packet header.

  Type: int

  Required: Yes

- **PayloadTagLength**

  The length of the payload tag.

  Type: int

  Required: Yes

- **ForwardPayloadLength**

  The total size of the payload.

  Type: int

  Required: Yes

- **UserForwardPayloadLength**

  The size of the usable payload.

  Type: int

  Required: Yes

- **NextNodeHopLength**

  The `NextNodeHopLength` is derived from the largest routing-information block that we expect to encounter. Other packets have `NextNodeHop` + `NodeDelay` sections, or a `Recipient` section, both of which are shorter.

  Type: int

  Required: Yes

- **SPRPKeyMaterialLength**

  The length of the strong pseudo-random permutation (SPRP) key.

  Type: int

  Required: Yes

- **NIKEName**

  The name of the non-interactive key exchange (NIKE) scheme used by Sphinx packets.

  `NIKEName` and `KEMName` are mutually exclusive.

  Type: string

  Required: Yes

- **KEMName**

  The name of the key encapsulation mechanism (KEM) used by Sphinx packets.

  `NIKEName` and `KEMName` are mutually exclusive.

  Type: string

  Required: Yes

Mix node: Debug section

The `Debug` section is the Katzenpost server debug configuration for advanced tuning.

```
[Debug]
                NumSphinxWorkers = 16
                NumServiceWorkers = 3
                NumGatewayWorkers = 3
                NumKaetzchenWorkers = 3
                SchedulerExternalMemoryQueue = false
                SchedulerQueueSize = 0
                SchedulerMaxBurst = 16
                UnwrapDelay = 250
                GatewayDelay = 500
                ServiceDelay = 500
```

```
KaetzchenDelay = 750
SchedulerSlack = 150
SendSlack = 50
DecoySlack = 15000
ConnectTimeout = 60000
HandshakeTimeout = 30000
ReauthInterval = 30000
SendDecoyTraffic = false
DisableRateLimit = false
GenerateOnly = false
```

- **NumSphinxWorkers**

  Specifies the number of worker instances to use for inbound Sphinx packet processing.

  Type: int

  Required: No

- **NumProviderWorkers**

  Specifies the number of worker instances to use for provider specific packet processing.

  Type: int

  Required: No

- **NumKaetzchenWorkers**

  Specifies the number of worker instances to use for Kaetzchen-specific packet processing.

  Type: int

  Required: No

- **SchedulerExternalMemoryQueue**

  If **true**, the experimental disk-backed external memory queue is enabled.

  Type: bool

  Required: No

- **SchedulerQueueSize**

  Specifies the maximum scheduler queue size before random entries will start getting dropped. A value less than or equal to zero is treated as unlimited.

  Type: int

  Required: No

- **SchedulerMaxBurst**

  Specifies the maximum number of packets that will be dispatched per scheduler wakeup event.

  Type:

  Required: No

- **UnwrapDelay**

  Specifies the maximum unwrap delay due to queueing in milliseconds.

Type: int

Required: No

- **GatewayDelay**

  Specifies the maximum gateway node worker delay due to queueing in milliseconds.

  Type: int

  Required: No

- **ServiceDelay**

  Specifies the maximum provider delay due to queueing in milliseconds.

  Type: int

  Required: No

- **KaetzchenDelay**

  Specifies the maximum kaetzchen delay due to queueing in milliseconds.

  Type: int

  Required: No

- **SchedulerSlack**

  Specifies the maximum scheduler slack due to queueing and/or processing in milliseconds.

  Type: int

  Required: No

- **SendSlack**

  Specifies the maximum send-queue slack due to queueing and/or congestion in milliseconds.

  Type: int

  Required: No

- **DecoySlack**

  Specifies the maximum decoy sweep slack due to external delays such as latency before a loop decoy packet will be considered lost.

  Type: int

  Required: No

- **ConnectTimeout**

  Specifies the maximum time a connection can take to establish a TCP/IP connection in milliseconds.

  Type: int

  Required: No

- **HandshakeTimeout**

Specifies the maximum time a connection can take for a link-protocol handshake in milliseconds.

Type: int

Required: No

- **ReauthInterval**

  Specifies the interval at which a connection will be reauthenticated in milliseconds.

  Type: int

  Required: No

- **SendDecoyTraffic**

  If **true**, decoy traffic is enabled. This parameter is experimental and untuned, and is disabled by default.

  Note

  This option will be removed once decoy traffic is fully implemented.

  Type: bool

  Required: No

- **DisableRateLimit**

  If **true**, the per-client rate limiter is disabled.

  Note

  This option should only be used for testing.

  Type: bool

  Required: No

- **GenerateOnly**

  If **true**, the server immediately halts and cleans up after long-term key generation.

  Type: bool

  Required: No

Configuring gateway nodes

The following configuration is drawn from the reference implementation in `katzenpost/docker/dirauth_mixnet/gateway1/katzenpost.toml`. In a real-world mixnet, the component hosts would not be sharing a single IP address. For more information about the test mixnet, see Using the Katzenpost Docker test network [https://katzenpost.network/docs/admin_guide/docker.html].

Table 4. Gateway node configuration sections

Gateway node: Server section
Gateway node: Logging section
Gateway node: Gateway section

Gateway node: PKI section
Gateway node: Management section
Gateway node: SphinxGeometry section
Gateway node: Debug section

Gateway node: Server section

The `Server` section configures mandatory basic parameters for each server node.

```
[Server]
    Identifier = "gateway1"
    WireKEM = "xwing"
    PKISignatureScheme = "Ed25519"
    Addresses = ["127.0.0.1:30004"]
    OnlyAdvertiseAltAddresses = false
    MetricsAddress = "127.0.0.1:30005"
    DataDir = "/dirauth_mixnet/gateway1"
    IsGatewayNode = true
    IsServiceNode = false
    [Server.AltAddresses]
        TCP = ["localhost:30004"]
```

- **Identifier**

  Specifies the human-readable identifier for a node, and must be unique per mixnet. The identifier can be an FQDN but does not have to be.

  Type: string

  Required: Yes

- **WireKEM**

  WireKEM specifies the key encapsulation mechanism (KEM) scheme for the PQ Noise [https://eprint.iacr.org/2022/539]-based wire protocol (link layer) that nodes use to communicate with each other. PQ Noise is a post-quantum variation of the Noise protocol framework [https://noiseprotocol.org/], which algebraically transforms ECDH handshake patterns into KEM encapsulate/decapsulate operations.

  This configuration option supports the optional use of hybrid post-quantum cryptography to strengthen security. The following KEM schemes are supported:

  - **Classical:** "x25519", "x448"

    ### Note

    X25519 and X448 are actually non-interactive key-exchanges (NIKEs), not KEMs. Katzenpost uses a hashed ElGamal cryptographic construction to convert them from NIKEs to KEMs.

  - **Post-quantum:** "mlkem768","sntrup4591761", "frodo640shake", "mceliece348864", "mceliece348864f", "mceliece460896", "mceliece460896f", "mceliece6688128", "mceliece6688128f", "mceliece6960119", "mceliece6960119f", "mceliece8192128", "mceliece8192128f", "CTIDH511", "CTIDH512", "CTIDH1024", "CTIDH2048",

  - **Hybrid post-quantum:** "xwing", "Kyber768-X25519", "MLKEM768-X25519", "MLKEM768-X448", "FrodoKEM-640-SHAKE-X448", "sntrup4591761-X448", "mceliece348864-X25519", "mceliece348864f-X25519", "mceliece460896-X25519", "mceliece460896f-

X25519", "mceliece6688128-X25519", "mceliece6688128f-X25519", "mceliece6960119-X25519", "mceliece6960119f-X25519", "mceliece8192128-X25519", "mceliece8192128f-X25519", "CTIDH512-X25519", "CTIDH512-X25519"

Type: string

Required: Yes

- **PKISignatureScheme**

  Specifies the cryptographic signature scheme that will be used by all components of the mix network when interacting with the PKI system. Mix nodes sign their descriptors using this signature scheme, and dirauth nodes similarly sign PKI documents using the same scheme.

  The following signature schemes are supported:

  - **Classical:** "ed25519", "ed448"
  - **Hybrid post-quantum:** "Ed25519 Sphincs+", "Ed448-Sphincs+", "Ed25519-Dilithium2", "Ed448-Dilithium3"

  Type: string

  Required: Yes

- **Addresses**

  Specifies a list of one or more address URLs in a format that contains the transport protocol, IP address, and port number that the server will bind to for incoming connections. Katzenpost supports URLs with that start with either "tcp://" or "quic://" such as: ["tcp://192.168.1.1:30001"] and ["quic://192.168.1.1:40001"].

  Type: []string

  Required: Yes

- **BindAddresses**

  If **true**, allows setting of listener addresses that the server will bind to and accept connections on. These addresses are not advertised in the PKI.

  Type: bool

  Required: No

- **MetricsAddress**

  Specifies the address/port to bind the Prometheus metrics endpoint to.

  Type: string

  Required: No

- **DataDir**

  Specifies the absolute path to a node's state directory. This is where persistence.db is written to disk and where a node stores its cryptographic key materials when started with the "-g" commmand-line option.

  Type: string

  Required: Yes

- **IsGatewayNode**

  If **true**, the server is a gateway node.

  Type: bool

  Required: No

- **IsServiceNode**

  If **true**, the server is a service node.

  Type: bool

  Required: No

Gateway node: Logging section

The `Logging` configuration section controls logging behavior across Katzenpost.

```
[Logging]
                Disable = false
                File = "katzenpost.log"
                Level = "INFO"
```

- **Disable**

  If **true**, logging is disabled.

  Type: bool

  Required: No

- **File**

  Specifies the log file. If omitted, `stdout` is used.

  An absolute or relative file path can be specified. A relative path is relative to the DataDir specified in the `Server` section of the configuration.

  Type: string

  Required: No

- **Level**

  Supported logging level values are ERROR | WARNING | NOTICE |INFO | DEBUG.

  Type: string

  Required: No

  ### Warning

  The DEBUG log level is unsafe for production use.

Gateway node: Gateway section

The `Gateway` section of the configuration is required for configuring a Gateway node. The section must contain `UserDB` and `SpoolDB` definitions. Bolt [https://github.com/boltdb/bolt] is an embedded database library for the Go programming language that Katzenpost has used in the past for its user and spool databases. Because Katzenpost currently persists data on Service nodes instead of Gateways, these databases will probably be deprecated in favour of in-memory concurrency structures. In the meantime, it remains necessary to configure a Gateway node as shown below, only changing the file paths as needed:

```
[Gateway]
    [Gateway.UserDB]
        Backend = "bolt"
            [Gateway.UserDB.Bolt]
                UserDB = "/dirauth_mixnet/gateway1/users.db"
    [Gateway.SpoolDB]
        Backend = "bolt"
            [Gateway.SpoolDB.Bolt]
                SpoolDB = "/dirauth_mixnet/gateway1/spool.db"
```

Gateway node: PKI section

The `PKI` section contains the directory authority configuration for a mix, gateway, or service node.

```
[PKI]
[PKI.dirauth]

    [[PKI.dirauth.Authorities]]
        Identifier = "auth1"
        IdentityPublicKey = """-----BEGIN ED25519 PUBLIC KEY-----
tqN6tpOVotHWXKCszVn2kS7vAZjQpvJjQF3Qz/Qwhyg=
-----END ED25519 PUBLIC KEY-----
"""
        PKISignatureScheme = "Ed25519"
        LinkPublicKey = """-----BEGIN XWING PUBLIC KEY-----
JnJ8ztQEIjAkKJcpuZvJAdkWjBim/5G5d8yoosEQHeGJeeBqNPdm2AitUbpiQPcd
tNCo9DxuC9IeqmsfwOYpV6AtOOsaInA6QnHDYcuBfZcQL5MU4+t2TzpBZQYlrSED
hPCKrAG+8GEUl6akseG371WQzEtPpEWWCJCJOiS/VDFZT7eKrldlumN6gfiB84sR
...
arFh/WKwYJUj+aGBsFYSqGdzC6MdY4x/YyFe2ze0MJEjThQE91y1d/LCQ3Sb7Ri+
u6PBi3JU2qzlPEejDKwK0t5tMNEAkq8iNrpRTdD/hS0gR+ZIN8Z9QKh7Xf94FWG2
H+r8OaqImQhgHabrWRDyLg==
-----END XWING PUBLIC KEY-----
"""
        WireKEMScheme = "xwing"
        Addresses = ["127.0.0.1:30001"]

    [[PKI.dirauth.Authorities]]
        Identifier = "auth2"
        IdentityPublicKey = """-----BEGIN ED25519 PUBLIC KEY-----
O51Ty2WLu4C1ETMa29s03bMXV72gnjJfTfwLV++LVBI=
-----END ED25519 PUBLIC KEY-----
"""
        PKISignatureScheme = "Ed25519"
        LinkPublicKey = """-----BEGIN XWING PUBLIC KEY-----
TtQkg2XKUnY602FFBaPJ+zpN0Twy20cwyyFxh7FNUjaXA9MAJXs0vUwFbJc6BjYv
f+olKnlIKFSmDvcF74U6w1F0ObugwTNKNxeYKPKhX4FiencUbRwkHoYHdtZdSctz
TKy08qKQyCAccqCRpdo6ZtYXPAU+2rthjYTOL7Zn+7SHUKCuJClcPnvEYjVcJxtZ
...
ubJIe5U4nMJbBkOqr7Kq6niaEkiLODa0tkpB8tKMYTMBdcYyHSXCzpo7U9sb6LAR
HktiTBDtRXviu2vbw7VRXhkMW2kjYZDtReQ5sAse04DvmD49zgTp1YxYW+wWFaL3
37X7/SNuLdHX4PHZXIWHBQ==
-----END XWING PUBLIC KEY-----
"""
        WireKEMScheme = "xwing"
        Addresses = ["127.0.0.1:30002"]

    [[PKI.dirauth.Authorities]]
        Identifier = "auth3"
        IdentityPublicKey = """-----BEGIN ED25519 PUBLIC KEY-----
zQvydRYJq3npeLcg1NqIf+SswEKE5wFmiwNsI9Z1whQ=
-----END ED25519 PUBLIC KEY-----
"""
        PKISignatureScheme = "Ed25519"
        LinkPublicKey = """
-----BEGIN XWING PUBLIC KEY-----
OYK9FiC53xwZ1VST3jDOO4tR+cUMSVRSekmigZMChSjDCPZbKut8TblxtlUfc/yi
Ugorz4NIvYPMWUt3QPwS2UWq8/HMWXNGPUiAevg12+oV+jOJXaJeCfY24UekJnSw
TNcdGaFZFSR0FocFcPBBnrK1M2B8w8eEUKQIsXRDM3x/8aRIuDif+ve8rSwpgKeh
...
OdVD3yw7OOS8uPZLORGQFyJbHtVmFPVvwja4G/o2gntAoHUZ2LiJJakpVhhlSyrI
yuzvwwFtZVfWtNb5gAKZCyg0aduR3qgd7MPerRF+YopZk3OCRpC02YxfUZrHv398
FZWJFK0R8iU52CEUxVpXTA==
-----END XWING PUBLIC KEY-----
"""
        WireKEMScheme = "xwing"
        Addresses = ["127.0.0.1:30003"]
```

- **Identifier**

  Specifies the human-readable identifier for a node, which must be unique per mixnet. The identifier can be an FQDN but does not have to be.

  Type: string

  Required: Yes

- **IdentityPublicKey**

  String containing the node's public identity key in PEM format. `IdentityPublicKey` is the node's permanent identifier and is used to verify cryptographic signatures produced by its private identity key.

  Type: string

  Required: Yes

- **PKISignatureScheme**

  Specifies the cryptographic signature scheme that will be used by all components of the mix network when interacting with the PKI system. Mix nodes sign their descriptors using this signature scheme, and dirauth nodes similarly sign PKI documents using the same scheme.

  Type: string

  Required: Yes

- **LinkPublicKey**

  String containing the peer's public link-layer key in PEM format. `LinkPublicKey` must match the specified `WireKEMScheme`.

  Type: string

Required: Yes

- **WireKEMScheme**

  The name of the wire protocol key-encapsulation mechanism (KEM) to use.

  Type: string

  Required: Yes

- **Addresses**

  Specifies a list of one or more address URLs in a format that contains the transport protocol, IP address, and port number that the server will bind to for incoming connections. Katzenpost supports URLs that start with either "tcp://" or "quic://" such as: ["tcp://192.168.1.1:30001"] and ["quic://192.168.1.1:40001"].

  Type: []string

  Required: Yes

Gateway node: Management section

The `Management` section specifies connectivity information for the Katzenpost control protocol which can be used to make run-time configuration changes. A configuration resembles the following:

```
[Management]
   Enable = false
   Path = "/dirauth_mixnet/mix1/management_sock"
```

- **Enable**

  If **true**, the management interface is enabled.

  Type: bool

  Required: No

- **Path**

  Specifies the path to the management interface socket. If left empty, then `management_sock` is located in the configuration's defined `DataDir`>.

  Type: string

  Required: No

Gateway node: SphinxGeometry section

The `SphinxGeometry` section defines parameters for the Sphinx encrypted nested-packet format used internally by Katzenpost.

Warning

The values in the `SphinxGeometry` configuration section must be programmatically generated by **gensphinx**. Many of the parameters are interdependent and cannot be individually modified. Do not modify the these values by hand.

The settings in this section are generated by the **gensphinx** utility, which computes the Sphinx geometry based on the following user-supplied directives:

- The number of mix node layers (not counting gateway and service nodes)
- The length of the application-usable packet payload
- The selected NIKE or KEM scheme

The output in TOML should then be pasted unchanged into the node's configuration file, as shown below. For more information, see Appendix: Using *gensphinx* [https://katzenpost.network/docs/admin_guide/gensphinx.html].

```
[SphinxGeometry]
                PacketLength = 3082
                NrHops = 5
                HeaderLength = 476
                RoutingInfoLength = 410
                PerHopRoutingInfoLength = 82
                SURBLength = 572
                SphinxPlaintextHeaderLength = 2
                PayloadTagLength = 32
                ForwardPayloadLength = 2574
                UserForwardPayloadLength = 2000
                NextNodeHopLength = 65
                SPRPKeyMaterialLength = 64
                NIKEName = "x25519"
                KEMName = ""
```

- **PacketLength**

  The length of a Sphinx packet in bytes.

  Type: int

  Required: Yes

- **NrHops**

  The number of hops a Sphinx packet takes through the mixnet. Because packet headers hold destination information for each hop, the size of the header increases linearly with the number of hops.

  Type: int

  Required: Yes

- **HeaderLength**

  The total length of the Sphinx packet header in bytes.

  Type: int

  Required: Yes

- **RoutingInfoLength**

  The total length of the routing information portion of the Sphinx packet header.

  Type: int

  Required: Yes

- **PerHopRoutingInfoLength**

  The length of the per-hop routing information in the Sphinx packet header.

Type: int

Required: Yes

- **SURBLength**

  The length of a single-use reply block (SURB).

  Type: int

  Required: Yes

- **SphinxPlaintextHeaderLength**

  The length of the plaintext Sphinx packet header.

  Type: int

  Required: Yes

- **PayloadTagLength**

  The length of the payload tag.

  Type: int

  Required: Yes

- **ForwardPayloadLength**

  The total size of the payload.

  Type: int

  Required: Yes

- **UserForwardPayloadLength**

  The size of the usable payload.

  Type: int

  Required: Yes

- **NextNodeHopLength**

  The `NextNodeHopLength` is derived from the largest routing-information block that we expect to encounter. Other packets have `NextNodeHop` + `NodeDelay` sections, or a `Recipient` section, both of which are shorter.

  Type: int

  Required: Yes

- **SPRPKeyMaterialLength**

  The length of the strong pseudo-random permutation (SPRP) key.

  Type: int

  Required: Yes

- **NIKEName**

The name of the non-interactive key exchange (NIKE) scheme used by Sphinx packets.

`NIKEName` and `KEMName` are mutually exclusive.

Type: string

Required: Yes

- **KEMName**

  The name of the key encapsulation mechanism (KEM) used by Sphinx packets.

  `NIKEName` and `KEMName` are mutually exclusive.

  Type: string

  Required: Yes

Gateway node: Debug section

The `Debug` section is the Katzenpost server debug configuration for advanced tuning.

```
[Debug]
                NumSphinxWorkers = 16
                NumServiceWorkers = 3
                NumGatewayWorkers = 3
                NumKaetzchenWorkers = 3
                SchedulerExternalMemoryQueue = false
                SchedulerQueueSize = 0
                SchedulerMaxBurst = 16
                UnwrapDelay = 250
                GatewayDelay = 500
                ServiceDelay = 500
                KaetzchenDelay = 750
                SchedulerSlack = 150
                SendSlack = 50
                DecoySlack = 15000
                ConnectTimeout = 60000
                HandshakeTimeout = 30000
                ReauthInterval = 30000
                SendDecoyTraffic = false
                DisableRateLimit = false
                GenerateOnly = false
```

- **NumSphinxWorkers**

  Specifies the number of worker instances to use for inbound Sphinx packet processing.

  Type: int

  Required: No

- **NumProviderWorkers**

  Specifies the number of worker instances to use for provider specific packet processing.

  Type: int

  Required: No

- **NumKaetzchenWorkers**

  Specifies the number of worker instances to use for Kaetzchen-specific packet processing.

  Type: int

  Required: No

- **SchedulerExternalMemoryQueue**

  If **true**, the experimental disk-backed external memory queue is enabled.

  Type: bool

  Required: No

- **SchedulerQueueSize**

  Specifies the maximum scheduler queue size before random entries will start getting dropped. A value less than or equal to zero is treated as unlimited.

  Type: int

  Required: No

- **SchedulerMaxBurst**

  Specifies the maximum number of packets that will be dispatched per scheduler wakeup event.

  Type:

  Required: No

- **UnwrapDelay**

  Specifies the maximum unwrap delay due to queueing in milliseconds.

  Type: int

  Required: No

- **GatewayDelay**

  Specifies the maximum gateway node worker delay due to queueing in milliseconds.

  Type: int

  Required: No

- **ServiceDelay**

  Specifies the maximum provider delay due to queueing in milliseconds.

  Type: int

  Required: No

- **KaetzchenDelay**

  Specifies the maximum kaetzchen delay due to queueing in milliseconds.

  Type: int

  Required: No

- **SchedulerSlack**

  Specifies the maximum scheduler slack due to queueing and/or processing in milliseconds.

  Type: int

  Required: No

- **SendSlack**

  Specifies the maximum send-queue slack due to queueing and/or congestion in milliseconds.

  Type: int

  Required: No

- **DecoySlack**

  Specifies the maximum decoy sweep slack due to external delays such as latency before a loop decoy packet will be considered lost.

  Type: int

  Required: No

- **ConnectTimeout**

  Specifies the maximum time a connection can take to establish a TCP/IP connection in milliseconds.

  Type: int

  Required: No

- **HandshakeTimeout**

  Specifies the maximum time a connection can take for a link-protocol handshake in milliseconds.

  Type: int

  Required: No

- **ReauthInterval**

  Specifies the interval at which a connection will be reauthenticated in milliseconds.

  Type: int

  Required: No

- **SendDecoyTraffic**

  If **true**, decoy traffic is enabled. This parameter is experimental and untuned, and is disabled by default.

  Note

  This option will be removed once decoy traffic is fully implemented.

  Type: bool

  Required: No

- **DisableRateLimit**

If **true**, the per-client rate limiter is disabled.

Note

This option should only be used for testing.

Type: bool

Required: No

- **GenerateOnly**

  If **true**, the server immediately halts and cleans up after long-term key generation.

  Type: bool

  Required: No

Configuring service nodes

The following configuration is drawn from the reference implementation in `katzenpost/docker/dirauth_mixnet/servicenode1/authority.toml`. In a real-world mixnet, the component hosts would not be sharing a single IP address. For more information about the test mixnet, see Using the Katzenpost Docker test network [https://katzenpost.network/docs/admin_guide/docker.html].

Table 5. Mix node configuration sections

Service node: Server section
Service node: Logging section
Service node: ServiceNode section
Service node: PKI section
Service node: Management section
Service node: SphinxGeometry section
Service node: Debug section

Service node: Server section

The `Server` section configures mandatory basic parameters for each server node.

```
[Server]
    Identifier = "servicenode1"
    WireKEM = "xwing"
    PKISignatureScheme = "Ed25519"
    Addresses = ["127.0.0.1:30006"]
    OnlyAdvertiseAltAddresses = false
    MetricsAddress = "127.0.0.1:30007"
    DataDir = "/dirauth_mixnet/servicenode1"
    IsGatewayNode = false
    IsServiceNode = true
    [Server.AltAddresses]
```

- **Identifier**

  Specifies the human-readable identifier for a node, and must be unique per mixnet. The identifier can be an FQDN but does not have to be.

Type: string

Required: Yes

- **WireKEM**

  WireKEM specifies the key encapsulation mechanism (KEM) scheme for the PQ Noise [https://eprint.iacr.org/2022/539]-based wire protocol (link layer) that nodes use to communicate with each other. PQ Noise is a post-quantum variation of the Noise protocol framework [https://noiseprotocol.org/], which algebraically transforms ECDH handshake patterns into KEM encapsulate/decapsulate operations.

  This configuration option supports the optional use of hybrid post-quantum cryptography to strengthen security. The following KEM schemes are supported:

  - **Classical:** "x25519", "x448"

    ## Note

    X25519 and X448 are actually non-interactive key-exchanges (NIKEs), not KEMs. Katzenpost uses a hashed ElGamal cryptographic construction to convert them from NIKEs to KEMs.

  - **Post-quantum:** "mlkem768","sntrup4591761", "frodo640shake", "mceliece348864", "mceliece348864f", "mceliece460896", "mceliece460896f", "mceliece6688128", "mceliece6688128f", "mceliece6960119", "mceliece6960119f", "mceliece8192128", "mceliece8192128f", "CTIDH511", "CTIDH512", "CTIDH1024", "CTIDH2048",

  - **Hybrid post-quantum:** "xwing", "Kyber768-X25519", "MLKEM768-X25519", "MLKEM768-X448", "FrodoKEM-640-SHAKE-X448", "sntrup4591761-X448", "mceliece348864-X25519", "mceliece348864f-X25519", "mceliece460896-X25519", "mceliece460896f-X25519", "mceliece6688128-X25519", "mceliece6688128f-X25519", "mceliece6960119-X25519", "mceliece6960119f-X25519", "mceliece8192128-X25519", "mceliece8192128f-X25519", "CTIDH512-X25519", "CTIDH512-X25519"

  Type: string

  Required: Yes

- **PKISignatureScheme**

  Specifies the cryptographic signature scheme that will be used by all components of the mix network when interacting with the PKI system. Mix nodes sign their descriptors using this signature scheme, and dirauth nodes similarly sign PKI documents using the same scheme.

  The following signature schemes are supported:

  - **Classical:** "ed25519", "ed448"

  - **Hybrid post-quantum:** "Ed25519 Sphincs+", "Ed448-Sphincs+", "Ed25519-Dilithium2", "Ed448-Dilithium3"

  Type: string

  Required: Yes

- **Addresses**

  Specifies a list of one or more address URLs in a format that contains the transport protocol, IP address, and port number that the server will bind to for incoming connections. Katzenpost supports URLs with that start with either "tcp://" or "quic://" such as: ["tcp://192.168.1.1:30001"] and ["quic://192.168.1.1:40001"].

Type: []string

Required: Yes

- **BindAddresses**

  If **true**, allows setting of listener addresses that the server will bind to and accept connections on. These addresses are not advertised in the PKI.

  Type: bool

  Required: No

- **MetricsAddress**

  Specifies the address/port to bind the Prometheus metrics endpoint to.

  Type: string

  Required: No

- **DataDir**

  Specifies the absolute path to a node's state directory. This is where persistence.db is written to disk and where a node stores its cryptographic key materials when started with the "-g" commmand-line option.

  Type: string

  Required: Yes

- **IsGatewayNode**

  If **true**, the server is a gateway node.

  Type: bool

  Required: No

- **IsServiceNode**

  If **true**, the server is a service node.

  Type: bool

  Required: No

Service node: Logging section

The `Logging` configuration section controls logging behavior across Katzenpost.

```
[Logging]
                Disable = false
                File = "katzenpost.log"
                Level = "INFO"
```

- **Disable**

  If **true**, logging is disabled.

Type: bool

Required: No

- **File**

  Specifies the log file. If omitted, `stdout` is used.

  An absolute or relative file path can be specified. A relative path is relative to the DataDir specified in the `Server` section of the configuration.

  Type: string

  Required: No

- **Level**

  Supported logging level values are ERROR | WARNING | NOTICE |INFO | DEBUG.

  Type: string

  Required: No

  Warning

  The DEBUG log level is unsafe for production use.

Service node: ServiceNode section

The `ServiceNode` section contains configurations for each network service that Katzenpost supports.

Services, termed Kaetzchen [https://github.com/katzenpost/katzenpost/blob/main/docs/Specificatons/pdf/kaetzchen.pdf], can be divided into built-in and external services. External services are provided through the CBORPlugin [https://pkg.go.dev/github.com/katzenpost/katzenpost@v0.0.35/server/cborplugin#ResponseFactory], a Go programming language implementation of the Concise Binary Object Representation (CBOR) [https://datatracker.ietf.org/doc/html/rfc8949], a binary data serialization format. While native services need simply to be activated, external services are invoked by a separate command and connected to the mixnet over a Unix socket. The plugin allows mixnet services to be added in any programming language.

```
[ServiceNode]

    [[ServiceNode.Kaetzchen]]
        Capability = "echo"
        Endpoint = "+echo"
        Disable = false

    [[ServiceNode.CBORPluginKaetzchen]]
        Capability = "spool"
        Endpoint = "+spool"
        Command = "/dirauth_mixnet/memspool.alpine"
        MaxConcurrency = 1
        Disable = false
        [ServiceNode.CBORPluginKaetzchen.Config]
            data_store = "/dirauth_mixnet/servicenode1/memspool.storage"
            log_dir = "/dirauth_mixnet/servicenode1"

    [[ServiceNode.CBORPluginKaetzchen]]
```

```
        Capability = "pigeonhole"
        Endpoint = "+pigeonhole"
        Command = "/dirauth_mixnet/pigeonhole.alpine"
        MaxConcurrency = 1
        Disable = false
        [ServiceNode.CBORPluginKaetzchen.Config]
            db = "/dirauth_mixnet/servicenode1/map.storage"
            log_dir = "/dirauth_mixnet/servicenode1"

    [[ServiceNode.CBORPluginKaetzchen]]
        Capability = "panda"
        Endpoint = "+panda"
        Command = "/dirauth_mixnet/panda_server.alpine"
        MaxConcurrency = 1
        Disable = false
        [ServiceNode.CBORPluginKaetzchen.Config]
            fileStore = "/dirauth_mixnet/servicenode1/panda.storage"
            log_dir = "/dirauth_mixnet/servicenode1"
            log_level = "INFO"

    [[ServiceNode.CBORPluginKaetzchen]]
        Capability = "http"
        Endpoint = "+http"
        Command = "/dirauth_mixnet/proxy_server.alpine"
        MaxConcurrency = 1
        Disable = false
        [ServiceNode.CBORPluginKaetzchen.Config]
            host = "localhost:4242"
            log_dir = "/dirauth_mixnet/servicenode1"
            log_level = "DEBUG"
```

Common parameters:

- **Capability**

  Specifies the protocol capability exposed by the agent.

  Type: string

  Required: Yes

- **Endpoint**

  Specifies the provider-side Endpoint where the agent will accept requests. While not required by the specification, this server only supports Endpoints that are lower-case local parts of an email address.

  Type: string

  Required: Yes

- **Command**

  Specifies the full path to the external plugin program that implements this `Kaetzchen` service.

  Type: string

  Required: Yes

- **MaxConcurrency**

  Specifies the number of worker goroutines to start for this service.

Type: int

Required: Yes

- **Config**

  Specifies extra per-agent arguments to be passed to the agent's initialization routine.

  Type: map[string]interface{}

  Required: Yes

- **Disable**

  If **true**, disables a configured agent.

  Type: bool

  Required: No

Per-service parameters:

- **echo**

  The internal `echo` service must be enabled on every service node of a production mixnet for decoy traffic to work properly.

- **spool**

  The `spool` service supports the `catshadow` storage protocol, which is required by the Katzen chat client. The example configuration above shows spool enabled with the setting:

  ```
  Disable = false
  ```

  ### Note

  `Spool`, properly `memspool`, should not be confused with the spool database on gateway nodes.

  - **data_store**

    Specifies the full path to the service database file.

    Type: string

    Required: Yes

  - **log_dir**

    Specifies the path to the node's log directory.

    Type: string

    Required: Yes

- **pigeonhole**

  The `pigeonhole` courier service supports the Blinding-and-Capability scheme (BACAP)-based unlinkable messaging protocols detailed in **Place-holder for research paper link**. Most of our future protocols will use the `pigeonhole` courier service.

  - **db**

Specifies the full path to the service database file.

Type: string

Required: Yes

- **log_dir**

  Specifies the path to the node's log directory.

  Type: string

  Required: Yes

- **panda**

  The `panda` storage and authentication service currently does not work properly.

  - **fileStore**

    Specifies the full path to the service database file.

    Type: string

    Required: Yes

  - **log_dir**

    Specifies the path to the node's log directory.

    Type: string

    Required: Yes

  - **log_level**

    Supported values are ERROR | WARNING | NOTICE |INFO | DEBUG.

    Warning

    The DEBUG log level is unsafe for production use.

    Type: string

    Required: Yes

    Required: Yes

- **http**

  The `http` service is completely optional, but allows the mixnet to be used as an HTTP proxy. This may be useful for integrating with existing software systems.

  - **host**

    The host name and TCP port of the service.

    Type: string

    Required: Yes

  - **log_dir**

Specifies the path to the node's log directory.

Type: string

Required: Yes

- **log_level**

  Supported values are ERROR | WARNING | NOTICE |INFO | DEBUG.

  Type: string

  Required: Yes

  Required: Yes

  Warning

  The DEBUG log level is unsafe for production use.

  Type: string

  Required: Yes

Service node: PKI section

The `PKI` section contains the directory authority configuration for a mix, gateway, or service node.

```
[PKI]
[PKI.dirauth]

    [[PKI.dirauth.Authorities]]
        Identifier = "auth1"
        IdentityPublicKey = """-----BEGIN ED25519 PUBLIC KEY-----
tqN6tpOVotHWXKCszVn2kS7vAZjQpvJjQF3Qz/Qwhyg=
-----END ED25519 PUBLIC KEY-----
"""
        PKISignatureScheme = "Ed25519"
        LinkPublicKey = """-----BEGIN XWING PUBLIC KEY-----
JnJ8ztQEIjAkKJcpuZvJAdkWjBim/5G5d8yoosEQHeGJeeBqNPdm2AitUbpiQPcd
tNCo9DxuC9Ieqmsfw0YpV6AtOOsaInA6QnHDYcuBfZcQL5MU4+t2TzpBZQYlrSED
hPCKrAG+8GEUl6akseG371WQzEtPpEWWCJCJOiS/VDFZT7eKrldlumN6gfiB84sR
...
arFh/WKwYJUj+aGBsFYSqGdzC6MdY4x/YyFe2ze0MJEjThQE91y1d/LCQ3Sb7Ri+
u6PBi3JU2qzlPEejDKwK0t5tMNEAkq8iNrpRTdD/hS0gR+ZIN8Z9QKh7Xf94FWG2
H+r8OaqImQhgHabrWRDyLg==
-----END XWING PUBLIC KEY-----
"""
        WireKEMScheme = "xwing"
        Addresses = ["127.0.0.1:30001"]

    [[PKI.dirauth.Authorities]]
        Identifier = "auth2"
        IdentityPublicKey = """-----BEGIN ED25519 PUBLIC KEY-----
O51Ty2WLu4C1ETMa29s03bMXV72gnjJfTfwLV++LVBI=
-----END ED25519 PUBLIC KEY-----
"""
        PKISignatureScheme = "Ed25519"
        LinkPublicKey = """-----BEGIN XWING PUBLIC KEY-----
TtQkg2XKUnY602FFBaPJ+zpN0Twy20cwyyFxh7FNUjaXA9MAJXs0vUwFbJc6BjYv
f+olKnlIKFSmDvcF74U6w1F0ObugwTNKNxeYKPKhX4FiencUbRwkHoYHdtZdSctz
TKy08qKQyCAccqCRpdo6ZtYXPAU+2rthjYTOL7Zn+7SHUKCuJClcPnvEYjVcJxtZ
...
ubJIe5U4nMJbBkOqr7Kq6niaEkiLODa0tkpB8tKMYTMBdcYyHSXCzpo7U9sb6LAR
HktiTBDtRXviu2vbw7VRXhkMW2kjYZDtReQ5sAse04DvmD49zgTp1YxYW+wWFaL3
37X7/SNuLdHX4PHZXIWHBQ==
-----END XWING PUBLIC KEY-----
"""
        WireKEMScheme = "xwing"
        Addresses = ["127.0.0.1:30002"]

    [[PKI.dirauth.Authorities]]
        Identifier = "auth3"
        IdentityPublicKey = """-----BEGIN ED25519 PUBLIC KEY-----
zQvydRYJq3npeLcg1NqIf+SswEKE5wFmiwNsI9Z1whQ=
-----END ED25519 PUBLIC KEY-----
"""
        PKISignatureScheme = "Ed25519"
        LinkPublicKey = """
-----BEGIN XWING PUBLIC KEY-----
OYK9FiC53xwZ1VST3jDOO4tR+cUMSVRSekmigZMChSjDCPZbKut8TblxtlUfc/yi
Ugorz4NIvYPMWUt3QPwS2UWq8/HMWXNGPUiAevg12+oV+jOJXaJeCfY24UekJnSw
TNcdGaFZFSR0FocFcPBBnrK1M2B8w8eEUKQIsXRDM3x/8aRIuDif+ve8rSwpgKeh
...
OdVD3yw7OOS8uPZLORGQFyJbHtVmFPVvwja4G/o2gntAoHUZ2LiJJakpVhhlSyrI
yuzvwwFtZVfWtNb5gAKZCyg0aduR3qgd7MPerRF+YopZk3OCRpC02YxfUZrHv398
FZWJFK0R8iU52CEUxVpXTA==
-----END XWING PUBLIC KEY-----
"""
        WireKEMScheme = "xwing"
        Addresses = ["127.0.0.1:30003"]
```

- **Identifier**

  Specifies the human-readable identifier for a node, which must be unique per mixnet. The identifier can be an FQDN but does not have to be.

  Type: string

  Required: Yes

- **IdentityPublicKey**

  String containing the node's public identity key in PEM format. `IdentityPublicKey` is the node's permanent identifier and is used to verify cryptographic signatures produced by its private identity key.

  Type: string

  Required: Yes

- **PKISignatureScheme**

  Specifies the cryptographic signature scheme that will be used by all components of the mix network when interacting with the PKI system. Mix nodes sign their descriptors using this signature scheme, and dirauth nodes similarly sign PKI documents using the same scheme.

  Type: string

Required: Yes

- **LinkPublicKey**

  String containing the peer's public link-layer key in PEM format. `LinkPublicKey` must match the specified `WireKEMScheme`.

  Type: string

  Required: Yes

- **WireKEMScheme**

  The name of the wire protocol key-encapsulation mechanism (KEM) to use.

  Type: string

  Required: Yes

- **Addresses**

  Specifies a list of one or more address URLs in a format that contains the transport protocol, IP address, and port number that the server will bind to for incoming connections. Katzenpost supports URLs that start with either "tcp://" or "quic://" such as: ["tcp://192.168.1.1:30001"] and ["quic://192.168.1.1:40001"].

  Type: []string

  Required: Yes

Service node: Management section

The `Management` section specifies connectivity information for the Katzenpost control protocol which can be used to make run-time configuration changes. A configuration resembles the following:

```
[Management]
   Enable = false
   Path = "/dirauth_mixnet/mix1/management_sock"
```

- **Enable**

  If **true**, the management interface is enabled.

  Type: bool

  Required: No

- **Path**

  Specifies the path to the management interface socket. If left empty, then `management_sock` is located in the configuration's defined `DataDir`>.

  Type: string

  Required: No

Service node: SphinxGeometry section

The `SphinxGeometry` section defines parameters for the Sphinx encrypted nested-packet format used internally by Katzenpost.

Warning

The values in the `SphinxGeometry` configuration section must be programmatically generated by **gensphinx**. Many of the parameters are interdependent and cannot be individually modified. Do not modify the these values by hand.

The settings in this section are generated by the **gensphinx** utility, which computes the Sphinx geometry based on the following user-supplied directives:

- The number of mix node layers (not counting gateway and service nodes)
- The length of the application-usable packet payload
- The selected NIKE or KEM scheme

The output in TOML should then be pasted unchanged into the node's configuration file, as shown below. For more information, see Appendix: Using *gensphinx* [https://katzenpost.network/docs/admin_guide/gensphinx.html].

```
[SphinxGeometry]
                PacketLength = 3082
                NrHops = 5
                HeaderLength = 476
                RoutingInfoLength = 410
                PerHopRoutingInfoLength = 82
                SURBLength = 572
                SphinxPlaintextHeaderLength = 2
                PayloadTagLength = 32
                ForwardPayloadLength = 2574
                UserForwardPayloadLength = 2000
                NextNodeHopLength = 65
                SPRPKeyMaterialLength = 64
                NIKEName = "x25519"
                KEMName = ""
```

- **PacketLength**

  The length of a Sphinx packet in bytes.

  Type: int

  Required: Yes

- **NrHops**

  The number of hops a Sphinx packet takes through the mixnet. Because packet headers hold destination information for each hop, the size of the header increases linearly with the number of hops.

  Type: int

  Required: Yes

- **HeaderLength**

  The total length of the Sphinx packet header in bytes.

  Type: int

  Required: Yes

- **RoutingInfoLength**

The total length of the routing information portion of the Sphinx packet header.

Type: int

Required: Yes

- **PerHopRoutingInfoLength**

  The length of the per-hop routing information in the Sphinx packet header.

  Type: int

  Required: Yes

- **SURBLength**

  The length of a single-use reply block (SURB).

  Type: int

  Required: Yes

- **SphinxPlaintextHeaderLength**

  The length of the plaintext Sphinx packet header.

  Type: int

  Required: Yes

- **PayloadTagLength**

  The length of the payload tag.

  Type: int

  Required: Yes

- **ForwardPayloadLength**

  The total size of the payload.

  Type: int

  Required: Yes

- **UserForwardPayloadLength**

  The size of the usable payload.

  Type: int

  Required: Yes

- **NextNodeHopLength**

  The `NextNodeHopLength` is derived from the largest routing-information block that we expect to encounter. Other packets have `NextNodeHop` + `NodeDelay` sections, or a `Recipient` section, both of which are shorter.

  Type: int

  Required: Yes

- **SPRPKeyMaterialLength**

  The length of the strong pseudo-random permutation (SPRP) key.

  Type: int

  Required: Yes

- **NIKEName**

  The name of the non-interactive key exchange (NIKE) scheme used by Sphinx packets.

  `NIKEName` and `KEMName` are mutually exclusive.

  Type: string

  Required: Yes

- **KEMName**

  The name of the key encapsulation mechanism (KEM) used by Sphinx packets.

  `NIKEName` and `KEMName` are mutually exclusive.

  Type: string

  Required: Yes

Service node: Debug section

The `Debug` section is the Katzenpost server debug configuration for advanced tuning.

```
[Debug]
                NumSphinxWorkers = 16
                NumServiceWorkers = 3
                NumGatewayWorkers = 3
                NumKaetzchenWorkers = 3
                SchedulerExternalMemoryQueue = false
                SchedulerQueueSize = 0
                SchedulerMaxBurst = 16
                UnwrapDelay = 250
                GatewayDelay = 500
                ServiceDelay = 500
                KaetzchenDelay = 750
                SchedulerSlack = 150
                SendSlack = 50
                DecoySlack = 15000
                ConnectTimeout = 60000
                HandshakeTimeout = 30000
                ReauthInterval = 30000
                SendDecoyTraffic = false
                DisableRateLimit = false
                GenerateOnly = false
```

- **NumSphinxWorkers**

  Specifies the number of worker instances to use for inbound Sphinx packet processing.

  Type: int

Required: No

- **NumProviderWorkers**

  Specifies the number of worker instances to use for provider specific packet processing.

  Type: int

  Required: No

- **NumKaetzchenWorkers**

  Specifies the number of worker instances to use for Kaetzchen-specific packet processing.

  Type: int

  Required: No

- **SchedulerExternalMemoryQueue**

  If **true**, the experimental disk-backed external memory queue is enabled.

  Type: bool

  Required: No

- **SchedulerQueueSize**

  Specifies the maximum scheduler queue size before random entries will start getting dropped. A value less than or equal to zero is treated as unlimited.

  Type: int

  Required: No

- **SchedulerMaxBurst**

  Specifies the maximum number of packets that will be dispatched per scheduler wakeup event.

  Type:

  Required: No

- **UnwrapDelay**

  Specifies the maximum unwrap delay due to queueing in milliseconds.

  Type: int

  Required: No

- **GatewayDelay**

  Specifies the maximum gateway node worker delay due to queueing in milliseconds.

  Type: int

  Required: No

- **ServiceDelay**

  Specifies the maximum provider delay due to queueing in milliseconds.

Type: int

Required: No

- **KaetzchenDelay**

  Specifies the maximum kaetzchen delay due to queueing in milliseconds.

  Type: int

  Required: No

- **SchedulerSlack**

  Specifies the maximum scheduler slack due to queueing and/or processing in milliseconds.

  Type: int

  Required: No

- **SendSlack**

  Specifies the maximum send-queue slack due to queueing and/or congestion in milliseconds.

  Type: int

  Required: No

- **DecoySlack**

  Specifies the maximum decoy sweep slack due to external delays such as latency before a loop decoy packet will be considered lost.

  Type: int

  Required: No

- **ConnectTimeout**

  Specifies the maximum time a connection can take to establish a TCP/IP connection in milliseconds.

  Type: int

  Required: No

- **HandshakeTimeout**

  Specifies the maximum time a connection can take for a link-protocol handshake in milliseconds.

  Type: int

  Required: No

- **ReauthInterval**

  Specifies the interval at which a connection will be reauthenticated in milliseconds.

  Type: int

  Required: No

- **SendDecoyTraffic**

If **true**, decoy traffic is enabled. This parameter is experimental and untuned, and is disabled by default.

Note

This option will be removed once decoy traffic is fully implemented.

Type: bool

Required: No

- **DisableRateLimit**

  If **true**, the per-client rate limiter is disabled.

  ### Note

  This option should only be used for testing.

  Type: bool

  Required: No

- **GenerateOnly**

  If **true**, the server immediately halts and cleans up after long-term key generation.

  Type: bool

  Required: No

Using the Katzenpost Docker test network

Katzenpost provides a ready-to-deploy Docker image for developers who need a non-production test environment for developing and testing client applications and server side plugins. By running this image on a single computer, you avoid the need to build and manage a complex multi-node mix net. The image can also be run using Podman [https://podman.io/]

The test mix network includes the following components:

- Three directory authority (PKI [https://katzenpost.network/docs/specs/pki/]) nodes
- Six mix [https://katzenpost.network/docs/specs/mixnet/] nodes, including one node serving also as both gateway and service provider
- A ping utility, **run-ping**

Requirements

Before running the Katzenpost docker image, make sure that the following software is installed.

- A Debian GNU Linux [https://debian.org] or Ubuntu [https://ubuntu.com] system
- Git [https://git-scm.com/]
- Go [https://go.dev/]
- GNU Make [https://www.gnu.org/software/make/]
- Prometheus [https://prometheus.io/docs/introduction/overview/]
- Docker [https://www.docker.com], Docker Compose [https://docs.docker.com/compose/], and (optionally) Podman [https://podman.io]

Note

If both Docker and Podman are present on your system, Katzenpost uses Podman. Podman is a drop-in daemonless equivalent to Docker that does not require superuser privileges to run.

On Debian, these software requirements can be installed with the following commands (running as superuser). **Apt** will pull in the needed dependencies.

```
# apt update
# apt install git golang make docker docker-compose podman
```

Preparing to run the container image

Complete the following procedure to obtain, build, and deploy the Katzenpost test network.

1. Install the Katzenpost code repository, hosted at `https://github.com/katzenpost`. The main Katzenpost repository contains code for the server components as well as the docker image. Clone the repository with the following command (your directory location may vary):

   ```
   ~$ git clone https://github.com/katzenpost/katzenpost.git
   ```

2. Navigate to the new `katzenpost` subdirectory and ensure that the code is up to date.

   ```
   ~$ cd katzenpost
   ```

```
~/katzenpost$ git checkout main
~/katzenpost$ git pull
```

3. (Optional) Create a development branch and check it out.

```
~/katzenpost$ git checkout -b devel
```

4. (Optional) If you are using Podman, complete the following steps:

    1. Point the DOCKER_HOST environment variable at the Podman process.

    ```
    $ export DOCKER_HOST=unix:///var/run/user/$(id -u)/podman/podman.sock
    ```

    2. Set up and start the Podman server (as superuser).

    ```
    $ podman system service -t 0 $DOCKER_HOST &
    $ systemctl --user enable --now podman.socket
    ```

Operating the test mixnet

Navigate to `katzenpost/docker`. The `Makefile` contains target operations to create, manage, and test the self-contained Katzenpost container network. To invoke a target, run a command with the using the following pattern:

```
 ~/katzenpost/docker$ make target
```

Running **make** with no target specified returns a list of available targets.

Table 1. Table 1: Makefile targets

[none]	Display this list of targets.
start	Run the test network in the background.
stop	Stop the test network.
wait	Wait for the test network to have consensus.
watch	Display live log entries until **Ctrl-C**.
status	Show test network consensus status.
show-latest-vote	Show latest consensus vote.
run-ping	Send a ping over the test network.
clean-bin	Stop all components and delete binaries.
clean-local	Stop all components, delete binaries, and delete data.
clean-local-dryrun	Show what clean-local would delete.
clean	Same as **clean-local**, but also deletes `go_deps` image.

Starting and monitoring the mixnet

The first time that you run **make start**, the Docker image is downloaded, built, installed, and started. This takes several minutes. When the build is complete, the command exits while the network remains running in the background.

```
~/katzenpost/docker$ make start
```

Subsequent runs of **make start** either start or restart the network without building the components from scratch. The exception to this is when you delete any of the Katzenpost binaries (dirauth.alpine, server.alpine, etc.). In that case, **make start** rebuilds just the parts of the network dependent on the deleted binary. For more information about the files created during the Docker build, see the section called “Network topology and components”.

Note

When running **make start** , be aware of the following considerations:

- If you intend to use Docker, you need to run **make** as superuser. If you are using **sudo** to elevate your privileges, you need to edit `katzenpost/docker/Makefile` to prepend **sudo** to each command contained in it.
- If you have Podman installed on your system and you nonetheless want to run Docker, you can override the default behavior by adding the argument **docker=docker** to the command as in the following:

  ```
  ~/katzenpost/docker$ make run docker=docker
  ```

After the **make start** command exits, the mixnet runs in the background, and you can run **make watch** to display a live log of the network activity.

```
~/katzenpost/docker$ make watch
    ...
    <output>
    ...
```

When installation is complete, the mix servers vote and reach a consensus. You can use the **wait** target to wait for the mixnet to get consensus and be ready to use. This can also take several minutes:

```
~/katzenpost/docker$ make wait
    ...
    <output>
    ...
```

You can confirm that installation and configuration are complete by issuing the **status** command from the same or another terminal. When the network is ready for use, **status** begins returning consensus information similar to the following:

```
~/katzenpost/docker$ make status
    ...
    00:15:15.003 NOTI state: Consensus made for epoch 1851128 with 3/3 signatur
    ...
```

Testing the mixnet

At this point, you should have a locally running mix network. You can test whether it is working correctly by using **run-ping**, which launches a packet into the network and watches for a successful reply. Run the following command:

```
~/katzenpost/docker$ make run-ping
```

If the network is functioning properly, the resulting output contains lines similar to the following:

```
19:29:53.541 INFO gateway1_client: sending loop decoy
    !19:29:54.108 INFO gateway1_client: sending loop decoy
    19:29:54.632 INFO gateway1_client: sending loop decoy
    19:29:55.160 INFO gateway1_client: sending loop decoy
    !19:29:56.071 INFO gateway1_client: sending loop decoy
    !19:29:59.173 INFO gateway1_client: sending loop decoy
    !Success rate is 100.000000 percent 10/10)
```

If **run-ping** fails to receive a reply, it eventually times out with an error message. If this happens, try the command again.

Note

If you attempt use **run-ping** too quickly after starting the mixnet, and consensus has not been reached, the utility may crash with an error message or hang indefinitely. If this happens, issue (if necessary) a **Ctrl-C** key sequence to abort, check the consensus status with the **status** command, and then retry **run-ping**.

Shutting down the mixnet

The mix network continues to run in the terminal where you started it until you issue a **Ctrl-C** key sequence, or until you issue the following command in another terminal:

```
~/katzenpost/docker$ make stop
```

When you stop the network, the binaries and data are left in place. This allows for a quick restart.

Uninstalling and cleaning up

Several command targets can be used to uninstall the Docker image and restore your system to a clean state. The following examples demonstrate the commands and their output.

- **clean-bin**

  To stop the network and delete the compiled binaries, run the following command:

  ```
  ~/katzenpost/docker$ make clean-bin
  ```

  ```
      [ -e voting_mixnet ] && cd voting_mixnet && DOCKER_HOST=unix:///run/user/1
      Stopping voting_mixnet_auth3_1        ... done
      Stopping voting_mixnet_servicenode1_1 ... done
      Stopping voting_mixnet_metrics_1      ... done
      Stopping voting_mixnet_mix3_1         ... done
      Stopping voting_mixnet_auth2_1        ... done
      Stopping voting_mixnet_mix2_1         ... done
      Stopping voting_mixnet_gateway1_1     ... done
      Stopping voting_mixnet_auth1_1        ... done
      Stopping voting_mixnet_mix1_1         ... done
      Removing voting_mixnet_auth3_1        ... done
      Removing voting_mixnet_servicenode1_1 ... done
      Removing voting_mixnet_metrics_1      ... done
      Removing voting_mixnet_mix3_1         ... done
      Removing voting_mixnet_auth2_1        ... done
      Removing voting_mixnet_mix2_1         ... done
      Removing voting_mixnet_gateway1_1     ... done
      Removing voting_mixnet_auth1_1        ... done
      Removing voting_mixnet_mix1_1         ... done
      removed 'running.stamp'
      rm -vf ./voting_mixnet/*.alpine
      removed './voting_mixnet/echo_server.alpine'
      removed './voting_mixnet/fetch.alpine'
      removed './voting_mixnet/memspool.alpine'
      removed './voting_mixnet/panda_server.alpine'
      removed './voting_mixnet/pigeonhole.alpine'
      removed './voting_mixnet/ping.alpine'
      removed './voting_mixnet/reunion_katzenpost_server.alpine'
  ```

```
    removed './voting_mixnet/server.alpine'
    removed './voting_mixnet/voting.alpine'
```

This command leaves in place the cryptographic keys, the state data, and the logs.

- **clean-local-dryrun**

  To diplay a preview of what **clean-local** would remove, without actually deleting anything, run the following command:

  ```
  ~/katzenpost/docker$ make clean-local-dryrun
  ```

- **clean-local**

  To delete both compiled binaries and data, run the following command:

  ```
  ~/katzenpost/docker$ make clean-local
  ```

  ```
      [ -e voting_mixnet ] && cd voting_mixnet && DOCKER_HOST=unix:///run/user/1
      Removing voting_mixnet_mix2_1         ... done
      Removing voting_mixnet_auth1_1        ... done
      Removing voting_mixnet_auth2_1        ... done
      Removing voting_mixnet_gateway1_1     ... done
      Removing voting_mixnet_mix1_1         ... done
      Removing voting_mixnet_auth3_1        ... done
      Removing voting_mixnet_mix3_1         ... done
      Removing voting_mixnet_servicenode1_1 ... done
      Removing voting_mixnet_metrics_1      ... done
      removed 'running.stamp'
      rm -vf ./voting_mixnet/*.alpine
      removed './voting_mixnet/echo_server.alpine'
      removed './voting_mixnet/fetch.alpine'
      removed './voting_mixnet/memspool.alpine'
      removed './voting_mixnet/panda_server.alpine'
      removed './voting_mixnet/pigeonhole.alpine'
      removed './voting_mixnet/reunion_katzenpost_server.alpine'
      removed './voting_mixnet/server.alpine'
      removed './voting_mixnet/voting.alpine'
      git clean -f -x voting_mixnet
      Removing voting_mixnet/
      git status .
      On branch main
      Your branch is up to date with 'origin/main'.
  ```

- **clean**

  To stop the the network and delete the binaries, the data, and the go_deps image, run the following command as superuser:

  ```
  ~/katzenpost/docker$ sudo make clean
  ```

Network topology and components

The Docker image deploys a working mixnet with all components and component groups needed to perform essential mixnet functions:

- message mixing (including packet reordering, timing randomization, injection of decoy traffic, obfuscation of senders and receivers, and so on)
- service provisioning

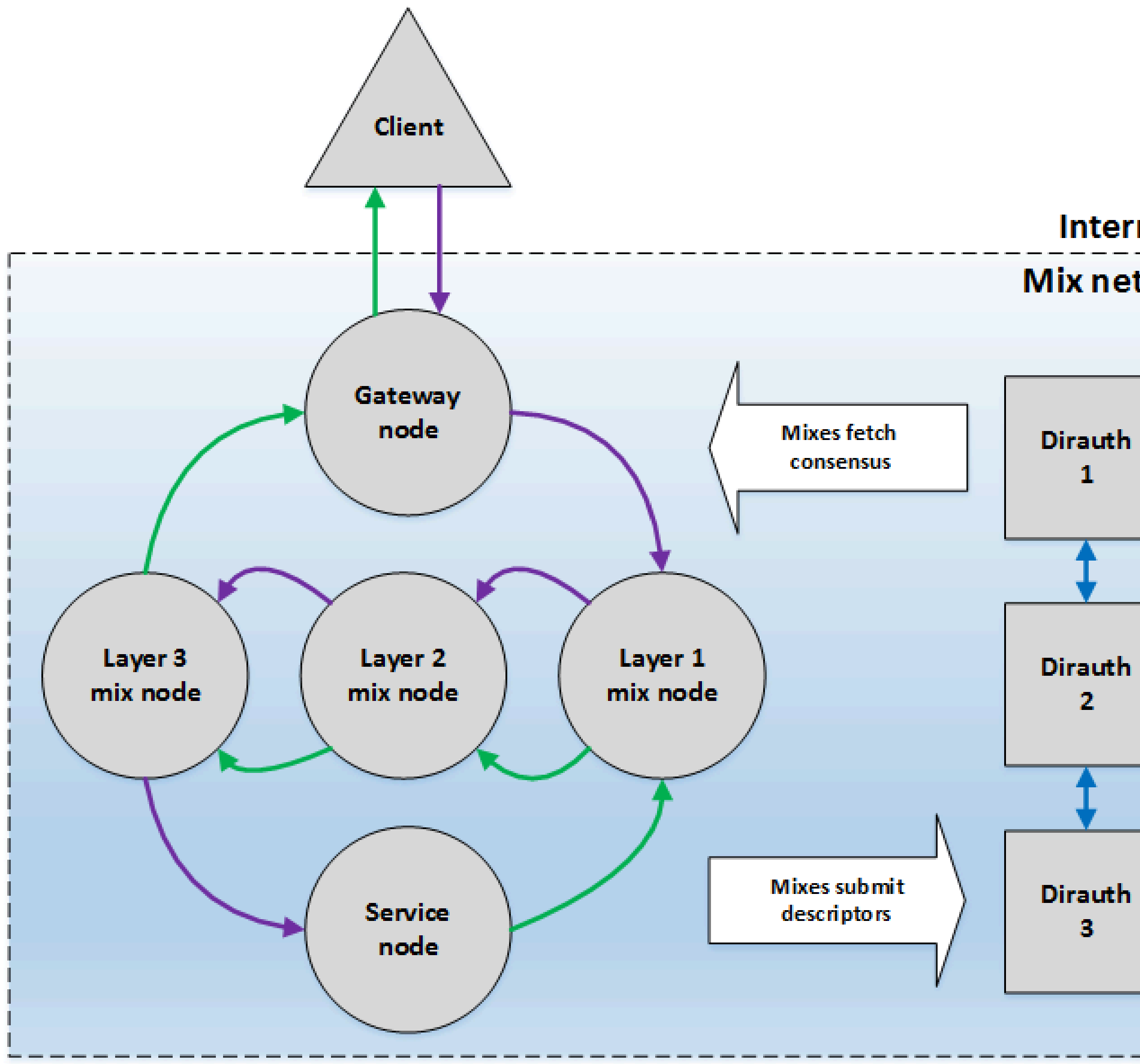
- internal authentication and integrity monitoring
- interfacing with external clients

Warning

While suited for client development and testing, the test mixnet omits performance and security redundancies. *Do not use it in production.*

The following diagram illustrates the components and their network interactions. The gray blocks represent nodes, and the arrows represent information transfer.

Figure 1. Test network topology



On the left, the **Client** transmits a message (shown by purple arrows) through the **Gateway node**, across three **mix node** layers, to the **Service node**. The **Service node** processes the request and responds with a reply (shown by the green arrows) that traverses the **mix node** layers before exiting the mixnet via the **Gateway node** and arriving at the **Client**.

On the right, directory authorities **Dirauth 1**, **Dirauth 2**, and **Dirauth 3** provide PKI services. The directory authorities receive **mix descriptors** from the other nodes, collate these into a **consensus**

document containing validated network status and authentication materials , and make that available to the other nodes.

The elements in the topology diagram map to the mixnet's component nodes as shown in the following table. Note that all nodes share the same IP address (127.0.0.1, i.e., localhost), but are accessed through different ports. Each node type links to additional information in Components and configuration of the Katzenpost mixnet [https://katzenpost.network/docs/admin_guide/components.html].

Table 2. Table 2: Test mixnet hosts

Node type	Docker ID	Diagram label	IP address	TCP port
Directory authority [https://katzenpost.network/docs/admin_guide/intro-dirauth.html]	auth1	Dirauth1	127.0.0.1 (localhost)	30001
	auth2	Dirauth 2		30002
	auth3	Dirauth 3		30003
Gateway node [https://katzenpost.network/docs/admin_guide/intro-gateway.html]	gateway1	Gateway node		30004
Service node [https://katzenpost.network/docs/admin_guide/intro-service.html]	servicenode1	Service node		30006
Mix node [https://katzenpost.network/docs/admin_guide/intro-mix.html]	mix1	Layer 1 mix node		30008
	mix2	Layer 2 mix node		30010
	mix3	Layer 3 mix node		30012

The Docker file tree

The following tree [https://manpages.debian.org/bookworm/tree/tree.1.en.html] output shows the location, relative to the `katzenpost` repository root, of the files created by the Docker build. During testing and use, you would normally touch only the TOML configuration file associated with each node, as highlighted in the listing. For help in understanding these files and a complete list of configuration options, follow the links in Table 2: Test mixnet hosts.

```
katzenpost/docker/voting_mixnet/
|---auth1
|   |---authority.toml
|   |---identity.private.pem
|   |---identity.public.pem
|   |---katzenpost.log
|   |---link.private.pem
|   |---link.public.pem
|   |---persistence.db
|---auth2
|   |---authority.toml
|   |---identity.private.pem
|   |---identity.public.pem
|   |---katzenpost.log
|   |---link.private.pem
|   |---link.public.pem
|   |---persistence.db
|---auth3
|   |---authority.toml
|   |---identity.private.pem
|   |---identity.public.pem
|   |---katzenpost.log
|   |---link.private.pem
|   |---link.public.pem
|   |---persistence.db
|---client
|   |---client.toml
|---client2
|   |---client.toml
|---dirauth.alpine
|---docker-compose.yml
|---echo_server.alpine
|---fetch.alpine
|---gateway1
|   |---identity.private.pem
|   |---identity.public.pem
|   |---katzenpost.log
|   |---katzenpost.toml
|   |---link.private.pem
|   |---link.public.pem
|   |---management_sock
|   |---spool.db
|   |---users.db
|---memspool.alpine
|---mix1
|   |---identity.private.pem
|   |---identity.public.pem
|   |---katzenpost.log
|   |---katzenpost.toml
|   |---link.private.pem
|   |---link.public.pem
|---mix2
|   |---identity.private.pem
|   |---identity.public.pem
|   |---katzenpost.log
|   |---katzenpost.toml
|   |---link.private.pem
|   |---link.public.pem
|---mix3
|   |---identity.private.pem
|   |---identity.public.pem
|   |---katzenpost.log
|   |---katzenpost.toml
|   |---link.private.pem
|   |---link.public.pem
|---panda_server.alpine
|---pigeonhole.alpine
|---ping.alpine
|---prometheus.yml
|---proxy_client.alpine
|---proxy_server.alpine
|---running.stamp
```

```
|---server.alpine
|---servicenode1
|   |---identity.private.pem
|   |---identity.public.pem
|   |---katzenpost.log
|   |---katzenpost.toml
|   |---link.private.pem
|   |---link.public.pem
|   |---management_sock
|   |---map.storage
|   |---memspool.13.log
|   |---memspool.storage
|   |---panda.25.log
|   |---panda.storage
|   |---pigeonHole.19.log
|   |---proxy.31.log
|---voting_mixnet
```

Examples of complete TOML configuration files are provided in Appendix: Configuration files from the Docker test mixnet [https://katzenpost.network/docs/admin_guide/docker-config.html].

Tuning the Katzenpost mixnet

To do

Using Katzenpost from behind a NAT device

To do

Katzenpost servers can be used from behind network address translation (NAT) devices. This applies to mix nodes, gateway nodes, service nodes, and directory authority nodes (dirauths). Port forwarding and other network configuration details depend on how you are hosting your servers and the type of router you use.

Some hosting scenarios, such as the use of an AWS EC2 instance, require no manual port forwarding. A Katzenpost node running on an EC2 instance with default network settings listens on its internal IP address yet can receive connections from publicly routed IP addresses. For home and small business routers, default policy is to block inbound connections from public addresses. In this scenario, you need to configure port forwarding to the appropriate internal IP address and port.

Table 1. Lan hosting options

	Mix node	Gateway node	Service node	Dirauth
Transparent port forwarding (EC2, etc.)				
Selective forwarding (home and small business routers)				
Tor				

Using Tor

Dirauth is a cooperative crash fault tolerant voting system composed of multiple dirauth nodes. However, each of these dirauth nodes must have knowledge of the other dirauth node's network connection information and public cryptographic key material. It's often convenient enough for dirauth operators to use a private git repo to coordinate changes to the dirauth configurations and public key materials. Therefore, dirauth operators wishing to operate a node behind a NAT device can configure their node to listen on a RFC1918 address such as 192.168.1.22 and yet tell the other dirauth node operators that their publicly routable address is some other IP address or perhaps a Tor Onion address.

Dirauths and gateways can be hosted as Tor onion services. (Mix and service nodes cannot be hosted in this manner because they communicate only inside the mix network.)

Any mix node type can operate behind a NAT device by using the *`BindAddresses`* parameter. When *`BindAddresses`* is set, a node listens at one address while advertising a different address as configured by the *`Addresses`* parameter, which specifies the address that the node shares with other nodes through the PKI document.

If you are configuring a Gateway node to listen on a Tor onion service, then you must make sure that the BindAddresses match the listening address and port in your "torrc" file:

```
HiddenServiceDir /var/lib/tor/my_website/
HiddenServicePort 1234 127.0.0.1:1234
```

We refer you to the Tor onion setup docs for more information: https://community.torproject.org/onion-services/setup/

The corresponding Gateway config Server section might look like this:

```
[Server]
Identifier = "gnunet"
PKISignatureScheme = "Ed25519 Sphincs+"
WireKEM = "KYBER768-X25519"
Addresses = [ "onion://your-onion-address.onion:1234"]
BindAddresses = [ "tcp://127.0.0.1:1234",]
DataDir = "/var/lib/pq-katzenpost-mixserver"
IsGatewayNode = true
IsServiceNode = false
...
```

DIAGRAM: ROUTER, IPs, etc.

mix node, device NAT device/router; Nat dev has an Internet route. Internall, mix node binds to an internal addrexs (show configuration). This is the same for Tor, but with Onion addess rahter that a static IP (see Tor and link)

NATPMP STUN TURN– these are Nat PENETRATION PROTOCLDS... NOT SUPPORTED

Two diagrams...Onion and IPv4/IPv6

This confguration only allows for the Gateway nodes (which are also mix nodes).

We need a to show an excerpt from a torrc to show where the onion listening port is confuigred.

Using *BindAddresses* with *Addresses*

In a conventional NAT configuration, a router sits between two IP networks, one with public, globally unique addresses and the other (the LAN) with reusable local addresses. A host with a public address can connect to a host with a local address because the router is configured with a mapping between the two networks. The router rewrites the destination and source headers of the packets that traverse it in both directions, creating the illusion of an unmediated end-to-end connection. The public host sends packs to a public address

Appendix: Configuration files from the Docker test mixnet

As an aid to adminstrators implementing a Katzenpost mixnet, this appendix provides lightly edited examples of configuration files for each Katzenpost node type. These files are drawn from a built instance of the Docker test mixnet. These code listings are meant to be used as a reference alongside the detailed configuration documentation in Components and configuration of the Katzenpost mixnet [https://katzenpost.network/docs/admin_guide/components.html]. You cannot use these listings as a drop-in solution in your own mixnets for reasons explained in the Network topology and components [https://katzenpost.network/docs/admin_guide/topology.html] section of the Docker test mixnet documentation.

Directory authority

Source: `../katzenpost/docker/voting_mixnet/auth1/authority.toml`

```
[Server]
  Identifier = "auth1"
  WireKEMScheme = "xwing"
  PKISignatureScheme = "Ed448-Dilithium3"
  Addresses = ["tcp://127.0.0.1:30001"]
  DataDir = "/voting_mixnet/auth1"

[[Authorities]]
  Identifier = "auth1"
  IdentityPublicKey = "-----BEGIN ED448-DILITHIUM3 PUBLIC KEY-----\nfvcvAfUpeu7
  PKISignatureScheme = "Ed448-Dilithium3"
  LinkPublicKey = "-----BEGIN XWING PUBLIC KEY-----\nsxxS04mftoEmwjxE/w [...] e
  WireKEMScheme = "xwing"
  Addresses = ["tcp://127.0.0.1:30001"]

[[Authorities]]
  Identifier = "auth2"
  IdentityPublicKey = "-----BEGIN ED448-DILITHIUM3 PUBLIC KEY-----\n5nsy6uFQ178
  PKISignatureScheme = "Ed448-Dilithium3"
  LinkPublicKey = "-----BEGIN XWING PUBLIC KEY-----\nkQzCJvaS6jg06szLea [...] P
  WireKEMScheme = "xwing"
  Addresses = ["tcp://127.0.0.1:30002"]

[[Authorities]]
  Identifier = "auth3"
  IdentityPublicKey = "-----BEGIN ED448-DILITHIUM3 PUBLIC KEY-----\nJzkFpS035de
  PKISignatureScheme = "Ed448-Dilithium3"
  LinkPublicKey = "-----BEGIN XWING PUBLIC KEY-----\n+pIUsgEGwHa8k4GZcb [...] 1
  WireKEMScheme = "xwing"
  Addresses = ["tcp://127.0.0.1:30003"]

[Logging]
  Disable = false
  File = "katzenpost.log"
  Level = "INFO"

[Parameters]
  SendRatePerMinute = 0
  Mu = 0.005
  MuMaxDelay = 1000
  LambdaP = 0.001
  LambdaPMaxDelay = 1000
  LambdaL = 0.0005
  LambdaLMaxDelay = 1000
  LambdaD = 0.0005
  LambdaDMaxDelay = 3000
  LambdaM = 0.0005
  LambdaG = 0.0
  LambdaMMaxDelay = 100
  LambdaGMaxDelay = 100

[Debug]
  Layers = 3
  MinNodesPerLayer = 1
  GenerateOnly = false

[[Mixes]]
  Identifier = "mix1"
  IdentityPublicKeyPem = "../mix1/identity.public.pem"

[[Mixes]]
  Identifier = "mix2"
  IdentityPublicKeyPem = "../mix2/identity.public.pem"

[[Mixes]]
  Identifier = "mix3"
  IdentityPublicKeyPem = "../mix3/identity.public.pem"

[[GatewayNodes]]
  Identifier = "gateway1"
  IdentityPublicKeyPem = "../gateway1/identity.public.pem"

[[ServiceNodes]]
  Identifier = "servicenode1"
  IdentityPublicKeyPem = "../servicenode1/identity.public.pem"

[Topology]

  [[Topology.Layers]]

    [[Topology.Layers.Nodes]]
      Identifier = "mix1"
      IdentityPublicKeyPem = "../mix1/identity.public.pem"

  [[Topology.Layers]]

    [[Topology.Layers.Nodes]]
      Identifier = "mix2"
      IdentityPublicKeyPem = "../mix2/identity.public.pem"

  [[Topology.Layers]]

    [[Topology.Layers.Nodes]]
      Identifier = "mix3"
      IdentityPublicKeyPem = "../mix3/identity.public.pem"

[SphinxGeometry]
```

```
  PacketLength = 3082
  NrHops = 5
  HeaderLength = 476
  RoutingInfoLength = 410
  PerHopRoutingInfoLength = 82
  SURBLength = 572
  SphinxPlaintextHeaderLength = 2
  PayloadTagLength = 32
  ForwardPayloadLength = 2574
  UserForwardPayloadLength = 2000
  NextNodeHopLength = 65
  SPRPKeyMaterialLength = 64
  NIKEName = "x25519"
  KEMName = ""
```

Mix node

Source: `../katzenpost/docker/voting_mixnet/mix1/katzenpost.toml`

```
[Server]
  Identifier = "mix1"
  WireKEM = "xwing"
  PKISignatureScheme = "Ed448-Dilithium3"
  Addresses = ["tcp://127.0.0.1:30010", "quic://[::1]:30011"]
  MetricsAddress = "127.0.0.1:30012"
  DataDir = "/voting_mixnet/mix1"
  IsGatewayNode = false
  IsServiceNode = false

[Logging]
  Disable = false
  File = "katzenpost.log"
  Level = "INFO"

[PKI]
  [PKI.Voting]

    [[PKI.Voting.Authorities]]
      Identifier = "auth1"
      IdentityPublicKey = "-----BEGIN ED448-DILITHIUM3 PUBLIC KEY-----\nfvcvAfU
      PKISignatureScheme = "Ed448-Dilithium3"
      LinkPublicKey = "-----BEGIN XWING PUBLIC KEY-----\nsxxS04mftoEmwjxE/w [..
      WireKEMScheme = "xwing"
      Addresses = ["tcp://127.0.0.1:30001"]

    [[PKI.Voting.Authorities]]
      Identifier = "auth2"
      IdentityPublicKey = "-----BEGIN ED448-DILITHIUM3 PUBLIC KEY-----\n5nsy6uF
      PKISignatureScheme = "Ed448-Dilithium3"
      LinkPublicKey = "-----BEGIN XWING PUBLIC KEY-----\nkQzCJvaS6jg06szLea [..
      WireKEMScheme = "xwing"
      Addresses = ["tcp://127.0.0.1:30002"]

    [[PKI.Voting.Authorities]]
      Identifier = "auth3"
      IdentityPublicKey = "-----BEGIN ED448-DILITHIUM3 PUBLIC KEY-----\nJzkFpS0
      PKISignatureScheme = "Ed448-Dilithium3"
      LinkPublicKey = "-----BEGIN XWING PUBLIC KEY-----\n+pIUsgEGwHa8k4GZcb [..
      WireKEMScheme = "xwing"
      Addresses = ["tcp://127.0.0.1:30003"]

[Management]
  Enable = false
  Path = "/voting_mixnet/mix1/management_sock"

[SphinxGeometry]
  PacketLength = 3082
  NrHops = 5
  HeaderLength = 476
  RoutingInfoLength = 410
  PerHopRoutingInfoLength = 82
  SURBLength = 572
  SphinxPlaintextHeaderLength = 2
  PayloadTagLength = 32
  ForwardPayloadLength = 2574
  UserForwardPayloadLength = 2000
  NextNodeHopLength = 65
  SPRPKeyMaterialLength = 64
  NIKEName = "x25519"
  KEMName = ""

[Debug]
  NumSphinxWorkers = 16
  NumServiceWorkers = 3
  NumGatewayWorkers = 3
  NumKaetzchenWorkers = 3
  SchedulerExternalMemoryQueue = false
  SchedulerQueueSize = 0
  SchedulerMaxBurst = 16
  UnwrapDelay = 250
  GatewayDelay = 500
  ServiceDelay = 500
  KaetzchenDelay = 750
  SchedulerSlack = 150
  SendSlack = 50
  DecoySlack = 15000
  ConnectTimeout = 60000
  HandshakeTimeout = 30000
  ReauthInterval = 30000
  SendDecoyTraffic = false
  DisableRateLimit = false
  GenerateOnly = false
```

Gateway node

Source: `../katzenpost/docker/voting_mixnet/gateway1/katzenpost.toml`

```
[Server]
  Identifier = "gateway1"
  WireKEM = "xwing"
  PKISignatureScheme = "Ed448-Dilithium3"
  Addresses = ["tcp://127.0.0.1:30004", "quic://[::1]:30005", "onion://thisisju
  BindAddresses = ["tcp://127.0.0.1:30004", "quic://[::1]:30005"]
  MetricsAddress = "127.0.0.1:30006"
  DataDir = "/voting_mixnet/gateway1"
  IsGatewayNode = true
```

```
    IsServiceNode = false

  [Logging]
    Disable = false
    File = "katzenpost.log"
    Level = "INFO"

  [Gateway]
    [Gateway.UserDB]
      Backend = "bolt"
      [Gateway.UserDB.Bolt]
        UserDB = "/voting_mixnet/gateway1/users.db"
    [Gateway.SpoolDB]
      Backend = "bolt"
      [Gateway.SpoolDB.Bolt]
        SpoolDB = "/voting_mixnet/gateway1/spool.db"

  [PKI]
    [PKI.Voting]

      [[PKI.Voting.Authorities]]
        Identifier = "auth1"
        IdentityPublicKey = "-----BEGIN ED448-DILITHIUM3 PUBLIC KEY-----\nfvcvAfU
        PKISignatureScheme = "Ed448-Dilithium3"
        LinkPublicKey = "-----BEGIN XWING PUBLIC KEY-----\nsxxS04mftoEmwjxE/w [..
        WireKEMScheme = "xwing"
        Addresses = ["tcp://127.0.0.1:30001"]

      [[PKI.Voting.Authorities]]
        Identifier = "auth2"
        IdentityPublicKey = "-----BEGIN ED448-DILITHIUM3 PUBLIC KEY-----\n5nsy6uF
        PKISignatureScheme = "Ed448-Dilithium3"
        LinkPublicKey = "-----BEGIN XWING PUBLIC KEY-----\nkQzCJvaS6jg06szLea [..
        WireKEMScheme = "xwing"
        Addresses = ["tcp://127.0.0.1:30002"]

      [[PKI.Voting.Authorities]]
        Identifier = "auth3"
        IdentityPublicKey = "-----BEGIN ED448-DILITHIUM3 PUBLIC KEY-----\nJzkFpS0
        PKISignatureScheme = "Ed448-Dilithium3"
        LinkPublicKey = "-----BEGIN XWING PUBLIC KEY-----\n+pIUsgEGwHa8k4GZcb [..
        WireKEMScheme = "xwing"
        Addresses = ["tcp://127.0.0.1:30003"]

  [Management]
    Enable = true
    Path = "/voting_mixnet/gateway1/management_sock"

  [SphinxGeometry]
    PacketLength = 3082
    NrHops = 5
    HeaderLength = 476
    RoutingInfoLength = 410
    PerHopRoutingInfoLength = 82
    SURBLength = 572
    SphinxPlaintextHeaderLength = 2
    PayloadTagLength = 32
    ForwardPayloadLength = 2574
    UserForwardPayloadLength = 2000
    NextNodeHopLength = 65
    SPRPKeyMaterialLength = 64
    NIKEName = "x25519"
    KEMName = ""

  [Debug]
    NumSphinxWorkers = 16
    NumServiceWorkers = 3
    NumGatewayWorkers = 3
    NumKaetzchenWorkers = 3
    SchedulerExternalMemoryQueue = false
    SchedulerQueueSize = 0
    SchedulerMaxBurst = 16
    UnwrapDelay = 250
    GatewayDelay = 500
    ServiceDelay = 500
    KaetzchenDelay = 750
    SchedulerSlack = 150
    SendSlack = 50
    DecoySlack = 15000
    ConnectTimeout = 60000
    HandshakeTimeout = 30000
    ReauthInterval = 30000
    SendDecoyTraffic = false
    DisableRateLimit = false
    GenerateOnly = false
```

Service node

Source: `../katzenpost/docker/voting_mixnet/servicenode1/katzenpost.toml`

```
[Server]
  Identifier = "servicenode1"
  WireKEM = "xwing"
  PKISignatureScheme = "Ed448-Dilithium3"
  Addresses = ["tcp://127.0.0.1:30007", "quic://[::1]:30008"]
  MetricsAddress = "127.0.0.1:30009"
  DataDir = "/voting_mixnet/servicenode1"
  IsGatewayNode = false
  IsServiceNode = true

[Logging]
  Disable = false
  File = "katzenpost.log"
  Level = "INFO"

[ServiceNode]

  [[ServiceNode.Kaetzchen]]
    Capability = "echo"
    Endpoint = "+echo"
    Disable = false

  [[ServiceNode.Kaetzchen]]
    Capability = "testdest"
    Endpoint = "+testdest"
```

```
      Disable = false

    [[ServiceNode.CBORPluginKaetzchen]]
      Capability = "spool"
      Endpoint = "+spool"
      Command = "/voting_mixnet/memspool.alpine"
      MaxConcurrency = 1
      Disable = false
      [ServiceNode.CBORPluginKaetzchen.Config]
        data_store = "/voting_mixnet/servicenode1/memspool.storage"
        log_dir = "/voting_mixnet/servicenode1"

    [[ServiceNode.CBORPluginKaetzchen]]
      Capability = "pigeonhole"
      Endpoint = "+pigeonhole"
      Command = "/voting_mixnet/pigeonhole.alpine"
      MaxConcurrency = 1
      Disable = false
      [ServiceNode.CBORPluginKaetzchen.Config]
        db = "/voting_mixnet/servicenode1/map.storage"
        log_dir = "/voting_mixnet/servicenode1"

    [[ServiceNode.CBORPluginKaetzchen]]
      Capability = "panda"
      Endpoint = "+panda"
      Command = "/voting_mixnet/panda_server.alpine"
      MaxConcurrency = 1
      Disable = false
      [ServiceNode.CBORPluginKaetzchen.Config]
        fileStore = "/voting_mixnet/servicenode1/panda.storage"
        log_dir = "/voting_mixnet/servicenode1"
        log_level = "INFO"

    [[ServiceNode.CBORPluginKaetzchen]]
      Capability = "http"
      Endpoint = "+http"
      Command = "/voting_mixnet/proxy_server.alpine"
      MaxConcurrency = 1
      Disable = false
      [ServiceNode.CBORPluginKaetzchen.Config]
        host = "localhost:4242"
        log_dir = "/voting_mixnet/servicenode1"
        log_level = "DEBUG"

  [PKI]
    [PKI.Voting]

      [[PKI.Voting.Authorities]]
        Identifier = "auth1"
        IdentityPublicKey = "-----BEGIN ED448-DILITHIUM3 PUBLIC KEY-----\nfvcvAfU
        PKISignatureScheme = "Ed448-Dilithium3"
        LinkPublicKey = "-----BEGIN XWING PUBLIC KEY-----\nsxxS04mftoEmwjxE/w [..
        WireKEMScheme = "xwing"
        Addresses = ["tcp://127.0.0.1:30001"]

      [[PKI.Voting.Authorities]]
        Identifier = "auth2"
        IdentityPublicKey = "-----BEGIN ED448-DILITHIUM3 PUBLIC KEY-----\n5nsy6uF
        PKISignatureScheme = "Ed448-Dilithium3"
        LinkPublicKey = "-----BEGIN XWING PUBLIC KEY-----\nkQzCJvaS6jg06szLea [..
        WireKEMScheme = "xwing"
        Addresses = ["tcp://127.0.0.1:30002"]

      [[PKI.Voting.Authorities]]
        Identifier = "auth3"
        IdentityPublicKey = "-----BEGIN ED448-DILITHIUM3 PUBLIC KEY-----\nJzkFpS0
        PKISignatureScheme = "Ed448-Dilithium3"
        LinkPublicKey = "-----BEGIN XWING PUBLIC KEY-----\n+pIUsgEGwHa8k4GZcb [..
        WireKEMScheme = "xwing"
        Addresses = ["tcp://127.0.0.1:30003"]

  [Management]
    Enable = true
    Path = "/voting_mixnet/servicenode1/management_sock"

  [SphinxGeometry]
    PacketLength = 3082
    NrHops = 5
    HeaderLength = 476
    RoutingInfoLength = 410
    PerHopRoutingInfoLength = 82
    SURBLength = 572
    SphinxPlaintextHeaderLength = 2
    PayloadTagLength = 32
    ForwardPayloadLength = 2574
    UserForwardPayloadLength = 2000
    NextNodeHopLength = 65
    SPRPKeyMaterialLength = 64
    NIKEName = "x25519"
    KEMName = ""

  [Debug]
    NumSphinxWorkers = 16
    NumServiceWorkers = 3
    NumGatewayWorkers = 3
    NumKaetzchenWorkers = 4
    SchedulerExternalMemoryQueue = false
    SchedulerQueueSize = 0
    SchedulerMaxBurst = 16
    UnwrapDelay = 250
    GatewayDelay = 500
    ServiceDelay = 500
    KaetzchenDelay = 750
    SchedulerSlack = 150
    SendSlack = 50
    DecoySlack = 15000
    ConnectTimeout = 60000
    HandshakeTimeout = 30000
    ReauthInterval = 30000
    SendDecoyTraffic = false
    DisableRateLimit = false
    GenerateOnly = false
```

Operating the Katzenpost mixnet

To do

CLI usage for directory authorities

To do

CLI usage for servers

To do

Appendix: Using *gensphinx*

To Do