# Using Katzenpost with NAT and Tor

## Table of Contents



Any Katzenpost server node can be configured to support NAT and similar network topologies (such as Tor) that traverse public and private network boundaries. All Katzenpost node types (mix nodes, gateway nodes, service nodes, and dirauths) can be used from behind network address translation (NAT) [https://www.rfc-editor.org/rfc/rfc1631] routers. A variation of NAT topology may be used for hosting in a container environment, such as Docker, or when a node runs in a virtualized environment, such as Qemu or KVM. Additionally, gateway nodes running a Tor onion service [https://tb-manual.torproject.org/onion-services/] can be accessed from the Internet using an onion address.

Some indirect network scenarios that involve NAT, such as Amazon EC2, are effectively transparent and require no special Katzenpost configuration. More typically, the router connecting a LAN with the Internet blocks incoming connections by default, and must be configured to forward traffic from the Internet to a destination LAN host based on port number. (Router configuration for a NAT topology is beyond the scope of this topic.) For these cases, where the host listens on a LAN `address:port` but is accessed publicly from another `address:port`, Katzenpost provides a mechanism to specify both addresses and to share access information with peers and clients.

### Note

Katzenpost does not support NAT penetration protocols such as NATPMP [https://www.rfc-editor.org/rfc/rfc6886], STUN [https://www.rfc-editor.org/rfc/rfc5389], and TURN [https://www.rfc-editor.org/rfc/rfc5766].

# *Addresses* and *BindAddresses*

In a direct network connection, the address defined in the server *Addresses* parameter defines the addresses on which the node listens for incoming connections, and which are advertised to other mixnet components in the PKI document. By supplying the optional *BindAddresses* parameter, you can define a second address group: internal addresses that are *not* advertised in the PKI document. This is useful for NAT and Tor scenarios.

### Note

The *Addresses* and *BindAddresses* parameters are closely analogous to Tor's *Address* and *ORPort* parameters. For more information, see the torrc man page [https://manpages.debian.org/testing/tor/torrc.5.en.html].

The following table shows the details for these two parameters. For more information about mixnet node configuration, see Components and configuration of the Katzenpost mixnet [https://katzenpost.network/docs/admin_guide/components.html].

**Table 1. *Addresses* and *BindAddresses* parameters**

| *Parameter* | Required | Description |
|---|---|---|
| *Addresses* | Yes | Specifies a list of one or more address URLs in a format that contains the transport protocol, IP ad- |

| *Parameter* | Required | Description |
|---|---|---|
| | | dress, and port number that the server will bind to for incoming connections. This value is advertised in the PKI document. 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"]`. Onion addresses, identified by transport protocol onion://, are also supported on some node types if *`BindAddresses`* is also present.<br><br>This parameter is overridden by *`BindAddresses`*; see below. |
| *`BindAddresses`* | No | If **true** (that is, if the parameter is present), this parameter allows you to set internal listener addresses that the server will bind to and accept connections on, but that are not advertised in the PKI document. |

For example, a mix node configuration file with a direct Internet connection, or with transparent NAT hosting such as that provided by Amazon EC2, contains a `Server` section specifying only the *`Addresses`* parameter, as in the following listing:

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

Importantly, the value of *`Addresses`*, which the node advertises to other peers, is its loopback address, 127.0.0.1, and a specified port. This means that there is no publicly routable address for this node, and that traffic arriving at the node is delivered there by a configured LAN. In all of these examples, nodes have necessarily been assigned a LAN addresses, but these non-routable RFC 1918 [https://www.rfc-editor.org/rfc/rfc1918] addresses play no role in Katzenpost server configurations. Rather, for portability, the configurations use the logically equivalent loopback address.

In contrast to the direct connection, the next example shows part of the configuration file for a similar node behind NAT, containing a `Server` section that specifies both the *`Addresses`* and *`BindAddresses`*parameters:

```
[Server]
  Identifier = "mix1"
  WireKEM = "xwing"
  PKISignatureScheme = "Ed448-Dilithium3"
  Addresses = ["tcp://203.0.113.10:30010"]
  BindAddresses = ["tcp://127.0.0.1:30010"]
  MetricsAddress = "127.0.0.1:30012"
  DataDir = "/voting_mixnet/mix1"
  IsGatewayNode = false
  IsServiceNode = false
```

Notice that the local listening address remains unchanged, *but has been relocated to the new* *`BindAddresses`* *parameter*. In addition, the value of the *`Addresses`* parameter has been changed to a public address (the NAT router's WAN address), including a port. It is assumed that the router is configured to forward traffic routed to this `address:port` to the mixnet node over the LAN.

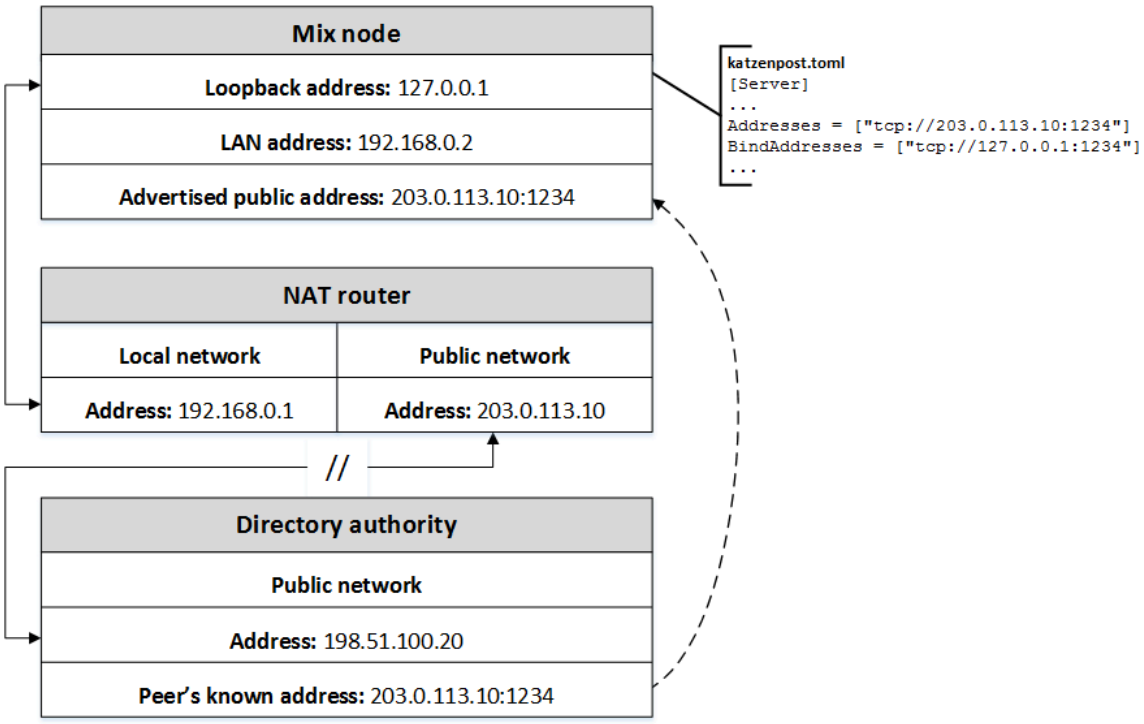
# Application scenarios

The following sections illustrate supported Katzenpost topologies that make use of the *BindAddresses* parameter.

## Peer-to-peer connections involving NAT

In this scenario, a mix node behind NAT listens on local addresses for connections, while advertising a public address and port to its peer, a directory authority, that is assumed to have a publicly routable address.

**Figure 1. Configuring a mix node behind NAT to be available to a dirauth**




Enlarge diagram [https://katzenpost.network/docs/admin_guide/pix/peer-to-peer-nat.png]

**Key observations**

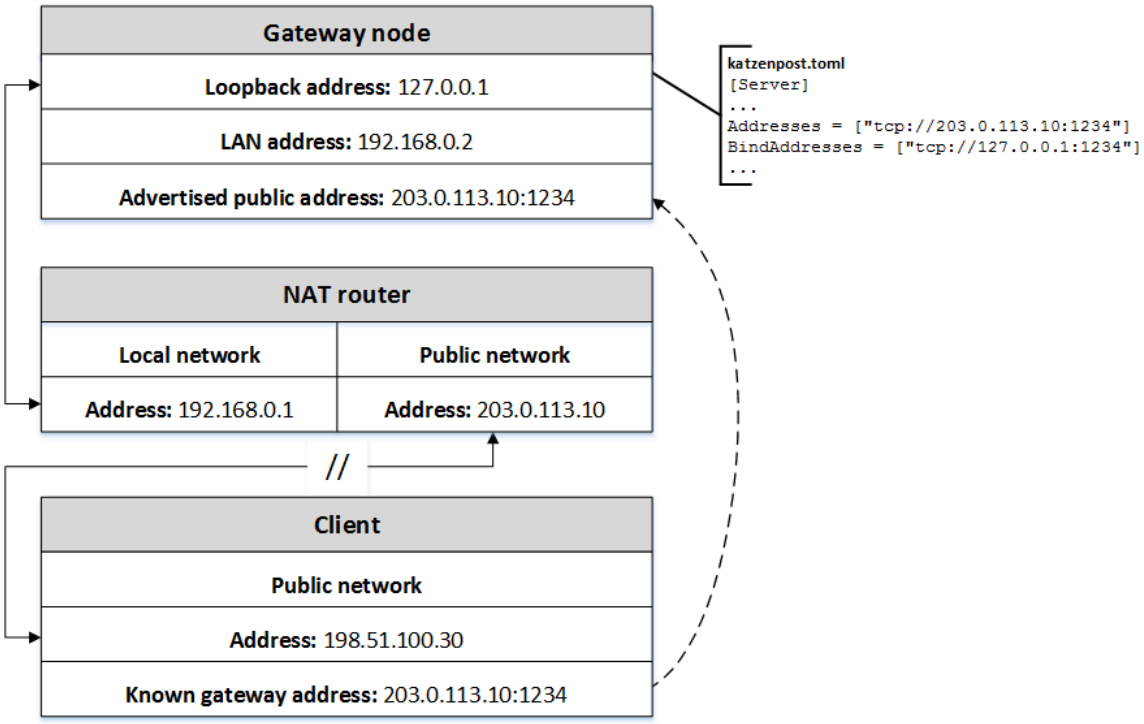
- The configuration file on the NATed mix node is `katzenpost.toml`.
- The relevant section of the configuration file is `[Server]`.
- The *Addresses* parameter specifies the publicly routable address (203.0.113.10) and a port (1234) over which the mix node can be reached from the Internet. This value is periodically advertised in the PKI document to other components of the mix network.
- The *BindAddresses* parameter specifies the loopback address (127.0.0.1) and a port (1234) on which the node listens for incoming Sphinx packets from peers.
- The mix node also has a LAN address, 192.168.0.2, which does not appear in the Katzenpost configuration, but is logically equivalent to the node's loopback address, 127.0.0.1.
- The NAT router has two configured addresses, public address 203.0.113.10 and LAN address 192.168.0.1.

- The NAT device routes traffic for 203.0.113.10:1234 to the LAN `address:port` of the mix node, 192.168.0.2:1234, and therefore to the configured listener bound to 127.0.0.1:1234.

# Client-to-gateway connections involving NAT

In this scenario, a gateway node behind NAT listens on local addresses for connections from the Internet, while advertising a public address and port to a client application that is assumed to have a publicly routable address.

**Figure 2. Configuring a gateway behind NAT to be available to a client**




Enlarge diagram [https://katzenpost.network/docs/admin_guide/pix/client-gateway-nat.png]

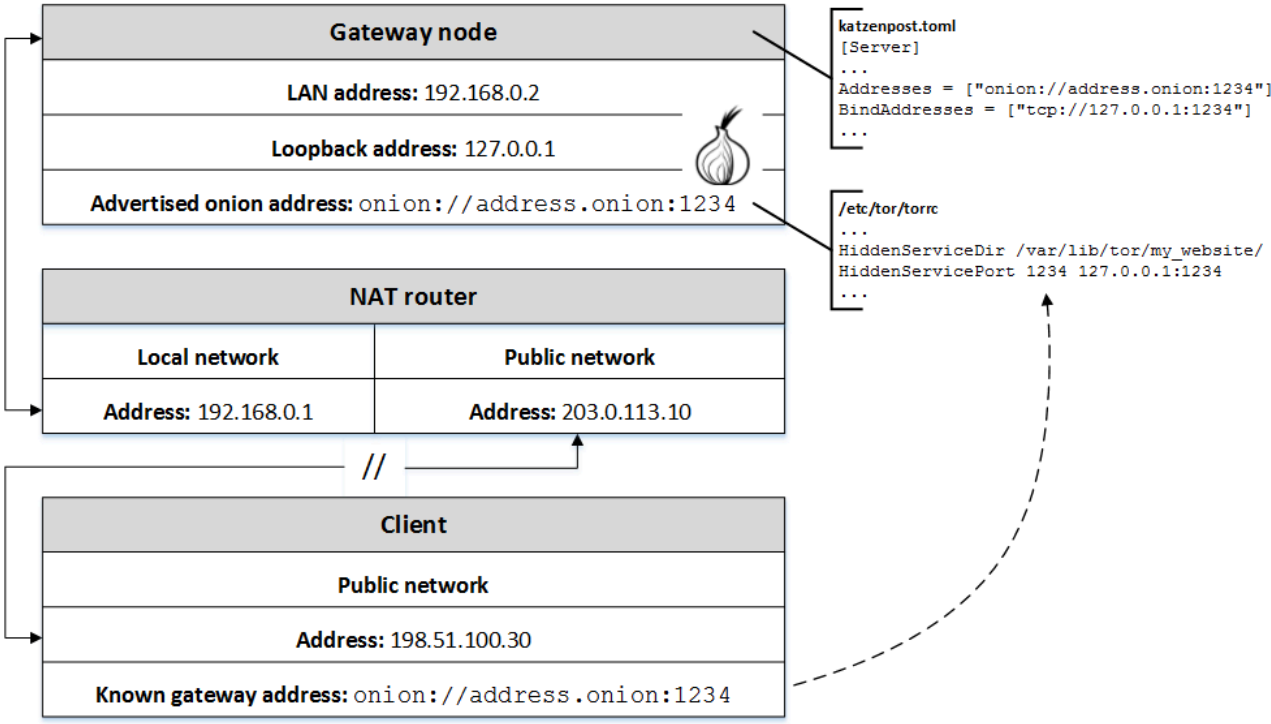
**Key observations**

- The configuration file on the NATed gateway node is `katzenpost.toml`.
- The relevant section of the configuration file is `[Server]`.
- The *`Addresses`* parameter specifies the publicly routable address (203.0.113.10) and a port (1234) over which the gateway node can be reached from the Internet. This value is periodically advertised in the PKI document to other components of the mix network.
- The *`BindAddresses`* parameter specifies the loopback address (127.0.0.1) and a port (1234) on which the node listens for incoming Sphinx packets from clients.
- The gateway node also has a LAN address, 192.168.0.2, which does not appear in the Katzenpost configuration, but is logically equivalent to the node's loopback address, 127.0.0.1.
- The NAT router has two configured addresses, public address 203.0.113.10, and LAN address 192.168.0.1.
- The NAT device routes traffic for 203.0.113.10:1234 to the LAN `address:port` of the gateway node, 192.168.0.2:1234, and therefore to the configured listener bound to 127.0.0.1:1234.

# Client-to-gateway connections over Tor

In this scenario, a gateway node behind NAT listens on local addresses for connections from the Internet, while advertising an onion address and port to a client application that is assumed to have a publicly routable address. Apart from the use of an onion address, this scenario is identical to the previous one.

### Figure 3. Configuring a gateway to be available to a client over Tor




Enlarge diagram [https://katzenpost.network/docs/admin_guide/pix/client-gateway-onion.png]

**Key observations**

- The configuration file on the NATed gateway node is `katzenpost.toml`.
- The relevant section of the configuration file is `[Server]`.
- The gateway node exposes itself to the network as an onion (hidden) service bound to the loopback address (127.0.0.1) and a port (1234). This requires a Tor daemon to be running on the node and for the Tor configuration file `/etc/tor/torrc` to contain the lines shown.
- The *`Addresses`* parameter specifies the onion address (`address.onion`) and a port (1234) over which the gateway node can be reached by way of the Tor network. This value is periodically advertised in the PKI document to other components of the mix network.
- The *`BindAddresses`* parameter specifies the loopback address (127.0.0.1) and a port (1234) on which the node listens for incoming Sphinx packets from clients.
- The gateway node also has a LAN address, 192.168.0.2, which does not appear in the Katzenpost configuration, but is logically equivalent to the node's loopback address, 127.0.0.1.
- The NAT router has two configured addresses, public address 203.0.113.10, and LAN address 192.168.0.1.
- The NAT device routes traffic for 203.0.113.10:1234 to the LAN address and port of the gateway node, 192.168.0.2:1234, and therefore to the configured onion service listener bound to 127.0.0.1:1234.

# Using NAT with directory authorities

Directory authorities have no support for the *BindAddresses* parameter. Unlike mix, gateway, and service nodes, dirauths have no mechanism to announce their addresses automatically. Rather, their public addresses must be added out-of-band to the PKI document on which peers and clients rely. Consequently, conventional NAT works without special Katzenpost configuration. This is demonstrated by a scenario involving three dirauths – dirauth1, dirauth2 and dirauth3 – where dirauth1 is behind NAT, with a `server` section like the following in its configuration:

```
[Server]
  Identifier = "dirauth1"
  PKISignatureScheme = "Ed25519 Sphincs+"
  WireKEMScheme = "KYBER768-X25519"
  Addresses = [ "tcp://192.168.1.123:52323" ]
  DataDir = "/var/lib/pq-katzenpost-authority"
```

The listening address on dirauth1 is LAN IP address `192.168.1.123`. It differs from the other examples in this topic by its use of a LAN address rather than the loopback address (127.0.0.1). This configuration is not seen by the other dirauth nodes; it only determines which network interface dirauth1 is listening on. Peer nodes, such as dirauth2 and dirauth3, have configuration files containing a public IP address, provided out-of-band, for dirauth1 in the `Authorities` section:

```
[[Authorities]]
  Identifier = "dirauth1"
  PKISignatureScheme = "Ed25519 Sphincs+"
  WireKEMScheme = "KYBER768-X25519"
  Addresses = [ "tcp://203.0.113.10:52323" ]
```

The NAT router passes incoming connections to their destination host based on `address:port` information in the usual manner.