The Proxy Network

a simple Java-based anonymizing network

Lorenzo Rossi

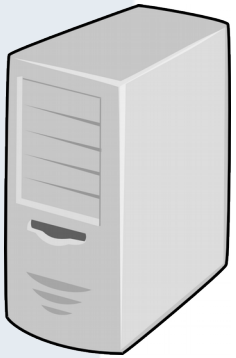
SNCS Project Course

Why?

- Aim of the attacker:
  - Create an association between a message received by a certain server and the IP address of the client that has produced that message.
- Why you need anonimity?
  - (we assume you are not the bad guy looking for safety for shady traffics!)
  - Electronic Vote
  - Anonymous witness of criminal offences (e.g MafiaLeaks)
  - Or simply for privacy

PN: Actors

The Client: it want to send messages anonymously to the receiver server. Knows the K_R^+ (public key of the receiver server)



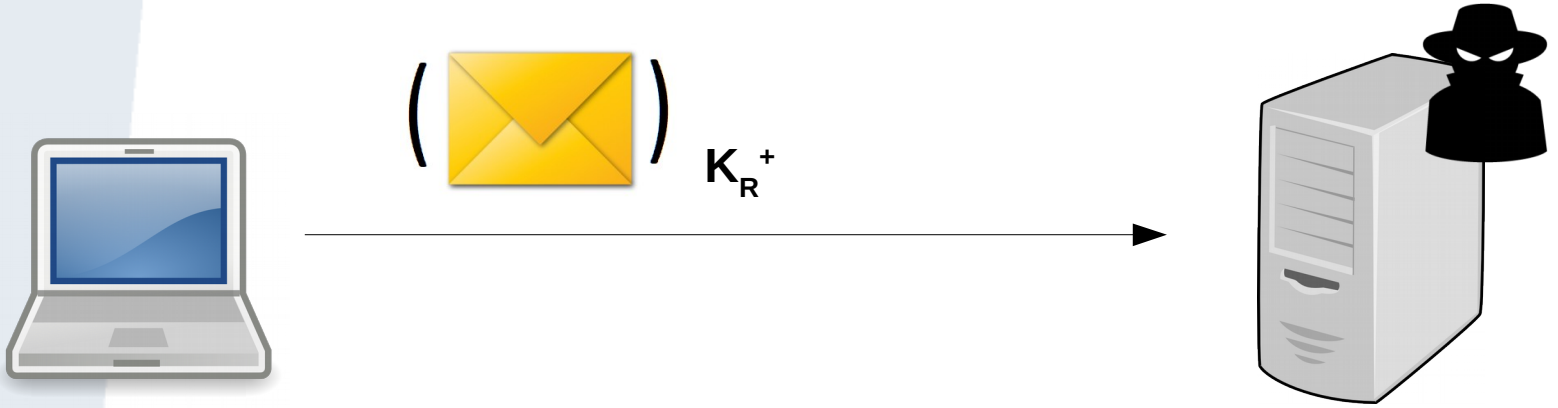
The Server: it receives messages encrypted with its public key (K_R^+). It can decrypt with its private key. Only the server can read the message encrypted with that key (PKI)

The Proxy: after decrypting its layer of encryption, it forwards the message to the next step of the routing

Assumptions

- All the communication channels between actors are protected with SSL (integrity, secrecy and server authentication are guaranteed)
- Protected (anonimous) communication is unidirectional.

Scenario #1 – Client – Server



1. Client encrypt the message with the receiver public key and sends to the server

2. Server receives the message, decrypt it with $\mathbf{K_R^-}$ and read the plaintext

The server machine is **compromised** by an attacker. The Attacker can intercept all client communication, decrypt, read the plaintext and **associate it to the IP address of the client**

Scenario #2 – Add 1 Proxy

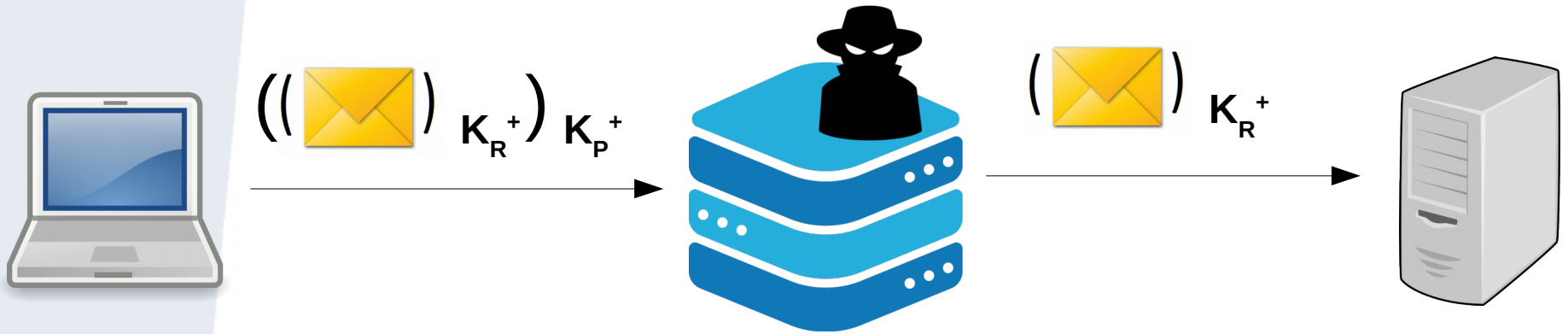
1. Client encrypt the message with the receiver public key and sends to the server

2. Proxy receives the message, decrypt it with K_P^- and forward to the server

3. Server receives the message, decrypt it with K_R^- and read the plaintext

The server machine is **compromised** by an attacker. The Attacker can intercept all proxy traffic towards the server, decrypt and read the plaintext **BUT can't associate it to the IP address of the client**.

Scenario #2 – Add 1 Proxy



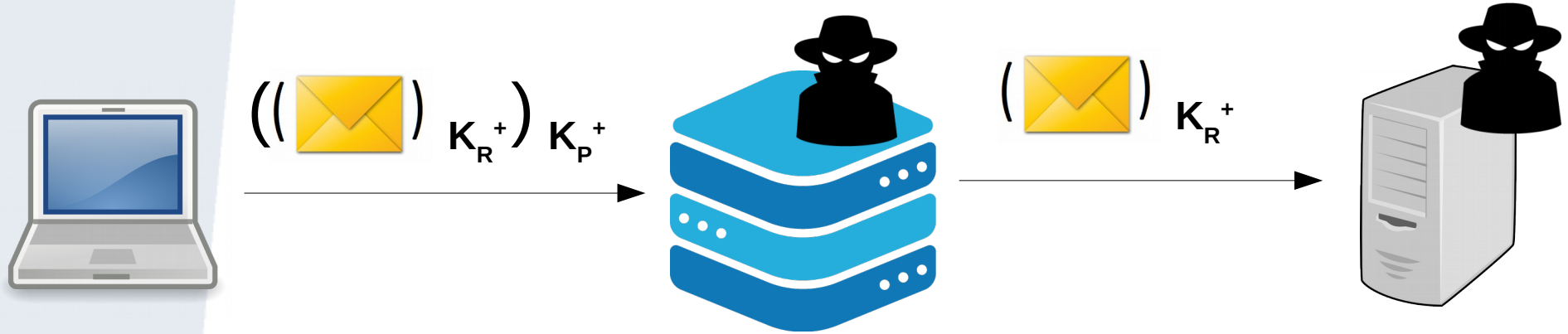
1. Client encrypt the message with the receiver public key and sends to the server

2. Proxy receives the message, decrypt it with K_P^- and forward to the server

3. Server receives the message, decrypt it with K_R^- and read the plaintext

The proxy machine is **compromised** by an Attacker. The Attacker can intercept all incoming proxy traffic, decrypt and read the client messages **BUT can't read what is the content of the messages**

Scenario #2 – Add 1 Proxy



1. Client encrypt the message with the receiver public key and sends to the server

2. Proxy receives the message, decrypt it with K_P^- and forward to the server

3. Server receives the message, decrypt it with K_R^- and read the plaintext

The proxy and the server machine are **compromised** by an Attacker. The Attacker can decrypt the whole client message using the K_P^- and the K_R^-. In this way the Attacker **succeeds in creating the <IPAddress, PlainText> association**

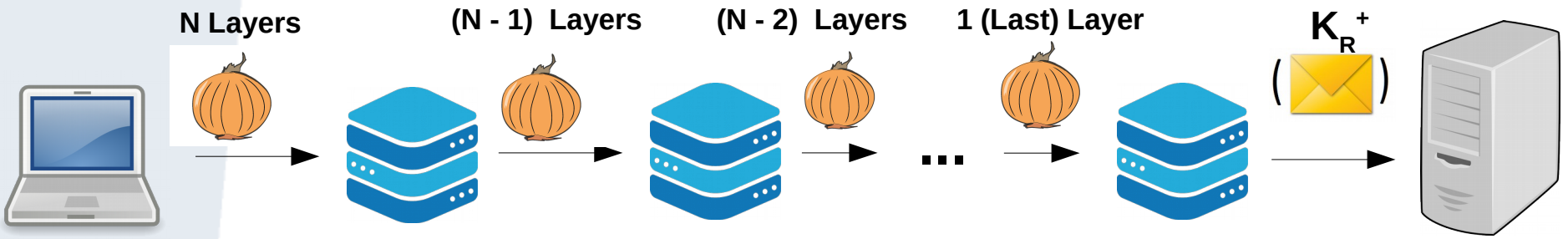
Scenario #2 – Add 1 Proxy

- With a proxy the security is **improved**. The Attacker needs to compromise 2 machines instead of 1 to obtain the association <IP Address, Message>
- What about adding more than 1 proxy?

Onion Routing

- In an onion network, messages are **encapsulated in layers of encryption**, analogous to layers of an onion. The encrypted data is transmitted through a series of network nodes called *onion routers*, each of which "peels" away a single layer, uncovering the data's next destination. When the final layer is decrypted, the message arrives at its destination.
- The sender remains anonymous because **each intermediary knows only the location of the immediately preceding and following nodes**.
- E.g existing network: Tor

Scenario #3 – Add N Proxies



1. Client encrypt the message with the receiver public key and sends to the server

2. Intermediate proxies **receive** messages, **decrypt** with their private key and **forward** the obtained message to the next specified destination

3. Server receives the message, decrypt it with $\mathbf{K_R^-}$ and read the plaintext

The Attacker needs to **compromise N** intermediate proxies to succeed in creating the association <IP, Message>

Proxy Network – Client Path Choice

- Given M proxies registered in the Registry Server, the client will choose randomly N of them and will envelop the message in this way:
- Given R the receiver server and chosen proxies are A, B, C, D in a pool of 100 available proxies registered in the ProxyNetwork:
  - The path will be:
    - **A → B → C → D**
  - The message (simplified) will be:

    (toB, (toC, (toD, (toR, (plaintext)$_{pubkR}$**)**$_{pubkD}$**)**$_{pubkC}$**)**$_{pubkB}$**)**$_{pubkA}$
  - And will be sent to the A intermediate proxy

Proxy Network – Proxy Behaviour

$(toB, (toC, (toD, (toR, (\underline{plaintext})_{pubkR})_{pubkD})_{pubkC})_{pubkB})_{pubkA}$

- On reception of a message:

  1) Decrypt the message using its private key

  2) Forward the decrypted message to the specified next destination (toX). The message will have 1 layer less

Note: Ephemeral Session (Message) Key

For each step of layer encryption, the encryption is made with the **encryption of a new generated symmetric key** K_s by the use of the public key K_x^+ of the proxy and **the encryption of the remaining message** (innerLevel made of the already encrypted steps) with K_s.

Layer decryption is made accordingly to that.

Comments

In this way, the Attacker needs to take control of **all the N proxies chosen by the client** to succeed in associating the message with the IP.

But, given that the client chooses randomly N nodes from a pool of M nodes, the attack will **certainly** be successful only if the Attacker **compromises all the pool of M nodes**.

So, as many nodes are available to be chosen for the route ***AND*** as many nodes the client chooses to put between itself and the receiver, as more robust will be the communication (from an anonimity point of view).

Increasing N (#hops of the route) **and M** (#available nodes)**, will increase the work of an attacker before succeeding in breaking the anonimity.**

But this is not secure at all, it's computationally secure.

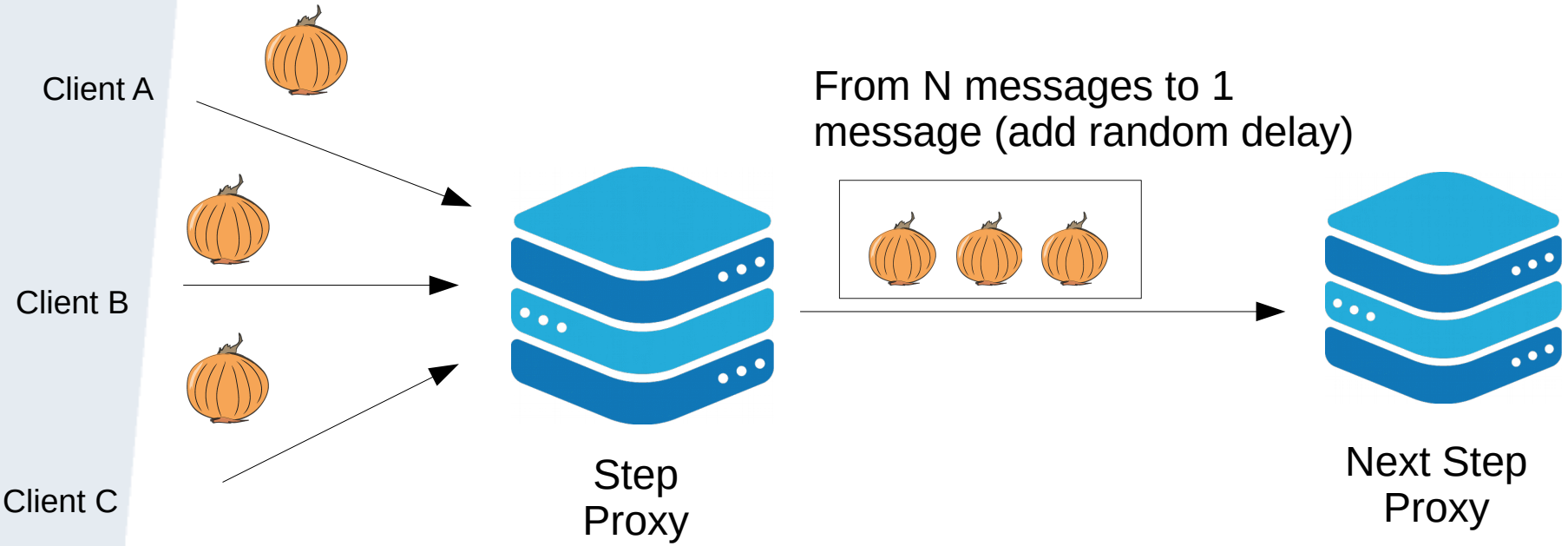
But it's not over yet!
Traffic Correlation Attack

The **Attacker** compromises the Server, observes the link to the Server and the link from the Client to the Proxy Network (client subnet). If there is not too much traffic towards the server, **the Attacker could correlate the packet outgoing from the Client subnet to the ingoing packet to the Server** by doing some size and latency related operations, **whatever the number of proxy of the path.**

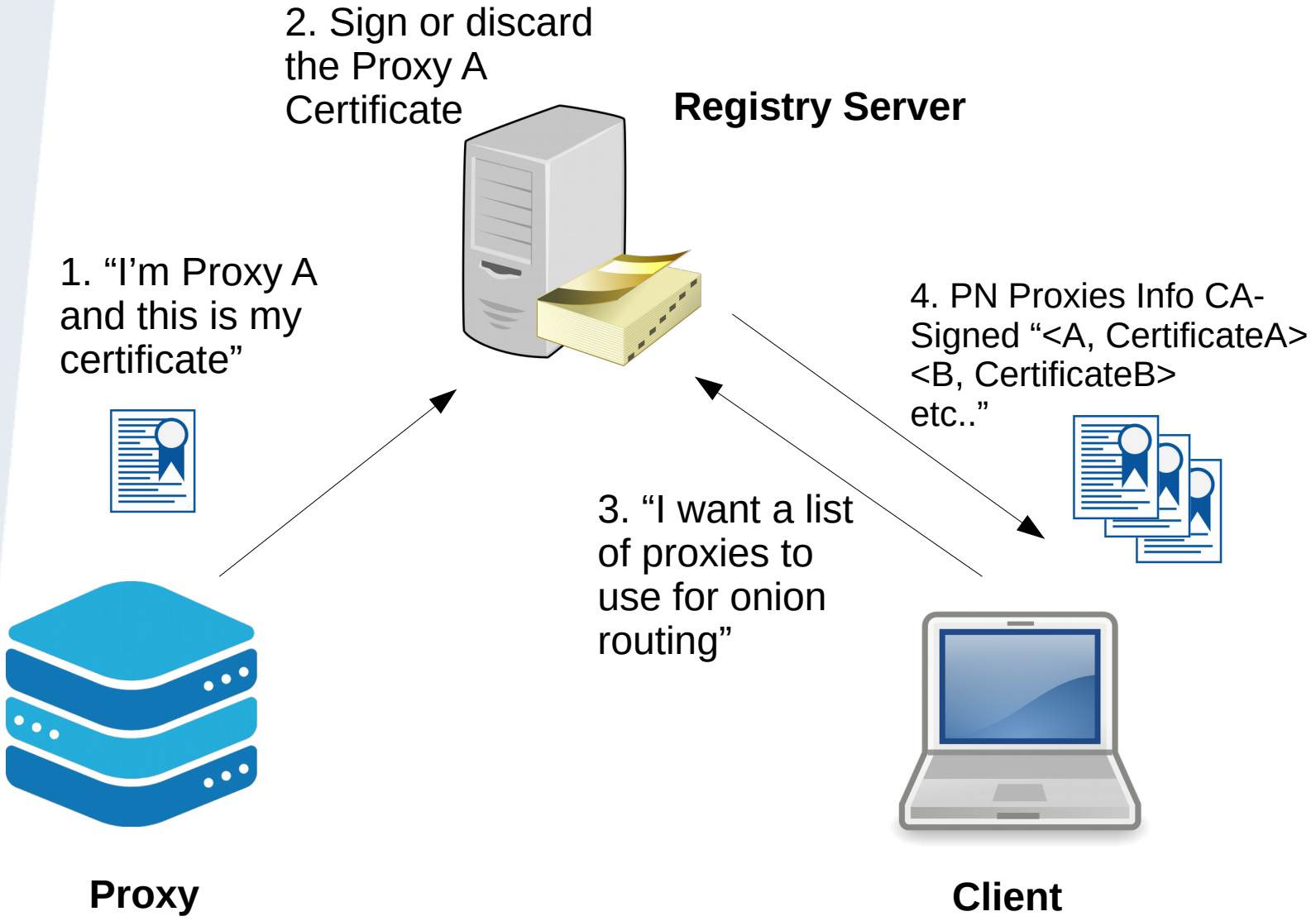
Solution: Garlic Routing – Add some resistance to Traffic Analysis

Generic Garlic Routing Def:

- bundling multiple messages together in a single *garlic message*.
- In **Proxy Network** interpreted as Threshold and Timed Mix batching strategy. Messages are not encrypted together but delayed and grouped together to randomize size and time details



PN: Proxies, clients and Registry Server - Initialization



PN Client API

```
// Create config. Choose Destination and number of steps

PNConfig config = new PNConfig();

config.destinationPublicKey = receiverKey;
config.IPAddress = IP_ADDRESS;
config.port = PORT;
config.routeSteps = 5;     // Increase to have stronger anonimity

// Init Client Endpoint
PNAPIClientEndpoint client = new PNAPIClientEndpoint(config);

// Send Message
client.sendMessage(mySerializableMessage);
```

Note: Enabling/ Disabling Garlic Routing is a config parameter of the network, not of the endpoints

PN Server API

```
// Generate key pair for the receiver
PNAPIServerEndpoint endpoint = new PNAPIServerEndpoint();

// Init endpoint and register handler for messages
endpoint.registerHandler(
    IP_ADDRESS,
    PORT,
    messageHandlerFunction,
    receiverPrivateKey
);

// Start listening
endpoint.start();
```