ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Decentralized Systems Engineering Project

A Secure Peerster

Julien Perrenoud, Loïc Serafin, Guillaume Vizier

December 12, 2018

Contents

1 Introduction

The initial implementation of Peerster contains several security flaws that a malicious node could exploit. First of all, the private messages are sent through the network in plain text. Therefore, it is very easy for any node on the direct path of a private message to intercept such messages and listen to the private conversation between two peers. Secondly, any node is able to register and announce any possible name to the network. Such a name is then used as is by other nodes, without checking if the node is actually and there is no check for duplicate and/or integrity, which leaves open the possibility of spoofing anyone on the network. Finally, the traffic from a node is relayed immediately by the neighbors, in a way that everybody is aware of the originator of a message. In this project, we aim to fix some of these flaws and overall increase the level of anonymity and privacy of users on Peerster. First, we will implement privacy of content via public key cryptography. Second, a basic mix network routing algorithm will be implemented to guarantee sender anonymity. Finally, we will use the already existing blockchain as a decentralized directory of public keys and mixer nodes.

2 Related work

2.1 Signature and encryption

Two main schemes exist for cyphering/decyphering messages, and for signing/verifying the signature: using a symmetric cryptographic scheme, or an asymmetric one.

Symmetric cryptographic schemes use the same key on both participants to the exchange of messages. The issue in our case is that we would need to create one key for each pair of peers in our network, leading to a combinatory explosion of the number of keys. In addition, two nodes would need to exchange their common key through a secure channel, which can only be created **after** they agreed on a key to use.

Asymmetric schemes make each user create a **pair** of keys: a public key and a private key. The public key is for public usage: its purpose is to be shared with everybody on the network (and beyond). The private key has to be kept secret by its owner. In this case, a node signs a message using its private key, and every node having the public key can verify the signature. To cypher a message, a node uses the public key of the destination, and only the private key can reverse the operation.

One issue remains: the transmission of the keys. Using an asymmetric encryption scheme, we do not bother with exchanging the keys in a secure way, because only the public key, which is common knowledge, is sent. However, the mapping *gossiper name* → *public key* is still problematic. Indeed, as of today, any node can transmit any received public key as belonging to it.

In the web of trust, it is advised to physically meet the person we are exchanging keys with. In our setting, we will propose a transaction containing a public key, along with the name of the owner and this name signed with the private key corresponding to the public key. Signing one's public key before sending it on a keys server is a practice advised for distributed systems of trust.

2.2 Mix Networks

Mix networks are a class of anonymous and hard-to-trace routing protocols initially described by David Chaum in 1981 paper *Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms* [1]. These protocols aim to implement **forward anonymity** (i.e. concealing the identity of the sender of a message) by routing the message through a series of relay (or "mixer") nodes. Each such node will remove information about the origin of the message and place it in a queue. Once a specific condition is met, the mixer node will shuffle the queue and forward the messages in a different order to the next relay. Finally, it is important to note that before a message is sent on a mixnet, it needs to be encrypted behind several layers of security (one for each mixer node on the path, who will decrypt it one layer at a time). Besides the obvious task of making the input of a shuffle indistinguishable from its output, this multi-layer encryption-decryption scheme (often called an *Onion*) ensures that a given message will follow its intended path and that the mixer nodes cannot read its content

In practice, mix networks were and are still often used as a way to anonymously browse the web or communicate via e-mail. This second kind is often distinguished in three[1] types. **Type I** (or *Cypherpunk*) remailers provide a simple implementation of Chaum's mix network. However, the implementation contained several security flaws. Mainly, the length of the original message was kept the same, and the messages were always forwarded with a fixed delay (if any). This made it possible to track the path of a given message through the network by looking at the lengths and timings of the messages exchanged between the mix nodes. These issues were addressed in 1995 by Lance Cottrell who released *MixMaster*, an implementation now referred to as a **type II** remailer. Finally in 2002, a **type III** remailer known as *MixMinion* [2] further improved on MixMaster. Amongst other things, MixMinion allows users to specify single use reply blocks that enable the recipient to anonymously respond to an incoming message.

Lastly, it would be hard to discuss anonymous routing protocols without mentioning **Tor** (shorthand for *The Onion Router*). Initially describe in [3], Tor is also derived from Chaum's original mix network and is in some ways very similar to the concepts explained above. The main difference is that a Tor relay nodes will simply wait a random delay before forwarding an incoming message, without performing any kind of shuffle. The main advantage of this is a big reduction in latency. Indeed, classic mixer nodes - who wait for a given number of incoming messages before forwarding them - might end up causing arbitrarily large delays in the delivery of a message. On the other hand, there is a clear trade-off with the security of the system, as one can argue that not shuffling messages allows one to trace a particular connection more easily. This is fair, however the performance of Tor is based on two key assumptions. First, the network should always be fairly busy. This will help disguise the traffic as it blends in with requests from other peers. Secondly, it is unlikely that someone will get access to the logs of every one of the nodes on the path of a message (which would be required to perform a successful attack on the message).

[1] This does not consider type 0 remailers comprised of a single node, as it is less relevant here.

2.3 Mixer Node Location and Storage of Keys

Even with a list of public keys and mixer nodes, one still needs to trust these data. They might be forged by malicious users in the network and are hardly detectable.
To ensure the security of communication data in he network, there are several ways to proceed. First of all, we can of course give the example of the physical meeting between the different users of the networks as well as those hosting mixer nodes to exchange public keys and mixer nodes addresses. But this kind of solution is never used practically speaking for user convenience. Another solution is to have a trusted third party that distributes to all users required data.
In *Cypherpunk* and *Mixmaster* (types I & II remailers), those third parties are hosted by trusted groups. For example, a list of available and trusted remailers can be found at all times on noreply.org[2]
In Mixminion[2], the third party is a set of directory servers managing a *Distributed Hash Table* (DHT) possibly hosted by the users themselves. These directory servers together manage the state of the mixer node network and the keys of all nodes in the system. This system generally makes it possible to distribute reliable data, but several attacks remain possible, in particular by exploiting the difference in knowledge of the different directory servers. The explanation for the attack is given in Mixminion design[2] paper in section 6.

3 System goals and functionalities and architecture

In this section, we will first discuss the goals of the improvements we plan to make on top of our Peerster implementation(s). We will then precise the foreseen functionalities to reach these goals, to finally discuss the architecture of our system.

3.1 System goals

a Our main goal is to improve the security and the anonymity of the Peerster.
Starting with such an unsafe specification, and given the little time we have to put our ideas in practice, we had to make tough choices. Thus we decided to focus on the cyphering and signing of all exchanges, with a mapping *gossiper name* → *public key* stored in the blockchain[3] for different nodes to know the same peers.
The second main goal of the project is to anonymize users' message by using the Mix-network system proposed by David Chaum[1], creating a route of several mixing nodes before reaching a message recipient, and shuffling messages in random order so that it could not be traced back in time.

3.2 Functionalities

- **Public Key Cryptography** - We will add, during the starting process of a node, the creation of a pair (public key, private key), along with the creation of a transaction, broadcast to the network, for the public key to be known by other peers in the network. We will still be

[2] `http://www.noreply.org/echolot/rlist2.html`

[3] We are aware that the so-called "blockchain" is not actually a blockchain, but we have to make concessions here.

vulnerable to man-in-the-middle attacks, but if we assume that the public keys are correctly linked to their possessor, as we assume that private keys cannot be forged or stolen, we should be safe against packet modifications and creation.

- **Security** - We will also replace the current calls to `WriteToUDP` by calls to a function to be implemented, that will cypher and sign a message. Also, we will replace the reception of a message (namely `ReadFromUDP`) by a call to another function, also to be implemented, that decypher and verify the signature of any received packet.

- **Blockchain-Based Key Directory** - It is not enough to create public and private key pairs for the network to be secure. Having all gossipers exchange directly their public keys through the Peerster network has many flaws, e.g. a simple man-in-the-middle attack, intercepting public key exchange and giving his own public key to catch all messages from the conversation. We plan to reuse our implemented blockchain to store everyone's public key alongside the current filenames mapping. This shared structure will prevent such attacks or at least make them detectable. We also need to have a trusted third-party to hold the list of mixing nodes and the list of pseudonyms.

- **Download Chain History** - In order to allow nodes joining after the creation, we will modify the current implementation of the blockchain by allowing the creation of a genesis block and the ability for a node to download the entire history of the chain from the previous nodes. We propose to build this using the gossip mechanism already in place, using new type of messages that will be exchanged between nodes arriving in the network and those already up to date.

- **Mixer Nodes** - Our first step towards the implementation of a mix network will be to upgrade each node to a potential mixer node. That means, each node should be capable of receiving, batching, deciphering, shuffling and forwarding messages to another node. In order to batch messages properly, mixer nodes will implement a simple threshold policy: waiting until a constant number of messages has been reached and then simply outputting a message at random.

- **Anonymous Routing** - Once mixer nodes are in places, we will give the nodes the possibility to conceal all of their communications by routing them through a series of nodes. This will be implemented via a new message type `OnionPacket` that wraps around a regular `GossipPacket`. When a node receives and deciphers such a packet, it can detect whether the packet is fully decrypted or not. If it is, the node will process its content as a regular `GossipPacket`. If it is not, it will forward it to the next node in the chain.

- **Blockchain-Based Mixer Nodes Directory** - To plan a routing path through several mixer nodes, one must know in advance multiple mixer nodes addresses. As for public keys, we decided to choose the blockchain to store the addresses of all mixer nodes when these mixer nodes manifest as such.

3.3 Architecture

3.3.1 Cryptography

As the introduction of cryptography in the Peerster touches almost only the lowest levels, it seems clear what the architecture changes will be: a message-passing structure, dedicated to the direct sending and receiving of messages. Also, the already existing blockchain handling part of the Peerster will be modifier (as already mentioned in Section3.1) to match the new requirements. We did not yet have the time or the energy needed to decide whether we will replace the file handling altogether, or simply enrich the structure and complexify the implementation.
We will assume that the readers of this report have the minimum required knowledge of asymmetric cryptography schemes, and will spare them the usual drawings about how Alice and Bob can cypher, decypher, sign, and verify the signature of, a message.
As using a blockchain to ensure the unicity of items[4] was described in Homework 3, we will not describe it here again. We also hope that the flows of this design have already been discussed with enough details for us not to need to report these here again should provide you with far more than enough information.
This part will be Guillaume's responsability.

3.3.2 Anonymous Routing

In order to seamlessly integrate the anonymous routing into the existing implementation, we will build the routing logic between the current UDP transport (and security layer) and the existing application layer. Practically, this means that any node wishing to use anonymous routing will be able to do so by encapsulating a regular `GossipPacket` into a new type of message called `OnionPacket` (described in 3.3.3). The `OnionPacket` will then be routed and forwarded using the mixing and decryption logic described in 3.4 until it reaches its final destination. At this point, the exit node will be able to identify that there is no more mixer node in the path and process the underlying `GossipPacket` normally.

3.3.3 Onion Structure

Drawing inspiration from MixMaster, we would like to ensure that packets are uniformly-sized so as to avoid tracking using the length of a message. The size of an `OnionPacket` will therefore always be exactly TBD. Furthermore, these bytes are separated into an initial `Header` followed by a `Payload`.

```go
type OnionPacket struct {
    Header  [N]byte
    Payload [M]byte
}
```

In a manner similar to Mixminion, the header will be split into 16 uniformly-sized byte chunks. After decrypting the layer, the first of these byte chunks should be deserializable into the following `SubHeader` structure.

[4] Namely, the names of the files available on the network.

```
type SubHeader struct {
    PrevHop NodeKey
    NextHop NodeKey
    Hash    [32]byte
}
```

3.4 Mixer Logic

Upon receiving a new `OnionPacket`, a mixer node is then expected to do several things.

1. First, it peels one layer by decyphering the entire `OnionPacket` using its private key.
2. Afterwards, it selects the first byte chunk of `Header` and tries to deserialize it into a `SubHeader`. If this fails, the packet is dropped silently.
3. Third, it computes the hash of `Payload` and verifies that it corresponds to the `Hash` found in `SubHeader`.
4. Now, the `NextHop` field is empty, in which case we know that we are the last hop on the route. In this case, we can deserialize the `Payload` into a regular `GossipPacket` and process the packet as one would normally process it from a direct peer. This ends the process.
5. Or, `NextHop` contains the public key of the next node on the route, in which case we need to forward the `OnionPacket`. At this point, we then "move up" all byte chunks in `Header` by one position. This ensures that the next hop will find its header in the first chunk as well, and effectively erase the routing information from the packet.
6. The packet now ready for re-distribution, we simply keep it in a queue with other messages. This queue will grow until it reaches a certain threshold of messages k. When the queue size is above k, the mixer node waits a random amount of time before selecting an entry at random and forwarding it to its next hop.

3.4.1 Keys Registration & Routing

The first step regarding the decentralized blockchain structure will be to allow a node to join the network at any time and therefore to modify its behaviour. A real genesis block will be introduced and will therefore not leave the possibility of parallel blockchains with different "genesis" block as we had in homework 3.
Two types of messages will be added to the Peerster protocol to make blockchain update requests:

- `BlockRequest` used to ask another node for a `Block` with hash `Hash`. Typically, it will be sent for any unknown block ancestor.

```
type BlockRequest struct {
    Origin   string
    Dest     string
    HopLimit uint32
    Hash     []byte
}
```

- BlockReply in response to the previous BlockRequest, giving the request block. A field is dedicated to specify it is the Genesis block, and thus to allow the requester to end his blockchain reconstruction.

```
type BlockReply struct {
   Dest     string
   HopLimit uint32
   Block    *Block
   Genesis  bool
}
```

On top of these new messages types, transactions stored in our blockchain will be changed to allow the storage of nodes public keys and mixer nodes addresses.

```
type TxPublic struct {
   File      File
   NodeKey   NodeKey
   MixerNode MixerNode
   HopLimit uint32
}

type NodeKey struct {
   Name string
   Key  string
}

type MixerNode struct {
   Address string
   Key     string
}
```

Once the network of mixer nodes has been set up, a regular node will be able to choose the route of its anonymous message. A function will allow you to choose the required length of your route and will randomly select mixer nodes to form this pre-computed route. The longer the chosen length, the more difficult it will be to determine the source of the message (as long as not all the selected nodes are malicious), but also the longer the transmission time will be.

For security reasons, it will be important to define a threshold of intermediary mixing nodes. If too few nodes are available, anonymity will not be guaranteed. We intend to simply drop the anonymous message in this case.

After choosing all intermediate mixer nodes in the path to the final destination, the sending node will have to encrypt its message N times for the N nodes present in this path, respecting the order of the path including the message recipient. Each mixing node will then decypher the message with its own private key before forwarding it to the next mixing node on the route. It will eventually reach the message recipient that will decypher also the anonymous message with its own private key.

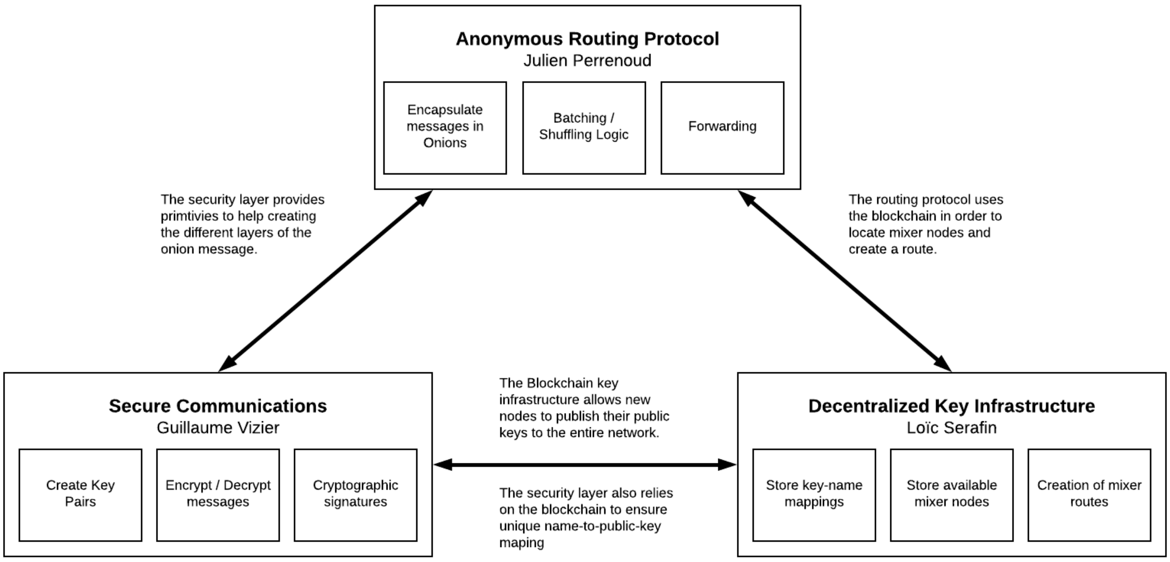
Anonymous Routing Protocol
Julien Perrenoud
Encapsulate messages in Onions
Batching / Shuffling Logic
Forwarding
The security layer provides primtivies to help creating the different layers of the onion message.
The routing protocol uses the blockchain in order to locate mixer nodes and create a route.
Secure Communications
Guillaume Vizier
Create Key Pairs
Encrypt / Decrypt messages
Cryptographic signatures
The Blockchain key infrastructure allows new nodes to publish their public keys to the entire network.
The security layer also relies on the blockchain to ensure unique name-to-public-key maping
Decentralized Key Infrastructure
Loïc Serafin
Store key-name mappings
Store available mixer nodes
Creation of mixer routes

Figure 1: Interaction between the three sub-components

3.5 Task share diagram

References

[1] David L. Chaum, *Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms*, 1984.

[2] George Danezis, Roger Dingledine, Nick Mathewson, *Mixminion: Design of a Type III Anonymous Remailer Protocol*, 2002.

[3] Roger Dingledine, Nick Mathewson, Paul Syverson, *Tor: The Second-Generation Onion Router*, 2002