Peer to peer Networks / Onion Routing

Kiryl Buloichyk
https://github.com/kiiril/Onion

Introduction

Onion routing is a technique for enabling anonymous communication over a computer network. In an onion network, messages are encapsulated in multiple layers of encryption, similar to the layers of an onion. These encrypted messages are transmitted through a series of network nodes known as "onion routers." As each message passes through a node, that node removes a single layer of encryption, revealing the next destination in the route. Importantly, each node knows only its immediate predecessor and successor, but cannot access the content of the message, as it remains protected by the remaining layers of encryption. When the message reaches the final node, it is fully decrypted and sent to the server, effectively breaking the link between the source of the request and the destination. This makes it significantly more difficult for eavesdroppers to trace end-to-end communications.

The following paragraphs will describe the key components of the onion network, the challenges encountered, and the solutions implemented.

Encryption/decryption mechanisms

Challenges:

- Utilizing cryptographic algorithms to securely encrypt the message in multiple layers

Implementation: To efficiently manage the encryption and decryption of large messages, symmetric cryptography is employed. Symmetric algorithms, such as AES, allow for the encryption and decryption of messages in layers, while accommodating increasing message sizes. The AES algorithm is a widely used symmetric encryption algorithm that provides strong security and efficiency. It uses the same key for both encryption and decryption, making it well-suited for scenarios where the key can be securely shared between communicating parties.

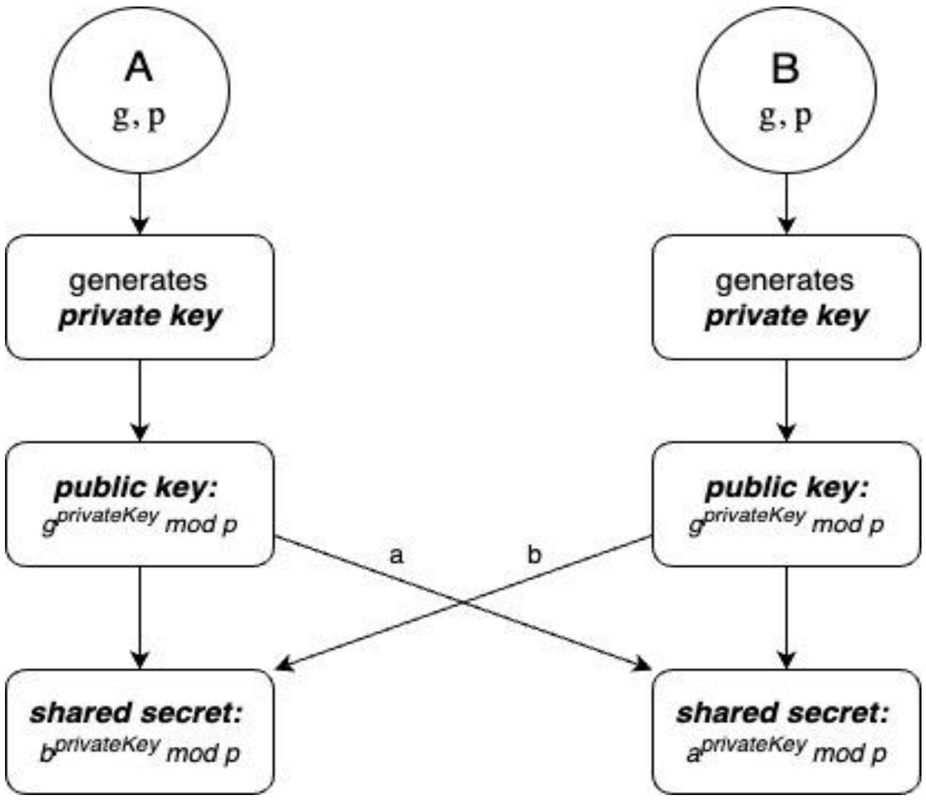
Key management

Challenges:

- Implementing a secure method to establish and manage symmetric keys used for encrypting and decrypting messages.
- Generating and distributing session keys to handle the encryption and decryption of each layer within a multi-layered message structure.
- Managing keys to be correctly used for each incoming message

Implementation: The Diffie-Hellman algorithm facilitates the secure establishment of a shared secret between two parties. The generated shared secret serves as a symmetric key for encrypting messages before sending and decrypting them upon receipt.
Steps in Diffie-Hellman key exchange:

1. **Agree on public parameters:** Both parties agree on public parameters, specifically a prime number p and a base g, also known as the modulus and generator. These parameters are publicly shared and used in subsequent computations (hardcoded in the implementation).
2. **Generate private and public keys:** Each party independently generates a private key (a random integer) and computes a corresponding public key. This public key is then sent to the other party.
3. **Compute the Shared Secret:** Upon receiving the other party's public key, each party uses it, along with their own private key, to compute a shared secret.


A
g, p
B
g, p
generates *private key*
generates *private key*
public key: $g^{privateKey}$ *mod p*
public key: $g^{privateKey}$ *mod p*
a
b
shared secret: $b^{privateKey}$ *mod p*
shared secret: $a^{privateKey}$ *mod p*

To handle the encryption and decryption of each layer of the message, session keys are utilized (also symmetric keys). The requesting peer first generates session keys for encrypting each layer of the message and keeps them for decrypting the response. These session keys are then passed down the chain, with each peer storing the key associated with the session ID. Each peer uses this key to decrypt their specific layer when the message reaches them.

Onion Data Structure

Challenges:

- Constructing a secure onion data structure that encapsulates the message in layers of encryption.
- Ensuring that each layer contains the appropriate instructions for the next node in the path.
- Deconstructing the onion structure securely at each node to prevent any information leakage.

Implementation: The onion data structure is composed of multiple encrypted layers, with the innermost layer containing the actual message. Each outer layer holds information about the previous and next peers in the chain, ensuring that as the message passes through each peer, one layer is decrypted to reveal the routing information while maintaining the confidentiality of the original message. The encryption and decryption of each layer are performed using session keys that are established prior to sending the message through the chain. Additionally, before sending the message to the next peer, it is encapsulated with a session ID, which every peer uses to identify the correct session key for encryption/decryption.

Peer Discovery

Challenges:

- Identifying and connecting nodes in a decentralized network
- Ensuring proper management of known peers

Implementation: Each peer begins with the hardcoded knowledge of the network's initial peer, known as the broadcaster. The Diffie-Hellman key exchange process is initiated to establish a symmetric key. Upon successful completion, the broadcaster is added to the list of active connections. Additionally, a new thread is spawned to handle incoming messages from the broadcaster.

A separate thread is also allocated to listening for incoming connections. When a new peer connects, the Diffie-Hellman key exchange process is repeated to establish a secure connection. The new peer is then added to the network, and existing peers are notified of this addition. Information about the currently known peers is also shared with the newly connected peer.

Routing

Challenges:

- Correctly forwarding the message through the network without revealing the entire route and message itself
- Ensuring that the response reaches the original sender while preserving the same level of anonymity and security

Implementation: Routing process operates as follows: After encrypting all message layers and placing them into a wrapper, the sender encrypts the entire message with the symmetric of the first peer in the chain and sends it. Upon receiving the message, the first peer decrypts it using its symmetric key associated with the sender. The revealed session ID is then used to retrieve the session key, allowing the peer to decrypt its assigned layer. The decrypted information is used to determine whether the peer is the final router or if there is a subsequent peer in the chain. If there is a next peer, the message is wrapped again, encrypted with the symmetric key of the next peer, and forwarded. If the peer is the last in the chain, the message is fully decrypted, and the peer proceeds to make the request to the server. The server's response is then sent back through the chain, with each peer using the session ID to determine the next recipient and the corresponding session key to encrypt its layer. Finally, the requesting peer decrypts all the layers to retrieve the server’s response, ensuring complete anonymity throughout the process.