Harvard John A. Paulson School of Engineering and Applied Sciences

WHERE SCIENCE AND ENGINEERING CONVERGE

CS262 Final Project: Tor-Based Privacy Preserving Network

April 2023

Hileamlak Yitayew*
ehmyitayew@college.harvard.edu

Jeremy Zhang*
jeremyzhang@college.harvard.edu

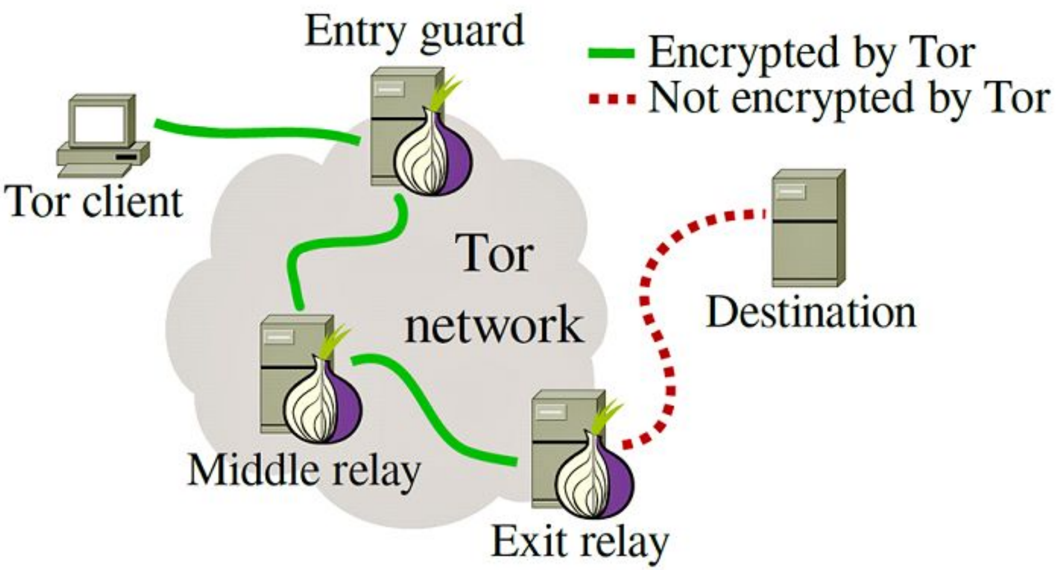
*All authors are contributing equally to this project and are listed in alphabetical order.

Motivation

- Privacy and anonymity among participants within a distributed network have been of paramount importance for the end user to ensure their safety.
    - Protect citizens during times of civil unrest (Arab Spring 2011)
    - Help soldiers communicate securely (war in Ukraine)
- In popular culture, Tor has unfortunately become synonymous with nefarious activities and the dark web. However,
    - It presents an interesting distributed systems problem worth studying
    - Demystify and clarify misunderstandings around Tor among students

Tor

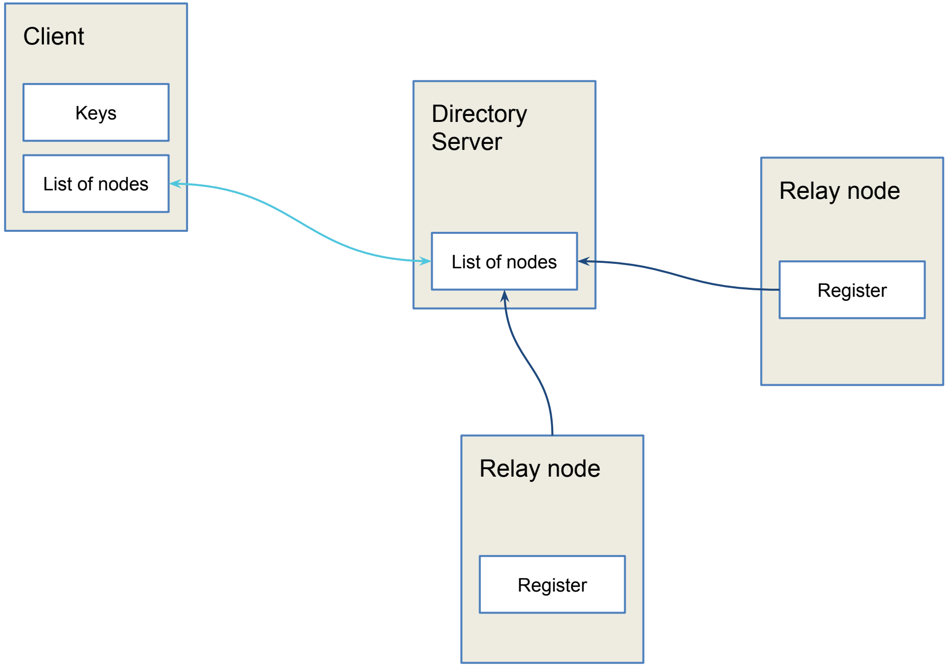
Tor


Entry guard
Encrypted by Tor
Not encrypted by Tor
Tor client
Tor
network
Destination
Middle relay
Exit relay

Project Goals

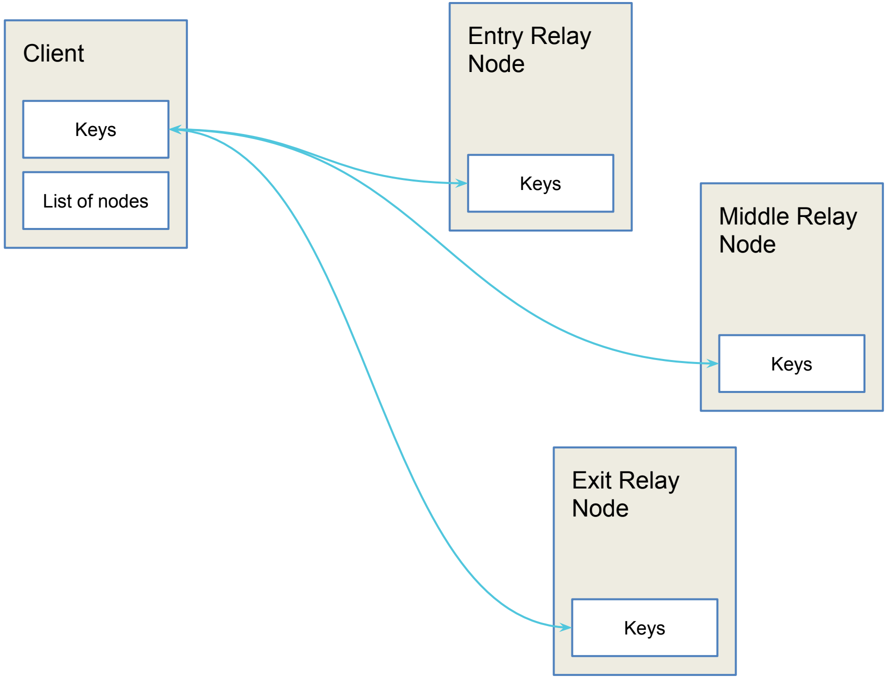
- Tor-like privacy network:
  - Client, server, 3 intermediate/relay nodes (at least 1 entry node, 1 middle node, 1 exit node)
  - Onion encryption – intermediate nodes do not see contents of message
- Fault Tolerance:
  - Detect relay node failures and replace them when they fail
  - Secure key exchange mechanism
  - Directory server listing relay nodes
- Out of scope:
  - Tor web browser
  - Packet routing and congestion control

Step 1: Query Directory Server



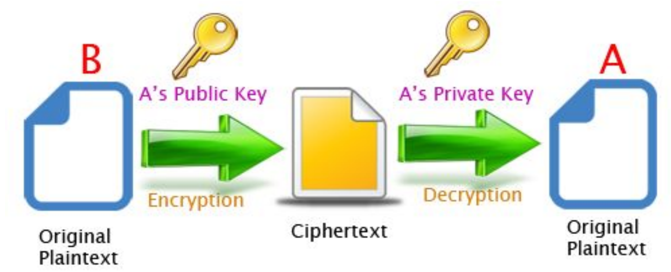
- On start, the client contacts the directory server
- Directory server returns a list of all available relay nodes and their type (entry, middle, or exit)

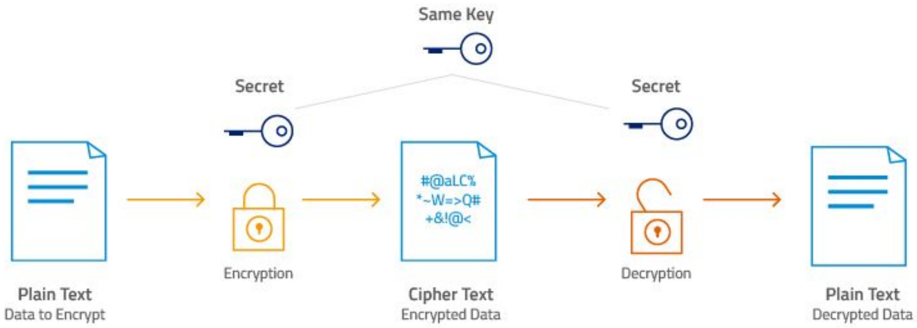
Step 2: Key Exchange



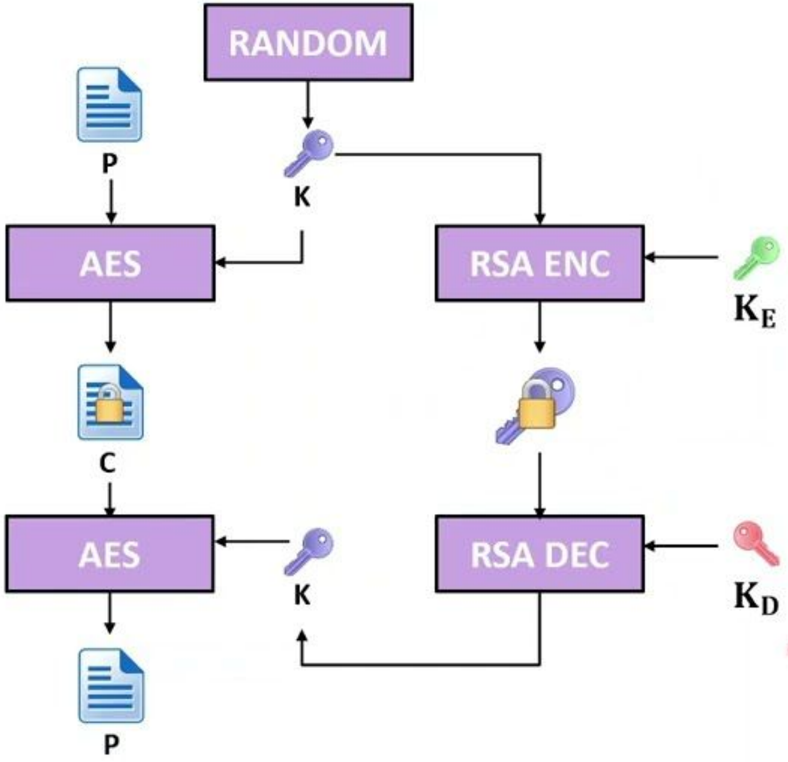
- Hybrid encryption
- Client generates 3 cryptographic keys
- Each relay has its own public private key pair
- Client contacts each relay node to share cryptographic keys,
- Send specific public keys and
- Receive public key from each relay

RSA vs AES


B
A's Public Key
Encryption
Original
Plaintext
Ciphertext
A's Private Key
Decryption
A
Original
Plaintext

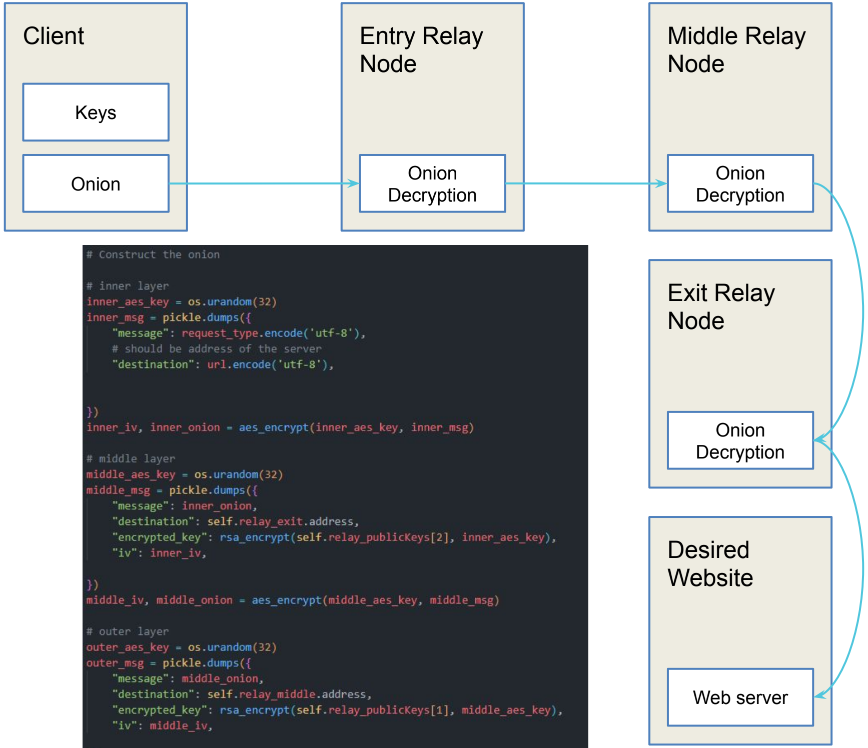

Same Key
Secret
Secret
#@aLC%
*~W=>Q#
+&!@<
Encryption
Decryption
Plain Text
Data to Encrypt
Cipher Text
Encrypted Data
Plain Text
Decrypted Data

Hybrid Encryption: Generic example using AES and RSA



- **Sender:**
  - Create symmetric key K
  - Send data encrypted using key K
  - Send key K encrypted with public key K_E
- **Receiver:**
  - Obtain symmetric key K by using private key K_D
  - Obtain data by using symmetric key K
- **Main problem:**
  - Once K_D is known, all symmetric keys K can be obtained, and all data can be decrypted
  - Can be prevented by using ephemeral Diffie-Hellman key exchanges EDH or ECDHE

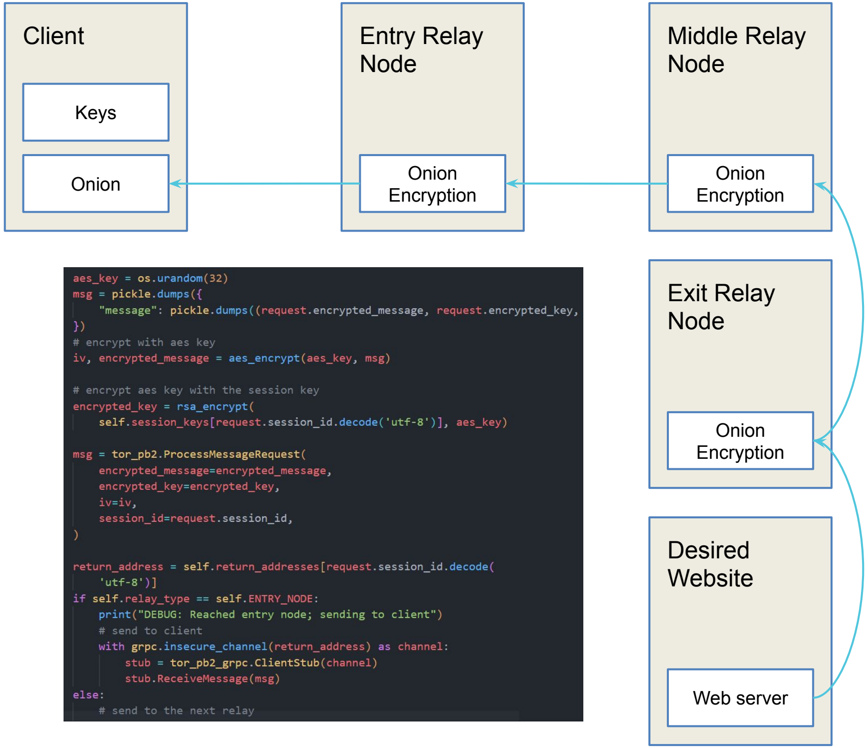
Step 3: Send Message




```python
# Construct the onion

# inner layer
inner_aes_key = os.urandom(32)
inner_msg = pickle.dumps({
    "message": request_type.encode('utf-8'),
    # should be address of the server
    "destination": url.encode('utf-8'),


})
inner_iv, inner_onion = aes_encrypt(inner_aes_key, inner_msg)

# middle layer
middle_aes_key = os.urandom(32)
middle_msg = pickle.dumps({
    "message": inner_onion,
    "destination": self.relay_exit.address,
    "encrypted_key": rsa_encrypt(self.relay_publicKeys[2], inner_aes_key),
    "iv": inner_iv,

})
middle_iv, middle_onion = aes_encrypt(middle_aes_key, middle_msg)

# outer layer
outer_aes_key = os.urandom(32)
outer_msg = pickle.dumps({
    "message": middle_onion,
    "destination": self.relay_middle.address,
    "encrypted_key": rsa_encrypt(self.relay_publicKeys[1], middle_aes_key),
    "iv": middle_iv,

})
```

- Client generates the onion (layered encryption packet)
- Client sends onion to entry node
- Entry node decrypts outer layer of onion, sends reminder to middle node
- Middle node decrypts the next layer of the onion, sends reminder to exit node
- Exit node decrypts last layer of onion, interacts with internet

Step 4: Receive Message




```python
aes_key = os.urandom(32)
msg = pickle.dumps({
    "message": pickle.dumps((request.encrypted_message, request.encrypted_key,
})
# encrypt with aes key
iv, encrypted_message = aes_encrypt(aes_key, msg)

# encrypt aes key with the session key
encrypted_key = rsa_encrypt(
    self.session_keys[request.session_id.decode('utf-8')], aes_key)

msg = tor_pb2.ProcessMessageRequest(
    encrypted_message=encrypted_message,
    encrypted_key=encrypted_key,
    iv=iv,
    session_id=request.session_id,
)

return_address = self.return_addresses[request.session_id.decode(
    'utf-8')]
if self.relay_type == self.ENTRY_NODE:
    print("DEBUG: Reached entry node; sending to client")
    # send to client
    with grpc.insecure_channel(return_address) as channel:
        stub = tor_pb2_grpc.ClientStub(channel)
        stub.ReceiveMessage(msg)
else:
    # send to the next relay
```

- Information is returned from desired web server to exit relay node
- Exit relay node encrypts information, sends to middle relay node
- Middle relay node encrypts information, sends to entry relay node
- Entry relay node encrypts information, sends to client

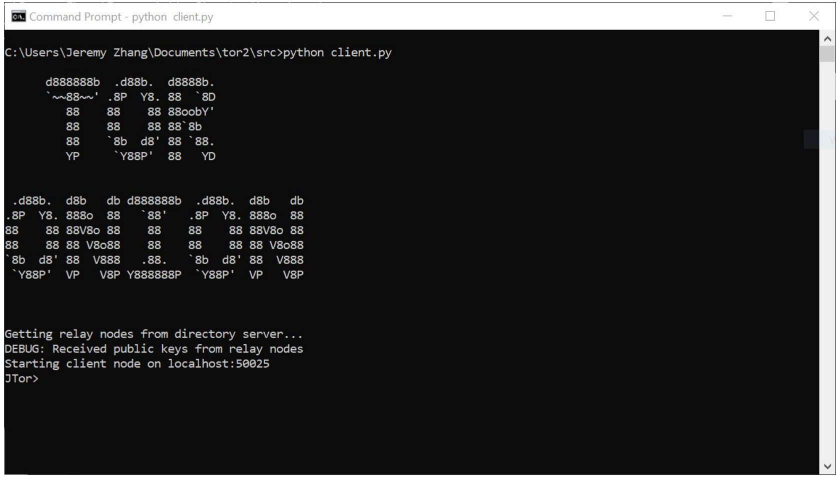
Step 5: Peeling the onion

```
aes_key = rsa_decrypt(
    self.client.privateKeys[0], request.encrypted_key)
# # decrypt message with aes key

first_layer = pickle.loads(aes_decrypt(
    aes_key, request.iv, request.encrypted_message))
first_layer_msg = pickle.loads(first_layer['message'])

      You, 1 second ago • Uncommitted changes
# second layer of peeling
encrypted_msg, encrypted_key, iv = first_layer_msg
aes_key = rsa_decrypt(self.client.privateKeys[1], encrypted_key)
second_layer = pickle.loads(aes_decrypt(
    aes_key, iv, encrypted_msg))
second_layer_msg = pickle.loads(second_layer['message'])

# third layer of peeling
encrypted_msg, encrypted_key, iv = second_layer_msg
aes_key = rsa_decrypt(self.client.privateKeys[2], encrypted_key)
third_layer = pickle.loads(aes_decrypt(
    aes_key, iv, encrypted_msg))
third_layer_msg = pickle.loads(third_layer['message'])

headers, content = third_layer_msg
```

- Client receives a message
- Client uses each of the private keys to decrypt the aes keys
- And use the aes keys to decrypt each layer of message

Demo

- Setup
- Sending a message
- Receiving a message



Future Work

- Fault tolerance (only fail stop) – more relay nodes, detect relay node failures, routing around failed nodes
- Hosting of servers on the cloud instead of localhost
- Multiple tab support
- Session persistence (Tunneling)
- Relay auto registry
- Directory server liveness test
- Security analysis

Harvard John A. Paulson
School of Engineering
and Applied Sciences

Thank you!