ITO-5163 Formative Submission

Hugo Mainland

I'll demonstrate the evolution of secure communication from an insecure baseline to a fully authenticated and confidential channel. The system models four actors on a network: Alice, Bob, a Man-in-the-Middle (MITM), and a Certificate Authority (CA). All interactions will take place over real HTTP using Flask servers, with a separate animation engine receiving events to visualise message flow.

Requirements

1. Real Networking

- All communication between Alice, Bob, MITM, and CA must use genuine HTTP requests.
- Each actor runs as an independent Flask server in its own terminal to visibly demonstrate network activity.

2. MITM Visibility and Interception

- All traffic must pass through the MITM server to simulate an untrusted network
- MITM must be able to read, log, modify, or substitute messages as the demonstration progresses.

3. Stepwise Security Progression

The demonstration must show, in order:

- Plain HTTP communication without any protection.
- MITM intercepting and modifying plaintext messages.
- Weak Diffie-Hellman key exchange (low prime) and how an attacker can break it.
- Strong Diffie–Hellman key exchange providing confidentiality but lacking integrity and authentication.
- Introduction of RSA signatures to sign Diffie–Hellman public values, brining integrity and preventing key substitution.
- Introduction of a Certificate Authority for trusted distribution of RSA public keys.

4. Certificate Authority Trust Model

- Alice and Bob must have the CA’s public key pre-installed or hard-coded.
- CA certificates can pass through MITM, but MITM cannot forge or tamper with them due to the CA’s signature.

5. Integrity

- Messages must be immutable once signed.
- Any modification by the MITM should be detectable once RSA signatures and certificates are introduced.

6. Authentication

- Alice and Bob must be able to verify the origin of Diffie–Hellman public values and other signed messages.
- MITM should be unable to impersonate either party after certificates and signatures are introduced.

7. Non-Repudiation

- Once RSA signatures are introduced, Alice and Bob should not be able to deny having sent a signed message.
- The system must show how signatures provide this property.

8. Confidentiality

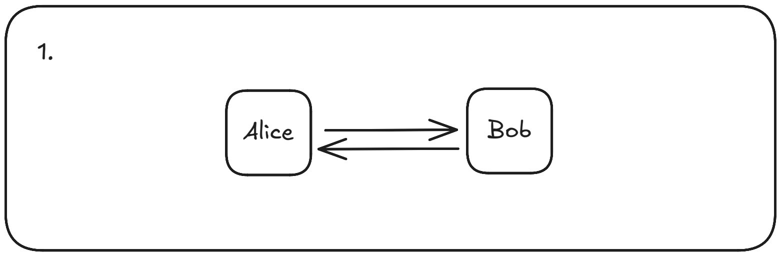
- Demonstrate that strong Diffie–Hellman establishes a shared secret unreadable to an eavesdropper.

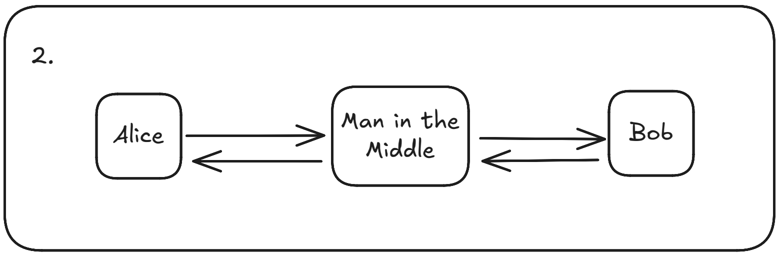
9. Animation Integration

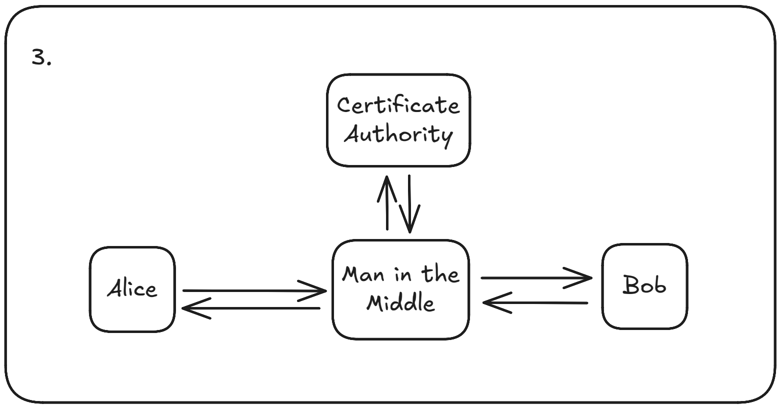
- Every networking event (send, receive, intercept) must trigger a callback to the animation engine.
- Animation should reflect the message path without interfering with the real HTTP logic.
- The manim library will be used for animation

Code Structure

The system will be structured into separate Flask servers for each actor, with a shared animation event emitter. The goal is to keep the networking fully real while allowing the animation engine to reflect all message activity.







1. Actors as Flask Servers

Each of the four actors (Alice, Bob, MITM, CA) runs as its own Flask server. Their responsibilities are:

- Alice and Bob:
  - Generate RSA keys.
  - Request certificates from the CA.
  - Perform Diffie–Hellman key exchange.
  - Send and receive messages over HTTP.
  - Sign and verify DH public values when signatures are introduced.
- MITM:
  - Relay all messages between Alice, Bob, and CA.
  - Log and optionally modify messages.
  - Demonstrate attacks at different stages.
- CA:
  - Provide certificates binding identities to RSA public keys.
  - Sign certificates with its private key.
  - Expose its public key to Alice and Bob.

2. Decorator-Based Animation Hooks

All networking operations that represent a communication event will be decorated with an annotation such as @animate. The decorator performs two tasks:

1. Sends an event notification to the animation engine with metadata (sender, recipient, payload type, etc.).
2. Executes the normal function logic, including the real HTTP request.