DAT 510: Assignment 2

Submission Deadline: 23:59, Friday, Oct. 11, 2024

Secure Communication

The objective of this assignment is to provide a comprehensive understanding of the Diffie-Hellman key exchange protocol and its application in secure communication, with a particular emphasis on its relevance to modern encryption protocols used in popular messaging apps like WhatsApp and Signal. Students will explore the theoretical foundations of the Diffie-Hellman algorithm (Chapter 10), including its mathematical principles and cryptographic significance. Additionally, the assignment aims to use shared secret from Diffie-Hellman in Hashed message authentication code (HMAC) (Chapter 12) to improve authentication of the message. Futhermore confidentiallity is improved through encryption from assignment 1.

Task I. Implementing Diffie-Hellman

After learning about cryptography, Alice and Bob decided to secure their communications using the Diffie-Hellman key exchange protocol. Below is their step-by-step procedure for establishing secure communications:

- Alice and Bob agreed to share public parameters p and g for the Diffie-Hellman protocol.
  - A large prime number p.
  - A base (or generator) g, which is a primitive root modulo p.
- Private keys
  - Alice chooses a private key a, which is a secret integer.
  - Bob chooses a private key b, which is also a secret integer.
- Public keys
  - Alice computes her public key as $A = g^a modp$.
  - Bob computes his public key as $B = g^b modp$.
  - Both public keys, A and B, are exchanged between Alice and Bob.
- Secret Key
  - Alice uses Bob's public key to compute the shared secret: $S = B^a modp$.
  - Bob uses Alice's public key to compute the shared secret: $S = A^b modp$.
  - Both calculations result in the same shared secret S because:

$$S = \left(g^b\right)^a \mod p = (g^a)^b \mod p \tag{1}$$

Task II. Implementing HMAC for Authentication

After establishing the shared secret through the Diffie-Hellman key exchange, Alice and Bob need to ensure that their communications remain not only confidential but also authentic. Without message authentication, a malicious third party could alter messages in transit, leading to potential attacks. To solve this problem, Alice and Bob will implement **HMAC** (Hashed Message Authentication Code) to verify the integrity and authenticity of the messages they exchange.

HMAC is a method that combines a cryptographic hash function and a secret key to produce a message authentication code. It ensures that the message hasn't been altered and verifies that it came from the expected sender. Alice and Bob will use the shared secret from Diffie-Hellman as the key input to the HMAC function.

- **HMAC Procedure**:
  - The input to the HMAC function will consist of a **message** to be authenticated and a **key** derived from the shared secret (using a key derivation function or directly from Diffie-Hellman).
  - The HMAC function will output an **authentication tag**, which Alice and Bob will use to verify the integrity and authenticity of their messages. The HMAC can be expressed mathematically as follows:

$$\text{HMAC}(K, m) = \text{hash}\left((K' \oplus \text{opad}) \parallel \text{hash}\left((K' \oplus \text{ipad}) \parallel m\right)\right) \tag{2}$$

    where:
    - K: The secret key (derived from the Diffie-Hellman shared secret). If K is longer than the block size of the hash function, it will first be hashed to create a shorter key K'. If K is shorter than the block size, it will be padded with zeros to match the block size.
    - m: The message to be authenticated.
    - hash: A secure cryptographic hash function (e.g., XOR).
    - K': A hashed or padded version of the secret key K that fits the block size of the hash function.
    - ipad: Inner padding (typically the byte 0x36 repeated to match the block size of the hash function).
    - opad: Outer padding (typically the byte 0x5c repeated to match the block size of the hash function).
    - $\parallel$: Concatenation operator.

    * $\oplus$: XOR (exclusive OR) operation.

- **Message Authentication Workflow**:
  - When Alice wants to send a message to Bob, she will compute the **HMAC** of the message using the shared key. This will produce an **authentication tag**.
  - Bob, upon receiving the message, will compute the HMAC of the received message using the same shared key. He will compare the computed tag with the one sent by Alice. If the tags match, the message is authenticated and verified as untampered.
  - Similarly, Bob will authenticate his messages using the same method when he sends a message to Alice.
- **Limitations of Using Simple XOR in Hashing**:
  - While XOR is a basic and fast operation, it lacks the cryptographic strength needed to ensure security in real-world applications. If you used XOR as the hash function in HMAC, it would be vulnerable to simple attacks, including key collisions and tampering.
  - For example, two different inputs can easily produce the same output using XOR, which makes it unsuitable for cryptographic hash functions.
  - Students should identify these weaknesses and propose a more secure hash function, such as **SHA-256** or **SHA-3**, which provide resistance to collisions, pre-image attacks, and second pre-image attacks.

Task III. Applying Encryption-Decryption from the Previous Assignment

Alice and Bob have already implemented Assignment 1 which results in finding the best encryption techniques in the given context. Alice and Bob can use the finding of Assignment 1 or better encryption they suggested in task 5 of Assignment 1 to encrypt the message. You can either encrypt only message or both message and hash given in Figure 1. The same could be followed in reverse for decryption.

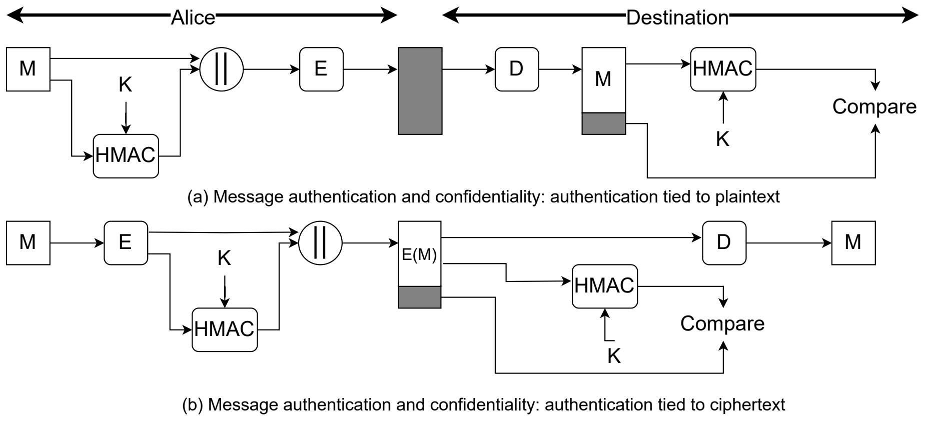
Alice
Destination
M
K
HMAC
||
E
D
M
HMAC
K
Compare
(a) Message authentication and confidentiality: authentication tied to plaintext
M
E
K
HMAC
||
E(M)
D
M
HMAC
K
Compare
(b) Message authentication and confidentiality: authentication tied to ciphertext

Figure 1: Encryption scheme: (a) HMAC followed by Encryption (b) Encryption followed by HMAC.

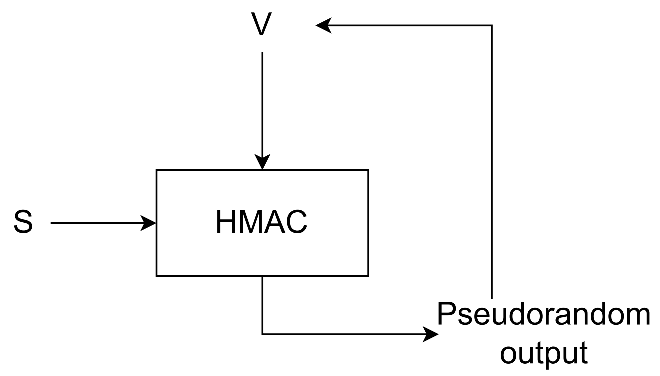

Figure 2: PRNG using HMAC for key derivation and authentication.

Task IV. Implementing Single Ratchet for Chain Key

In secure communication protocols, **forward secrecy** ensures that even if one session key is compromised, past and future communications remain secure. To achieve this, a **ratcheting mechanism** is used to evolve cryptographic keys over time. In this task, you will implement a **Single Ratchet** mechanism, which securely updates the **chain key** and generates new **message keys** for each message exchanged between Alice and Bob.

Single Ratchet Overview:

The Single Ratchet uses a **chain key** that updates after each message is sent or received. This key evolution ensures that each message uses a unique message key, thereby providing forward secrecy. The process works as follows:

1. Alice and Bob begin with an initial **chain key** derived from the shared Diffie-Hellman secret (same as Task 1 and 2).
2. Every time a message is sent or received, the chain key is updated using a pseudo-random function (such as HMAC in Chapter 12). Note that this is a **separate HMAC** from the one used for message authentication. This HMAC is responsible for evolving the chain key, ensuring that a fresh key is derived for every message, as shown in Figure 2, which is different from the HMAC used to generate the authentication tag for each message.
3. A new **message key** is derived from the chain key for encrypting and authenticating each message.

In this task, you will implement the Single Ratchet to ensure that every message exchanged between Alice and Bob uses a unique key.

- **Chain Key Ratcheting**:
  - Alice and Bob start with an initial **chain key** K_{chain}, which is derived from the shared secret or any pre-shared key.
  - Each time Alice or Bob sends or receives a message, they will apply an HMAC-based PRNG to the current chain key to derive a new chain key. The function can be expressed as:

$$K'_{chain} = \text{HMAC}(K_{chain}, \text{“ratchet”}) \tag{3}$$

    where K_{chain} is the current chain key, and “ratchet” is an optional label to differentiate the key derivation step.
  - The newly derived chain key K'_{chain} is then used to derive the message key for the current message.
- **Message Key Derivation**:
  - Once the new chain key is derived, the next step is to use the chain key to generate a **message key** that will be used for encrypting or authenticating the message.
  - The message key $K_{message}$ can be derived as:

$$K_{message} = \text{HMAC}(K_{chain}, \text{“message”}) \tag{4}$$

    where K_{chain} is the current chain key after ratcheting.
  - This message key will be unique for every message, ensuring that even if a message key is compromised, future communications remain secure.
- **Steps to Implement**:
  1. Initialize the chain key for both Alice and Bob (Task 1 and 2).
  2. For each message:
     - Ratchet the chain key using the HMAC function.
     - Derive the message key from the updated chain key.
     - Use the message key for encryption or authentication of the message.
  3. After sending or receiving each message, update the chain key for future messages.
- **Security Note**: The Single Ratchet mechanism ensures that even if a message key is compromised, future messages are still secure due to the continuous evolution of the chain key. However, it lacks **post-compromise security**, which is addressed in the next task through the **Double Ratchet** mechanism.
- **Task Objectives**:

  - Implement the Single Ratchet mechanism for the chain key.
  - Derive a new message key for each message using the ratcheted chain key.
  - Ensure that the chain key updates correctly after each message is sent or received.

Task V. Implementing Double Ratchet for Diffie-Hellman

In the previous task, you implemented the **Single Ratchet** mechanism, which provides forward secrecy by updating the chain key for each message. However, the Single Ratchet lacks **post-compromise security**. To address this, we introduce the **Double Ratchet Algorithm**, which combines the symmetric ratcheting of the chain key with periodic **Diffie-Hellman (DH) key exchanges**. This ensures that even if a key is compromised, future communications can remain secure.

Double Ratchet Overview:

The Double Ratchet mechanism works by periodically performing a Diffie-Hellman key exchange between Alice and Bob. After each key exchange, a new **root key** is derived, which in turn generates new **chain keys** for both sending and receiving messages. This periodic DH exchange provides **post-compromise security**, ensuring that future messages remain secure even if previous keys were compromised.

The Double Ratchet consists of two components:

1. The **Diffie-Hellman (DH) Ratchet**, where Alice and Bob exchange new DH public keys and use them to derive new root keys.
2. The **Symmetric Key Ratchet**, which updates the chain key for each message, ensures that each message uses a fresh key.

Steps to Implement:

- **Initial Setup**:
  - Alice and Bob each generate an initial **Diffie-Hellman key pair**$(\mathrm{DH}_{private}, \mathrm{DH}_{public})$ and exchange their public keys to compute the shared secret.
  - They use the Diffie-Hellman shared secret to derive an initial **root key**K_{root}, which will be used to generate the first set of chain keys.
- **Diffie-Hellman Ratcheting**:

  - After a certain number of messages (or at defined intervals), Alice and Bob will each generate a new Diffie-Hellman key pair and exchange their new DH public keys.
  - Using the new Diffie-Hellman shared secret, they derive a new **root key**K'_{root}. The new root key is then used to reset both the sending and receiving chain keys, providing fresh keys for future communication.
  - The root key can be updated using:

$$K'_{root} = \text{HMAC}(K_{root}, \text{DH shared secret}) \tag{5}$$

- **Symmetric Key Ratchet**(Same as in Task 4):
  - After each message is sent or received, the chain key is ratcheted forward using an HMAC-based PRNG, as explained in the Single Ratchet mechanism.
  - A new **message key**$K_{message}$ is derived from the ratcheted chain key for encrypting or authenticating the message:

$$K_{message} = \text{HMAC}(K_{chain}, \text{"message"}) \tag{6}$$

- **Post-Compromise Security**:
  - Even if a previous chain key or message key is compromised, future communications are protected by the periodic DH key exchanges. Each DH exchange results in a new root key, effectively severing the link to previous compromised keys.
  - The combination of the Diffie-Hellman ratcheting with the symmetric ratchet provides enhanced security by limiting the exposure of compromised keys to only a small window of messages.

Task Objectives:

- Implement the **Double Ratchet** mechanism by combining the Diffie-Hellman ratcheting with the symmetric chain key ratcheting.
- Ensure that Alice and Bob generate new DH key pairs and exchange them periodically to derive new root keys.
- Use the root keys to generate fresh chain keys for both sending and receiving.
- Ensure that each message uses a unique message key and that future messages remain secure even if a previous key is compromised.

Assignment Approval (by TA and SA)

Assignment approval will have a weight on your grade for the assignment. If you are not going to get the approval before the deadline, your assignment will not be evaluated and you will fail the assignment.

What needs to be done to get the approval for the assignment:

1. Show all parts of the assignment are working i.e. show the code with proper comments, and results.
2. Code should have a proper README file that describes the contents of the directory and any special instructions needed to run your programs (i.e. if it requires any packages, commands to install the package. Describe any command line arguments with the required parameters).
3. The Source code submitted for the assignment should be your own code. If you have used sources from the internet everything should be added to the references. If you used someone's code without reference, that will also be treated as plagiarism.
4. Provide the references in Code and Report, and show these parts for TAs and Student Assistants.
5. You **CAN** use available libraries/packages/classes for implementing the core functionality of the assignment.

You have to use Python as a programming language for this assignment.

Assignment Submission

Deadline: 23:59, Friday, Oct. 11, 2024 (submit your assignment through canvas)
Final submission:

1. Source Code
   - The Source code submitted for the assignment should be your own code. If you have used sources from the internet everything should be added to the references. If you used someone's code without reference, that will also be treated as plagiarism.
   - Source code should be a single, compressed directory in .tar.gz or .zip format.
   - Directory should contain a file called README that describes the contents of the directory and any special instructions needed to run your programs (i.e. if it requires packages, commands to install the package. describe any command-line arguments with the required parameters).
   - You should **NOT** use available libraries/packages/classes for implementing the core functionality of the assignment.
2. A **separate** report with PDF format
   - Texts in the report should be readable by humans, and recognizable by machines;
   - Other formats will **NOT** be opened, read, and will be considered missing;
   - Report should follow the formal report style guide on the next page.
   - Each student should write an individual report. Each report will be checked for plagiarism. If it is copied from somewhere else, you will fail the assignment.

NOTE: Please upload the archive file in *.zip, *.tar only and report in *.pdf format only to the website https://stavanger.instructure.com/.

Note: The assignment is individual and can **NOT** be solved in groups.

Project Title

Abstract

Write an abstract of your complete assignment. A single paragraph of about 200 words maximum.

1. Introduction

The introduction section should explain all the concepts used in this assignment for a better understanding of the background of the assignment.

2. Design and Implementation

A detailed description of your implementation part, which consists of all the concepts you have implemented and discusses the way of implementation.

3. Test Results

Results of testing the software, as you observed/recorded them. Note that this section is only for observations you make during testing. You can add screenshots in an organized way for a better understanding of the software. Your analysis belongs in the Discussion section.

4. Discussion

Discuss your point of view of the assignment results.

5. Conclusion

A paragraph that restates the objective from your introduction, relates it to your results and discussion and describes any future improvements you would recommend.

References

A bibliography of all of the sources you got information from in your report.