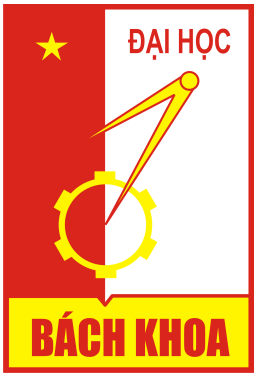

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SoICT

APPLIED CRYPTOGRAPHY
PROGRAMMING ASSIGNMENT II

Project 2 Short-answer Questions

TEACHER: Tran Vinh Duc

Submitted By:

1	Nguyen Minh Triet 20214975
2	Nguyen Hai Ninh 20214968
3	Hoang Son Tung 20214979

TABLE OF CONTENTS

1. DH Ratchet Update Frequency: Security with Less Frequent Updates

1.1. Problem

In our implementation, Alice and Bob increment their Diffie-Hellman ratchets every time they exchange messages. Could the protocol be modified to have them increment the DH ratchets once every ten messages without compromising confidentiality against an eavesdropper (i.e., semantic security)?

1.2. Answer

Even though the DH-ratchet updates are infrequent (every ten messages), the protocol can remain secure against eavesdroppers. Here's how: Alice and Bob keep track of message exchanges. Every ten messages, they reset the counter and generate a new DH-ratchet key pair to create fresh chain keys. Imagine an eavesdropper steals a chain key. They could only decrypt ten future messages at most. After that, the DH ratchet kicks in, throws away the old keys, and creates new ones. This cuts off the eavesdropper's access to future messages. Basically, thanks to the mathematical assumption behind DH keys, even with these infrequent updates, the system remains secure and protects the content of messages (CPA security).

2. No DH Key Updates: Forward Secrecy Loss vs. Break-in Recovery

2.1. Problem

What if they never update their DH keys at all? Please explain the security consequences of this change with regard to Forward Secrecy and Break-in Recovery

2.2. Answer

Without updating DH keys, the system is vulnerable if someone breaks in. However, past messages are still safe (forward secrecy). Here's why:

An attacker who hacks Alice's system could steal her current key for receiving messages from Bob. This key allows them to guess future keys using a special function (KDF). With these future keys, they could decrypt all messages Bob sends to Alice. The same applies if they steal the key for sending messages. The takeaway: a stolen key grants access to all future messages, which breaks **break-in recovery**.

The good news is that even if an attacker steals a current key, they cannot decrypt past messages. This is because the KDF is a one-way function – it's easy to create future keys from a current one, but impossible to reverse the process and access past keys. So, **forward secrecy** remains intact.

3. Double Ratchet Chain Length in Message Exchange

3.1. Problem

Consider the following conversation between Alice and Bob, protected via the Double Ratchet Algorithm according to the spec:
A: Hey Bob, can you send me the locker combo?
A: I need to get my laptop
B: Sure, it's 1234!
A: Great, thanks! I used it and deleted the previous message.
B: Did it work?
What is the length of the longest sending chain used by Alice? By Bob? Please explain.

3.2. Answer

Imagine a chain where each link is a key used to scramble messages. There are two chains, one for Alice sending messages and one for Bob.

- **Sending messages:** This chain is built using a special method (Diffie-Hellman) to create the first key. Then, all subsequent messages sent in a row use the same chain to create new keys.
- **Chain length:** Since Alice can only send up to two messages in a row, her chain is at most two keys long. Bob can only send one message at a time, so his chain is just one key long.

4. Double Ratchet Thwarts Mallory: Forward Secrecy in Action

4.1. Problem

Unfortunately, in the situation above, Mallory has been monitoring their communications and finally managed to compromise Alice's phone and steal all her keys just before she sent her third message. Mallory will be unable to determine the locker combination. State and describe the relevant security property and justify why double ratchet provides this property.

4.2. Answer

Even if Mallory intercepts messages right before Alice sends her third one, she's still out of luck in cracking the secret code (locker code). This is because of a security feature called **Forward Secrecy**. Here's how it works:

- **Forgetful keys:** The system throws away old keys used to scramble messages (previous sending chain keys and the receiving key for Bob's first message).
- **Fresh keys, new messages:** Every time Alice or Bob sends a message, a brand new key is created using a special method (DH-ratchet). It's like a back-and-forth game (ping-pong) where new keys are created with each message exchange.

Mallory might get the key for the third message, but without the older ones, she can't decrypt past messages. Forward Secrecy ensures that even if someone snoops on current communications, they can't unlock past conversations.

5. Flawed Government Surveillance: Ineffective and Risky

5.1. Problem

The method of government surveillance is deeply flawed. Why might it not be as effective as intended? What are the major risks involved with this method?

5.2. Answer

Government surveillance that relies on a constant public key has several weaknesses:

1. Broken Forward Secrecy: Unlike secure systems, this method doesn't update encryption keys regularly. If someone hacks the government's system and steals the key, they can decrypt ALL messages, not just future ones. This breaks a crucial security feature called "forward secrecy."

2. Dependence on User Compliance: The system depends on people following complex rules. Mistakes, like using the wrong key, can block the government from decrypting messages. There's also extra information required in each message header, and missing or incorrect information further hinders decryption.

3. User Workarounds and Reduced Effectiveness: Users can make it harder for the government to track them by using fake usernames, VPNs, or proxy servers. While this frustrates government efforts, it also prevents them from decrypting communication in the first place.

6. ECDSA vs. RSA Signature Performance in SubtleCrypto

6.1. Problem

The SubtleCrypto library is able to generate signatures in various ways, including both ECDSA and RSA keys. For both the ECDSA and RSA-based signature techniques, please compare:
(a) Which keys take longer to generate (timing SubtleCrypto.generateKey)
(b) Which signature takes longer to generate (timing SubtleCrypto.sign)
(c) Which signature is longer in length (length of output of SubtleCrypto.sign)
(d) Which signature takes longer to verify (timing SubtleCrypto.verify)

6.2. Answer

Running the script reveals some key differences between ECDSA and RSA for digital signatures:
(a) RSA key takes longer to generate than ECDSA key.
(b) RSA signature takes longer to generate than ECDSA signature.
(c) RSA signature is longer in length than ECDSA signature.
(d) ECDSA signature takes longer to verify than RSA signature.