# Question 1

The methods (ii): $y=H(s) \oplus H(\text{password})$ and (iii) : $y= E_s( H(\text{password}) )$ are

effectively secured by the salt against precomputed dictionary attacks because they incorporate the salt in a way that a dictionary attack would require knowledge of the salt, which is assumed to be adequately long and random. Method i is potentially less secure due to the XOR with the password, and method iv's security depends on the implementation details of the MAC and the use of the salt.

Now, let's make some explanations and break down the security of each method against precomputed dictionary attacks:

1. For i: The hash H is computed using both the salt and the password, then XORed with the password. This method might not be secure because the final step of XORing with the password can potentially undo some of the security provided by the use of a salt if the password is weak.
2. For ii: This method is more secure against precomputed dictionary attacks because it uses the salt to modify the hash function itself. The attacker cannot precompute the hash of common passwords because they would need the specific salt for each password.
3. For iii: AES encryption of the hash of the password with the salt as the key is secure as long as the key (salt in this case) is kept secret. This method does not allow for precomputed hashes because the salt is not known in advance.
4. For iv: The use of a MAC where the key is the password and the message is the salt is potentially secure, but it depends on how the MAC is constructed and used. If the salt changes for each password, then it is secure.

# Question 2

(a) By December 2024, there are 5 cipher suites specified in TLS 1.3. These cipher suites are:

1. TLS_AES_128_GCM_SHA256
2. TLS_AES_256_GCM_SHA384
3. TLS_CHACHA20_POLY1305_SHA256
4. TLS_AES_128_CCM_8_SHA256
5. TLS_AES_128_CCM_SHA256

(b) **Encryption using TLS_AES_128_CCM_SHA256**:

- TLS_AES_128_CCM_SHA256 is a cryptographic method used to secure data transmission.
- It employs symmetric-key encryption, meaning the same key is used for both encryption and decryption.
- Data, once encrypted, appears as an unintelligible jumble of ciphertext.
- When an appropriate algorithm is used, attackers cannot access the original data.
- This ensures confidentiality and data security during transmission.

**Session Key Generation**:

- A session key is generated to secure communication between users or computers.
- It is derived from a hash value using a function like CryptDeriveKey, known as a session-key derivation scheme.
- The session key is a randomly generated encryption and decryption key.
- Its purpose is to safeguard the security of a communication session.

Now, I’d like to make some more explanation concerning Data Message Encryption and Session Key Generation:

Encryption - TLS_AES_128_CCM_SHA256 provided confidentiality to the data that it transmits and uses symmetric-key encryption . Just one key is used in both the encryption and decryption processes. Once data has been encrypted with an algorithm, it will appear as a jumble of ciphertext.

As long as an appropriate algorithm is used, attackers will not be able to access the actual data.

**Encryption - TLS_AES_128_CCM_SHA256** provided confidentiality to the data that it transmits and uses symmetric-key encryption . Just one key is used in both the encryption and decryption processes. Once data has been encrypted with an algorithm, it will appear as a jumble of ciphertext.

As long as an appropriate algorithm is used, attackers will not be able to access the actual data .

**Data Message Encryption**:

**Symmetric-Key Encryption**: TLS_AES_128_CCM_SHA256 is an AEAD (Authenticated Encryption with Associated Data) cipher suite that uses symmetric-key encryption. It means the same key is used for both encryption and decryption.

**AES-128 CCM Mode**: For encryption, it uses the AES-128 algorithm in CCM (Counter with CBC-MAC) mode. CCM mode provides both confidentiality (encryption) and integrity (authentication) of the data. The data is encrypted using AES-128, and a MAC (Message Authentication Code) is generated to ensure data integrity.

**The session key** is a temporary key that is unique to each session and is used to protect the confidentiality and integrity of data exchanged during that session. It is generated securely during the handshake process to ensure the security of the communication session.

**Session Key Generation**:

During the TLS handshake process, the two communicating parties agree on a shared secret. This shared secret is typically generated using a key exchange mechanism such as Elliptic Curve Diffie-Hellman (ECDH) or RSA, depending on the chosen key exchange method.

Once the shared secret is established, it serves as the basis for deriving session keys.

A key derivation function (KDF) is used to derive various session keys from the shared secret. These keys include encryption keys, MAC keys, and possibly other keys needed for secure communication.

The derived session keys are used for encryption and authentication during the session. In this case, they are used for AES-128 encryption and HMAC-SHA-256 authentication in the TLS_AES_128_CCM_SHA256 cipher suite.

## Question 3

(a) To find out who issued the certificate for https://mail.google.com and how long the certificate will be valid, you can check the certificate information in your browser. In Google Chrome, you can click on the padlock icon in the address bar and then click "Certificate".

This will open a window displaying the details of the certificate. According to this information, the certificate for https://mail.google.com is issued by Google Trust Services, and it is valid until January 13, 2025.

(b) The number of certificates contained in a browser can vary depending on the browser and its version. However, most modern browsers come with a pre-installed list of trusted certificate authorities (CAs) that are used to validate SSL/TLS certificates presented by websites. All major browsers have a pre-installed list of all major CAs public keys.

Therefore, the number of certificates contained in a browser can be quite large.

(c) The significance of a CA certificate being contained in the browser is that it allows the browser to verify the authenticity of SSL/TLS certificates presented by websites. When a user visits a secure website, the website presents its SSL/TLS certificate to the user's browser. The browser then checks the certificate against its list of trusted CAs to ensure that it was issued by a trusted party. If the certificate is valid and issued by a trusted CA, the browser establishes a secure connection with the website.

Therefore, the inclusion of CA certificates in the browser is critical to ensuring the security of HTTPS connections.

(d) Secure / Multipurpose Internet Mail Extensions(SMIME) certificates are used for signing and encrypting emails. In these certificates the following two identifiers are used to identify the certificate:

1. An identifier of the message digest algorithm.

2. An identifier of the algorithm used to encrypt the message digest.

A message digest is nothing but a cryptographic hash function containing a string of digits created by a one-way hashing formula.

## Question 4

**Zoombombing** ---- It is a type of cyberattack which describes when someone hijacks a Zoom teleconferencing chat. It is the unwanted intrusion into a video conference call by an individual, which causes disruption. Reports of Zoombombing have flooded in from around the country. It has caused a number of problems for schools and educators, because unwanted participants posted lewd content. Some schools had to suspend using video conferencing altogether. The problem reached such prominence that the United States Federal Bureau of Investigation (FBI) warned of video-teleconferencing and online classroom hijacking, which it called "Zoom-bombing. The organization has advised users of teleconferencing software to keep meetings private and to require the use of passwords or other forms of access control such as "waiting rooms" to limit access only to specific people. They also suggest limiting screen-sharing access to the host of the online meeting only.

Technical Vulnerability that is allowing this to happen:-

1. Most Importantly, Zoom users should not share meeting links publicly. This is perhaps the single most obvious precautions you can take.
2. Second, set your meetings to "private". Zoom now sets all new meetings to "private" by default, requiring attendees to provide a password for access.
3. To avoid the risk of Zoom Bombing, share your personal meeting ID only with your most trusted contacts.
4. Restrict video sharing. If the meeting host is the only person who needs to share video, such as in a seminar or presentation, the host should change Zoom's screen-sharing setting to "Host only.

## Question 5

### (a) Key Exchange Protocol Analysis

#### i) Authentication and Secure Key Agreement

1. **Authentication**:
    - The protocol uses public key encryption to send identities and nonces. The use of {“Alice”} Bob and {“Bob”} Alice ensures that only the intended recipient can decrypt and verify the sender's identity.
    - The proofs proof A and proof B are based on the shared secret gab mod p, which is only computable by A and B if they know each other's public values and their own private values. This provides mutual authentication.
2. **Key Agreement**:
    - The session key K=h($g^{ab}$ mod p) is derived from the shared secret $g^{ab}$ mod p. This ensures that both parties have the same session key.
    - Key Control: Neither party has full control over the session key since it is derived from both a and b.
3. **Key Authentication**:
    - The use of proof A and proof B ensures that both parties can verify that the other party has computed the same shared secret, thus authenticating the key.

#### ii) Protocol Modification

To eliminate the nonces RA and RB while maintaining mutual authentication, we can modify the protocol as follows:

1. A → B: $g^a$ mod p, {“Alice”} Bob
2. A ← B: $g^b$ mod p, {“Bob”} Alice, proof B
3. A → B: proof A

In this modification:

- The nonces RA and RB are removed.
- The proofs proof A and proof B still ensure mutual authentication based on the shared secret $g^{ab}$ mod p.

## (b) Insecurity of Simplified IKE Phase

The simplified protocol is insecure because it allows an attacker to impersonate one of the participants. Here's how an attack could occur:

1. **Assume an attacker C wants to impersonate A to B**:
    - C intercepts the message from A to B: “Alice”, “Bob”, $g^a$ mod p.
    - C sends to B: “Alice”, “Bob”, $g^c$ mod p (where c is C's private key).
2. **B responds to C (thinking it's A)**:
    - B sends: “Bob”, “Alice”, $g^b$ mod p, [$g^c$ mod p]B.
3. **C intercepts and sends to A**:
    - C sends: “Bob”, “Alice”, $g^b$ mod p, [$g^a$ mod p]C.
4. **A responds to C (thinking it's B)**:
    - A sends: “Alice”, “Bob”, [$g^b$ mod p, $g^a$ mod p]A.

In this scenario, C successfully establishes a session key with B using $g^{bc}$ mod p and with A using $g^{ac}$ mod p, while B and A believe they are communicating with each other. The lack of proper mutual authentication and verification of signatures allows this attack.

## Question 6

In this scenario, two branches of a corporate network are connected through the Internet. Since the Internet is untrusted and the goal is to secure communication between the two routers at each branch while encapsulating the entire original IP packet, **tunnel mode** is the most appropriate choice. It ensures that not only the data payload but also the original IP header is encrypted and protected.

Here are the detailed steps of how a packet from node 10.1.1.5 in branch 1 travels to node 10.2.1.6 in branch 2 using IPsec in tunnel mode:

STEP 1 Packet Creation: Node 10.1.1.5 in branch 1 creates a packet destined for 10.2.1.6 in branch 2. This packet includes the data to be sent and the original IP header.

STEP 2 Routing: The packet is sent to the local router in branch 1, which is the entry point to the corporate network.

STEP 3 Encapsulation: The local router in branch 1, before sending the packet over the Internet, encapsulates the entire original packet (including the original IP header) inside a new IP packet. The new packet has a different outer IP header that is used for routing the packet over the Internet. This encapsulated packet is what will be secured using IPsec.

STEP 4 IPsec Processing: The router applies IPsec with ESP in tunnel mode to the encapsulated packet. This means the entire original packet, including its original IP header, is encrypted and authenticated.

STEP 5 Routing over the Internet: The encrypted packet is then sent over the Internet towards branch 2.

STEP 6 Arrival at Branch 2 Router: The packet arrives at the router in branch 2,which is the gateway for branch 2's network.

STEP 7 Decapsulation: The branch 2 router receives the encrypted packet, decrypts and verifies it using IPsec, and then extracts the original packet (including the original IP header) from the ESP-encapsulated payload.

STEP 8 Routing to Node 10.2.1.6: The branch 2 router forwards the original packet (with its original IP header intact) to node 10.2.1.6 in branch 2's network.

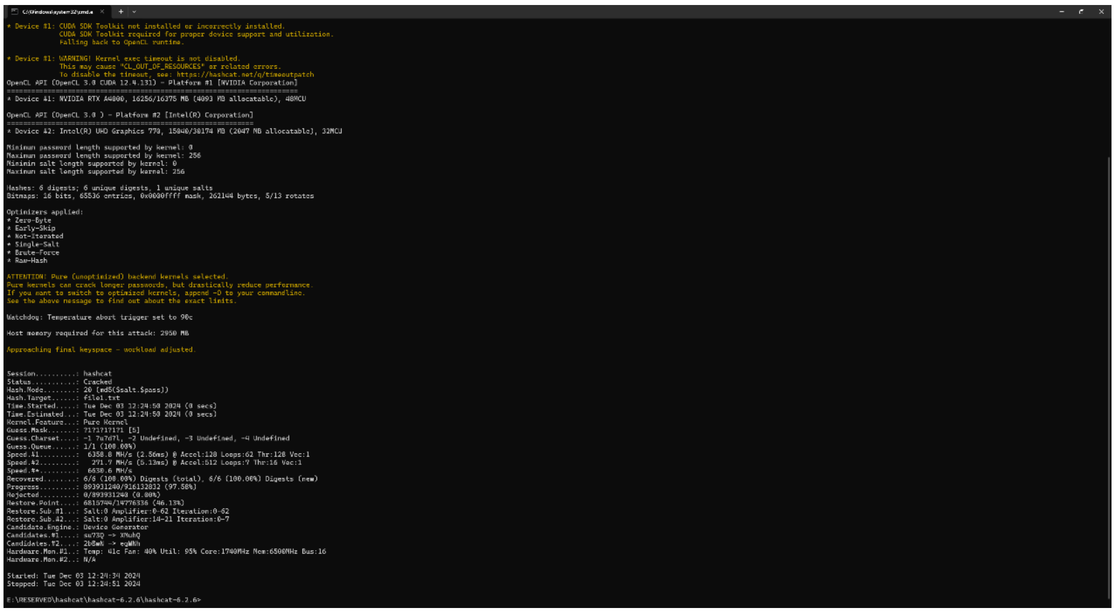
STEP 9 Delivery to Destination: Node 10.2.1.6 receives the packet from the branch 2 router and processes it as a normal incoming packet.

In summary, in tunnel mode, IPsec ensures that the original packet, including its original IP header, is encrypted and protected while it traverses the untrusted Internet, and it is only decrypted and delivered to its destination within the corporate network. This way, secure communication between the two branches is achieved.

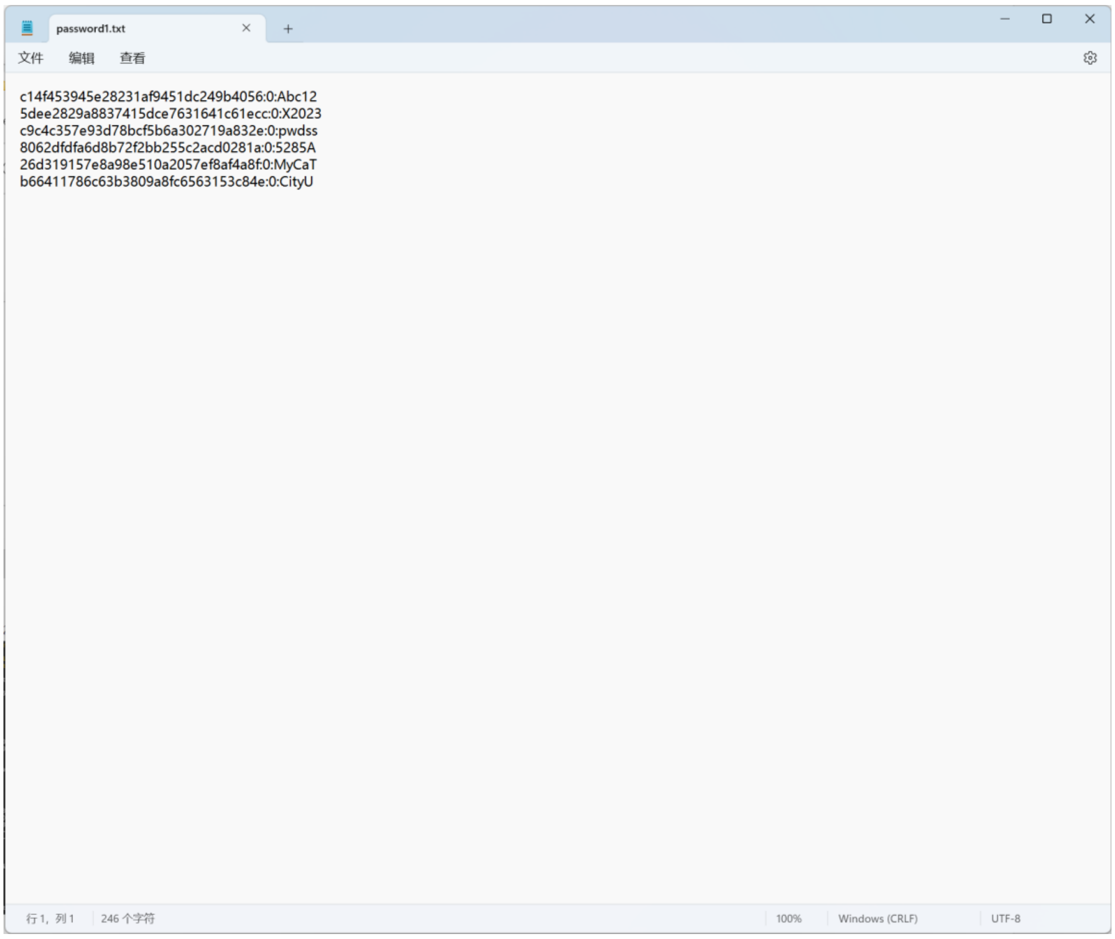
## Question 7

(a) The screenshots and results and shown as follows:

File 1 :



```
* Device #1: CUDA SDK Toolkit not installed or incorrectly installed.
             CUDA SDK Toolkit required for proper device support and utilization.
             Falling back to OpenCL runtime.

* Device #1: WARNING! Kernel exec timeout is not disabled.
             This may cause "CL_OUT_OF_RESOURCES" or related errors.
             To disable the timeout, see: https://hashcat.net/q/timeoutpatch
OpenCL API (OpenCL 3.0 CUDA 12.4.131) - Platform #1 [NVIDIA Corporation]
========================================================================
* Device #1: NVIDIA RTX A4000, 16256/16375 MB (4093 MB allocatable), 48MCU

OpenCL API (OpenCL 3.0 ) - Platform #2 [Intel(R) Corporation]
=============================================================
* Device #2: Intel(R) UHD Graphics 770, 15840/30174 MB (2047 MB allocatable), 32MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256
Minimin salt length supported by kernel: 0
Maximum salt length supported by kernel: 256

Hashes: 6 digests; 6 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates

Optimizers applied:
* Zero-Byte
* Early-Skip
* Not-Iterated
* Single-Salt
* Brute-Force
* Raw-Hash

ATTENTION! Pure (unoptimized) backend kernels selected.
Pure kernels can crack longer passwords, but drastically reduce performance.
If you want to switch to optimized kernels, append -O to your commandline.
See the above message to find out about the exact limits.

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 2950 MB

Approaching final keyspace - workload adjusted.


Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 20 (md5($salt.$pass))
Hash.Target......: file1.txt
Time.Started.....: Tue Dec 03 12:24:50 2024 (0 secs)
Time.Estimated...: Tue Dec 03 12:24:50 2024 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Mask.......: ?1?1?1?1?1 [5]
Guess.Charset....: -1 ?u?d?l, -2 Undefined, -3 Undefined, -4 Undefined
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:  6358.8 MH/s (2.56ms) @ Accel:128 Loops:62 Thr:128 Vec:1
Speed.#2.........:   271.7 MH/s (5.13ms) @ Accel:512 Loops:7 Thr:16 Vec:1
Speed.#*.........:  6630.6 MH/s
Recovered........: 6/6 (100.00%) Digests (total), 6/6 (100.00%) Digests (new)
Progress.........: 893931240/916132832 (97.58%)
Rejected.........: 0/893931240 (0.00%)
Restore.Point....: 6815744/14776336 (46.13%)
Restore.Sub.#1...: Salt:0 Amplifier:0-62 Iteration:0-62
Restore.Sub.#2...: Salt:0 Amplifier:14-21 Iteration:0-7
Candidate.Engine.: Device Generator
Candidates.#1....: su73Q -> XMuhQ
Candidates.#2....: 2b8wN -> egWNh
Hardware.Mon.#1..: Temp: 41c Fan: 40% Util: 95% Core:1740MHz Mem:6500MHz Bus:16
Hardware.Mon.#2..: N/A

Started: Tue Dec 03 12:24:34 2024
Stopped: Tue Dec 03 12:24:51 2024

E:\RESERVED\hashcat\hashcat-6.2.6\hashcat-6.2.6>
```



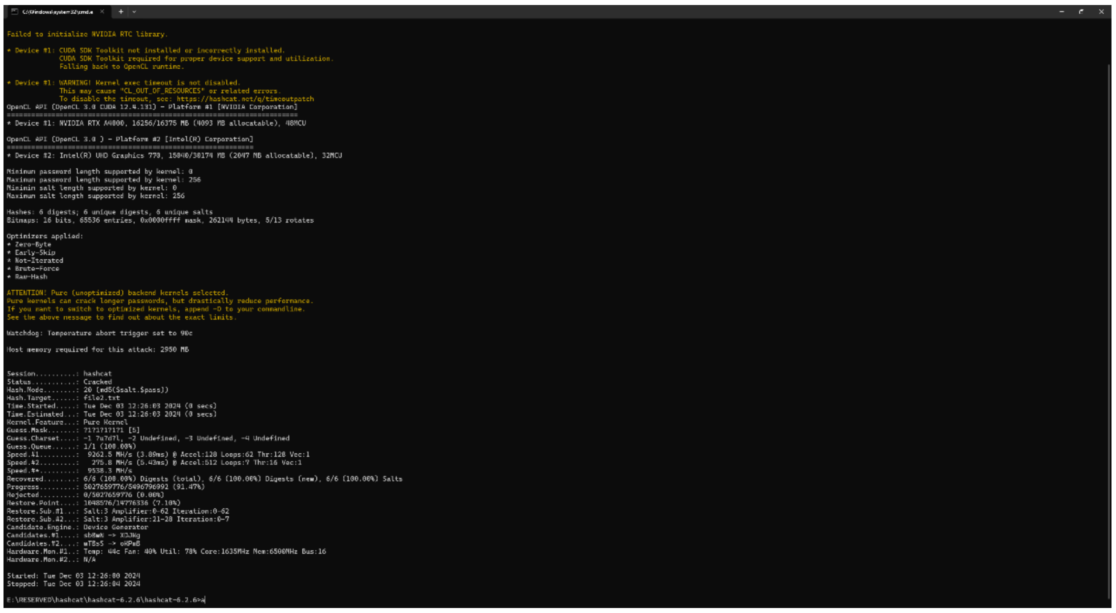
password1.txt

文件 编辑 查看

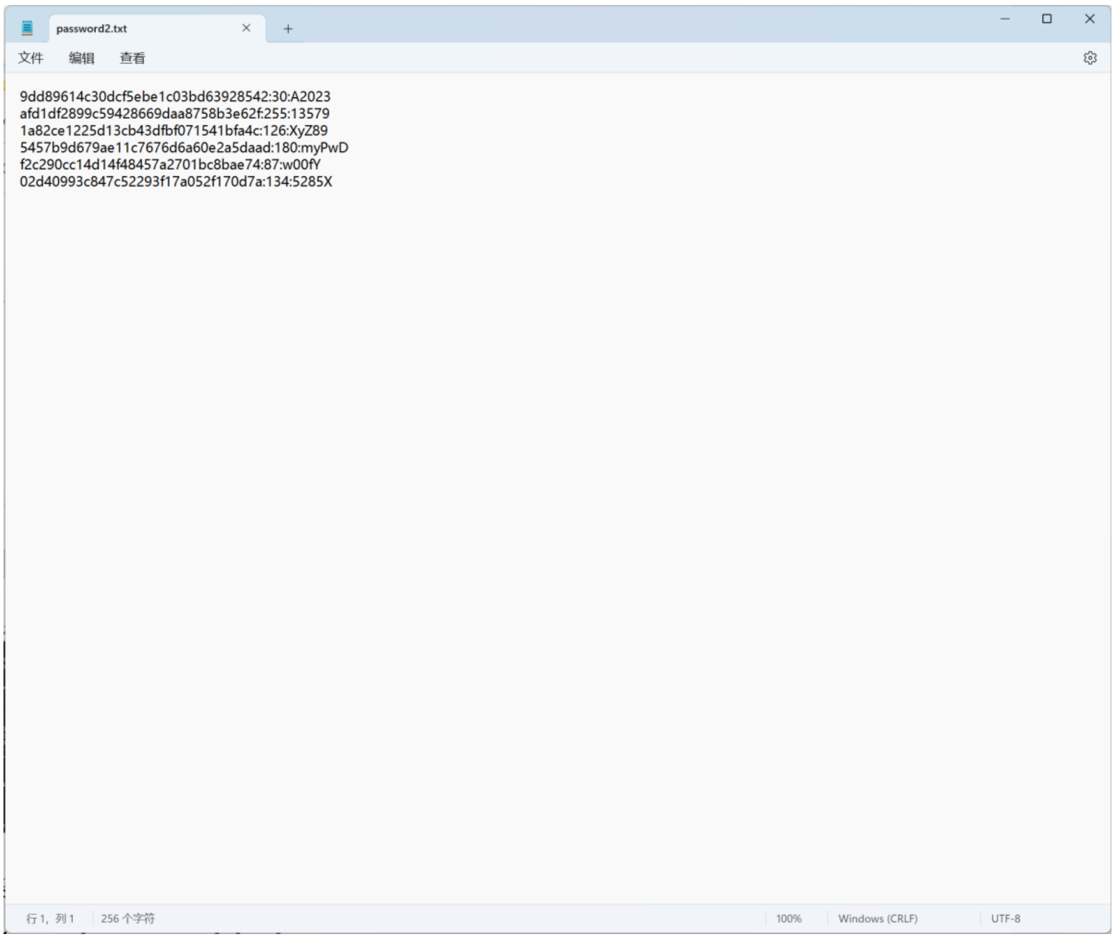
```
c14f453945e28231af9451dc249b4056:0:Abc12
5dee2829a8837415dce7631641c61ecc:0:X2023
c9c4c357e93d78bcf5b6a302719a832e:0:pwdss
8062dfdfa6d8b72f2bb255c2acd0281a:0:5285A
26d319157e8a98e510a2057ef8af4a8f:0:MyCaT
b66411786c63b3809a8fc6563153c84e:0:CityU
```

行 1，列 1 | 246 个字符 | 100% | Windows (CRLF) | UTF-8

File 2:



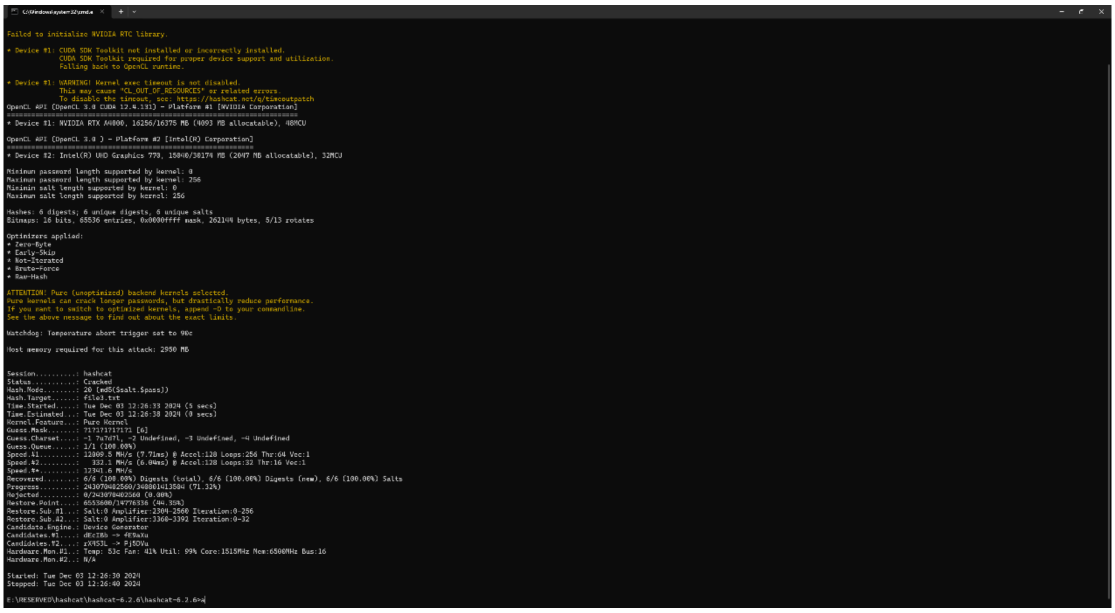
```
Failed to initialize NVIDIA RTC library.

* Device #1: CUDA SDK Toolkit not installed or incorrectly installed.
             CUDA SDK Toolkit required for proper device support and utilization.
             Falling back to OpenCL runtime.

* Device #1: WARNING! Kernel exec timeout is not disabled.
             This may cause "CL_OUT_OF_RESOURCES" or related errors.
             To disable the timeout, see: https://hashcat.net/q/timeoutpatch
OpenCL API (OpenCL 3.0 CUDA 12.4.131) - Platform #1 [NVIDIA Corporation]
========================================================================
* Device #1: NVIDIA RTX A4000, 16256/16375 MB (4093 MB allocatable), 48MCU

OpenCL API (OpenCL 3.0 ) - Platform #2 [Intel(R) Corporation]
=============================================================
* Device #2: Intel(R) UHD Graphics 770, 15040/30174 MB (2047 MB allocatable), 32MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256
Minimin salt length supported by kernel: 0
Maximum salt length supported by kernel: 256

Hashes: 6 digests; 6 unique digests, 6 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates

Optimizers applied:
* Zero-Byte
* Early-Skip
* Not-Iterated
* Brute-Force
* Raw-Hash

ATTENTION! Pure (unoptimized) backend kernels selected.
Pure kernels can crack longer passwords, but drastically reduce performance.
If you want to switch to optimized kernels, append -O to your commandline.
See the above message to find out about the exact limits.

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 2950 MB


Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 20 (md5($salt.$pass))
Hash.Target......: file2.txt
Time.Started.....: Tue Dec 03 12:26:03 2024 (0 secs)
Time.Estimated...: Tue Dec 03 12:26:03 2024 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Mask.......: ?1?1?1?1?1 [5]
Guess.Charset....: -1 ?u?d?l, -2 Undefined, -3 Undefined, -4 Undefined
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:  9262.5 MH/s (3.89ms) @ Accel:128 Loops:62 Thr:128 Vec:1
Speed.#2.........:   275.8 MH/s (5.43ms) @ Accel:512 Loops:7 Thr:16 Vec:1
Speed.#*.........:  9538.3 MH/s
Recovered........: 6/6 (100.00%) Digests (total), 6/6 (100.00%) Digests (new), 6/6 (100.00%) Salts
Progress.........: 5027659776/5496796992 (91.47%)
Rejected.........: 0/5027659776 (0.00%)
Restore.Point....: 1048576/14776336 (7.10%)
Restore.Sub.#1...: Salt:3 Amplifier:0-62 Iteration:0-62
Restore.Sub.#2...: Salt:3 Amplifier:21-28 Iteration:0-7
Candidate.Engine.: Device Generator
Candidates.#1....: sb8wN -> XDJKg
Candidates.#2....: wTBs5 -> oKPmB
Hardware.Mon.#1..: Temp: 44c Fan: 40% Util: 78% Core:1635MHz Mem:6500MHz Bus:16
Hardware.Mon.#2..: N/A

Started: Tue Dec 03 12:26:00 2024
Stopped: Tue Dec 03 12:26:04 2024

E:\RESERVED\hashcat\hashcat-6.2.6\hashcat-6.2.6>a
```



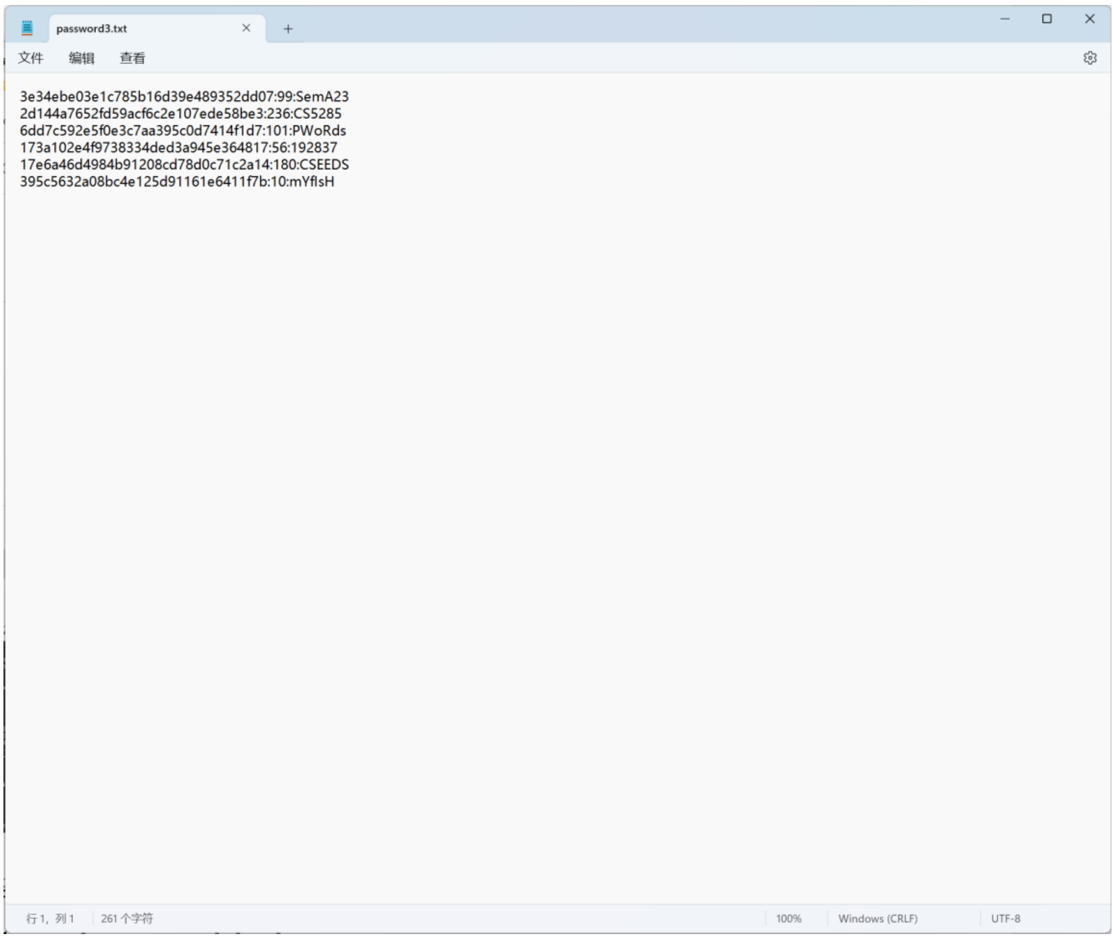
```
9dd89614c30dcf5ebe1c03bd63928542:30:A2023
afd1df2899c59428669daa8758b3e62f:255:13579
1a82ce1225d13cb43dfbf071541bfa4c:126:XyZ89
5457b9d679ae11c7676d6a60e2a5daad:180:myPwD
f2c290cc14d14f48457a2701bc8bae74:87:w00fY
02d40993c847c52293f17a052f170d7a:134:5285X
```

(b) The screenshots and results and shown as follows:

File 3:



```
Failed to initialize NVIDIA RTC library.

* Device #1: CUDA SDK Toolkit not installed or incorrectly installed.
             CUDA SDK Toolkit required for proper device support and utilization.
             Falling back to OpenCL runtime.

* Device #1: WARNING! Kernel exec timeout is not disabled.
             This may cause "CL_OUT_OF_RESOURCES" or related errors.
             To disable the timeout, see: https://hashcat.net/q/timeoutpatch
OpenCL API (OpenCL 3.0 CUDA 12.4.131) - Platform #1 [NVIDIA Corporation]
========================================================================
* Device #1: NVIDIA RTX A4000, 16256/16375 MB (4093 MB allocatable), 48MCU

OpenCL API (OpenCL 3.0 ) - Platform #2 [Intel(R) Corporation]
=============================================================
* Device #2: Intel(R) UHD Graphics 770, 15840/30174 MB (2047 MB allocatable), 32MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256
Minimin salt length supported by kernel: 0
Maximum salt length supported by kernel: 256

Hashes: 6 digests; 6 unique digests, 6 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates

Optimizers applied:
* Zero-Byte
* Early-Skip
* Not-Iterated
* Brute-Force
* Raw-Hash

ATTENTION! Pure (unoptimized) backend kernels selected.
Pure kernels can crack longer passwords, but drastically reduce performance.
If you want to switch to optimized kernels, append -O to your commandline.
See the above message to find out about the exact limits.

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 2950 MB


Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 20 (md5($salt.$pass))
Hash.Target......: file3.txt
Time.Started.....: Tue Dec 03 12:26:33 2024 (5 secs)
Time.Estimated...: Tue Dec 03 12:26:38 2024 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Mask.......: ?1?1?1?1?1?1 [6]
Guess.Charset....: -1 ?u?d?l, -2 Undefined, -3 Undefined, -4 Undefined
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........: 12009.5 MH/s (7.71ms) @ Accel:128 Loops:256 Thr:64 Vec:1
Speed.#2.........:   332.1 MH/s (6.04ms) @ Accel:128 Loops:32 Thr:16 Vec:1
Speed.#*.........: 12341.6 MH/s
Recovered........: 6/6 (100.00%) Digests (total), 6/6 (100.00%) Digests (new), 6/6 (100.00%) Salts
Progress.........: 243070402560/340801413504 (71.32%)
Rejected.........: 0/243070402560 (0.00%)
Restore.Point....: 6553600/14776336 (44.35%)
Restore.Sub.#1...: Salt:0 Amplifier:2304-2560 Iteration:0-256
Restore.Sub.#2...: Salt:0 Amplifier:3360-3392 Iteration:0-32
Candidate.Engine.: Device Generator
Candidates.#1....: dEcIBb -> fE9aXu
Candidates.#2....: rX4S3L -> Pj5DVu
Hardware.Mon.#1..: Temp: 53c Fan: 41% Util: 99% Core:1515MHz Mem:6500MHz Bus:16
Hardware.Mon.#2..: N/A

Started: Tue Dec 03 12:26:30 2024
Stopped: Tue Dec 03 12:26:40 2024

E:\RESERVED\hashcat\hashcat-6.2.6\hashcat-6.2.6>a
```



```
3e34ebe03e1c785b16d39e489352dd07:99:SemA23
2d144a7652fd59acf6c2e107ede58be3:236:CS5285
6dd7c592e5f0e3c7aa395c0d7414f1d7:101:PWoRds
173a102e4f9738334ded3a945e364817:56:192837
17e6a46d4984b91208cd78d0c71c2a14:180:CSEEDS
395c5632a08bc4e125d91161e6411f7b:10:mYflsH
```

(c) According to the results we get, it takes 2 seconds more to crack file 2 than file 1.

The answer is :Yes, it did. The results supported the theory.

(d) It takes 6 seconds more to crack file 3.

## Question 8

I have sent one test email to the shown mailbox from my private mailbox.