Week.13.txt

- April 5th, 2021
    - Digital Signature Equals Signed Message Digest
        - Public key crypto is utilized to create/generate digital signatures
        - The sender, also known as signer, can encrypt the message with his own secret key, and then send the signature of the plaintext along with a message
        - At the receiver's side, the recepient can compute the signature utilizing the sender/signer's public key
            - The recepient compares the newly/locally generated signature with the signature sent by the sender
                - However, this approach is expensive, because public key cryptos are more complex compute, compared to shared-key cryptos
        - Since public key cryptos are computationally expensive, digital signatures are computed by first computing a (message) digest of the original message or plaintext. This is done via hash function.
            - Next, the (message) digest is encrypted via the sender's secret/private key. Finally, the encrypted (message) digest, also known as the digital signature, is sent along with the original message/plaintext
                - This is done instead of instead of encrpyting the entire message and generating a digital signature
        - When the recepient receives the message, he can compute the hash of the original message/plaintext, decrypt the encrypted message digest, also referred to as digital signature, and then compare whether the locally generated hash matches the one in the decrypted message
        - A real life application of digital signatures is when you connect to an institution's network (i.e. McMaster University). During the connection phase, you are asked to accept a certificate from McMaster's Wi-Fi network
            - Opening the certificate reveals several fields. In particular, one field contains information about the certificate signing authority.
        - In RSA-2048, the size of the key is 2048
            - Similarly, in RSA-4096, the key is 4096
        - i.e. Diagram of Digitally Signing Messages

```
          |---------|     |----------|
          | Large   |     | H: Hash  |      |------|
          | message |---->| function |----> | H(m) |
          | `m`     |     |----------|      |------|
          |---------|                          |
               |                               |
               |                               |
               |                               V
               |                         |-----------|
               |                          | Digital   |
               |       ((K_B)-): -------->| Signature |
               |      Bob's Private Key   | (Encrypt) |
               |                          |-----------|
               |                                |
               |                                |
               |                                |
               |                                V
               |                      |------------------|
               V                      | Encrypted        |
             /---\                    | Message Digest   |
           + | A | <----------------- | (((K_B)-)(H(m))) |
             \---/                    |------------------|
          - In this example, Bob sends a digitally signed message to
            Alice
              - He signs the message using his private/secret key
              - The signed message is the encrypted hash of the
                plaintext message
      - i.e. Diagram of Verifying The Integrity of Digitally Signed
             Signatures
                                   |------------------|
             /---\                 | Encrypted        |
             | A |---------------> | Message Digest   |
             \---/                 | [((K_B)-)(H(m))] |
               |                   |------------------|
               |                            |
               |                            |
               V                            |
          |---------|                       |
          | Large   |                       |
          | Message |                       |
          | `m`     |                       |
          |---------|                       V
               |                      |-----------|
               |                      | Digital   |
               |                      | Signature |<---- ((K_B)+):
               V                      | Decrypt   |      Bob's
          |----------|                |-----------|      public key
          | H: Hash  |                      |
          | function |                      |
          |----------|                      |
               |                            |
               |                            |
               |                            |
               V                            V
           |------|                     |------|
           | H(m) |                     | H(m) |
           |------|                     |------|
               |                            |
               |                            |
            |-------> Equal ??? <--------|
```

- In this example, Alice verifies the signature and integrity of the digitally signed message, that is sent by Bob
    - Alice verifies the integrity of the original message by decrypting the encrypted digital signature and comparing it against the hash of the original message/plaintext, that is locally computed

- Hash Function Algorithms
    - Hash functions are utilized to generate message digests
        - There are many different kinds/types of hash functions
    - MD5 use to be a widely used hash function
        - It computes a 128-bit message digest in a 4-step process
        - The specification for MD5 is available via RFC 1321
        - In 1996, a flaw was found in the design of MD5
            - It is easy for an attacker to intentionally generate a hash collision
            - MD5 is considered to be cryptographically broken and unsuitable for further use
                - However, MD5 is used in less security demanding situations. For example, MD5 can be used if a user wants to verify whether a newly downloaded application/software has been tampered with or not
        - Compared to other hash functions, MD5 is relatively cheaper to compute
    - Currently, hash functions like SHA-2 and SHA-3, with different number of bits in the message digest, are commonly utilized
        - The size of the message digest can be:
            - 224 bits
            - 256 bits
            - 384 bits
            - 512 bits
- Certification For Public Key
    - In symmetric key crypto, communicating parties such as Alice and Bob need to be able to establish the shared secret beforehand
        - There needs to be an algorithm that can accomplish this
    - There are 2 solutions to the symmetric key crypto problem:
        1. Use a trusted key distribution center (KDC) acting as an intermediary between entities
            - The KDC has shared secrets with Alice and Bob, respectively
        2. Use Diffie-Hellman to derive shared secrets without the involvement of a third party or central authority
    - Public key crypto suffers from similar problem to symmetric key crypto
        - Utilizing public/private keys does not guarantee the identity of the communicating party

            - For example: If Alice obtains Bob's public key, how can Alice verify that this public key actually belongs to Bob, and not an attacker, like Trudy
            - For example: When connecting to McMaster's Wi-Fi, how can you tell that the public key in the certificate actually belongs to McMaster, and not an attacker with a rogue access point named "Mac-WiFi" that is pretending to be campus Wi-Fi
    - The ability to verify whether a public key actually belongs to an acclaimed person is non-trivial
        - In this situation, a third party that acts as a trusted certificate authority (CA) must be involved. The third party is able to verify whether a public key actually belongs to a certain individual or not
            - For example: When you first connect to McMaster's Wi-Fi, the certificate contains information of who issued, or assigned, it. For McMaster, the certificate issuer is Thawte Inc.
- Certification Authorities (1)
    - The certification authority (CA) will bind a particular public key to a particular entity
    - Before a certificate can be issued to a person or a server, they must provide proof of identity to the certificate authority (CA)
        - For example, if you want to have your own public registry and your own public key registered with a certificate authority (CA), then you may have to provide the certificate authority (CA) with your driver's licence and other information that can be used as proof of identity. Finally, the certificate authority (CA) will create the certificate that binds your entity with its public key
    - Certificates that are provided or signed by the certificate authority utilize digital signature
        - The underlying assumption is that the user already knows the certificate authority's public key, ahead of time.
            - However, the user does not trust, without proof, the public key of the remote party they are trying to communicate with, such as `wireless.mcmaster.ca` or an E-commerce site like Amazon
    - When a user tries to connect to a remote host, via a protocol such as HTTPS, the remote host send the user a certificate during the initial handshake process
        - Then, the user verifies the authenticity of the certificate with the help of a certification authority (CA)
    - i.e. Diagram of Certification Authentication Process

```
|----------|                |-------------|
| Bob's    |                | Bob's       |
| Public   |                | Identifying |
| Key      |                | Information |
| ((K_B)+) |                |-------------|
|----------|                       |
  |     |                          |
  |     |                          |
  |     |                          |
  |     |                          |
  |     |-------|         |--------|
  |             |         |
  |             |         |
  |             |         |
  |             V         V
  |           |-------------|
  |           | Certificate |
  |           | Authority   |
  |           | (CA)        |
  |           |-------------|
  |
  |
  |
  V
|---------|             |-----------|
|         |             | CA's      |
| Encrypt | <-----------| Private   |
|         |             | Key       |
|---------|             | ((K_CA)-) |
    |                   |-----------|
    |
    |
    |
    V
|-------------|
| Certificate |
| For Bob's   |
| Public Key, |
| Signed By   |
| Certificate |
| Authority   |
| (CA)        |
|-------------|
```

- This figure illustrates the entire process of assigning a public key to a particular entity or a person
- Initially, Bob will provide some identification information to the certificate authority (CA) as proof. Then, the certificate authority (CA) will take Bob's public key, which can be easily generated using some tools, and sign his public key with the certificate authority's private key
- The certificate contains Bob's public key as well as

    additional information such as the name of the certificate authority (CA), name and location of Bob, the digital signature that is generated utilizing the certificate authority's private key, etc.
    - A user connecting to Bob, or his services can utilize the pre-stored public key of the certificate authority (CA) to verify whether the certificate is actually signed by the certificate authority (CA). In turn, this also verifies the public key that is provided by Bob, and the user can be assured that the public key is associated with Bob and not a third party
  - To summarize:
    - The certification authority (CA) binds a particular public key to a particular entity
    - If an entity, like a person or a server, wants to register its public key with the certification authority (CA), then the entity needs to:
      - Provide "proof of identity" to CA)
      - CA creates certificate binding the entity to its public key
      - The certificate containing the entity's public key is digitally signed by the CA
        - The CA says, "This is the entity's public key"
- Certification Authorities (2)
  - If an entity, like Alice, wants Bob's public key, then she can request for Bob's certificate from the certification authority (CA), and then apply the certification authority's public key to the certificate
    - This can be part of the 4-way handshake when doing the SSL session
    - Once Alice has Bob's public key, she needs to determine whether she has the right key or not
      - Instead of verifying the public key directly, Alice can use a hash function to verify whether the hash value stored in the certificate is the same as the one she computes locally
  - The underlying assumption in this approach is that your browser has already stored or previously accepted the public key of the certification authority (CA)
    - A potential vulnerability in this approach is the certification authority (CA). Users need to trust their respective certification authority (CA)
      - If the certification authority (CA) gets compromised, then this entire mechanism will not work
        - Note: This has happened before
  - i.e. Diagram of Verifying Public Key

```
    |--------------|        |---------|          |----------|
    | Certificate  |        |         |          | Bob's    |
    | For Bob's    |------> | Decrypt |--------> | Public   |
    | Public Key,  |        |         |          | Key      |
    | Signed By CA |        |---------|          | ((K_B)+) |
    |--------------|           /\               |----------|
                                ||
                                ||
                                ||
                                ||
                         |---------------|
                         | Certification |
                         | Authority's   |
                         | Public Key    |
                         |---------------|
```

        - When Alice wants Bob's public key, she will:
            - Get Bob's certificate (from Bob or elsewhere)
            - Apply the certification authority's public key to Bob's certificate, and obtain Bob's public key
            - Use a hash function to see if the values are the same or not

- What Have We Learned So Far?
    - Message confidentiality can be achieved using symmetric key crypto and public key crypto
        - In symmetric key crypto, the sender and receiver utilize the same key for encryption as well as decryption of the data
        - In public key crypto, encryption and decryption are done asymmetrically
            - For instance, the sender can encrypt messages by utilizing the public key of the receiver, and then the receiver will decode the message utilizing its own secret key
    - Hash functions such as SHA-2 and MD5 can be utilized to generate a message digest
        - This can be usd to verify whether a message has been tampered with during transmission or not
    - Authenticity of digital messages
        - Not only are we interested in whether the message itself has been tampered with, but we also want to know if the originator of the message is actually the person they claim to be
            - Also, we want to ensure that someone else cannot generate another person's digital signature
    - Security attacks such as ARP poisoning, IP/MAC address spoofing, and phishing attacks all have one thing in common
        - The sender of the message is not the actual or original sender. An attacker can pretend to be someone else by spoofing his IP/MAC address
            - To combat this, authentication is used to verify the authenticity of the sender/receiver
- Authentication
    - The purpose of authentication is for communicating parties

  to prove their identity to one another
  - For example, if Bob and Alice are communicating with one another, how can Alice prove to Bob that she is indeed Alice? How can this be achieved? What is the solution to this problem?
- A real life example of authentication is when you login to your bank account, like `TD.com` or TDCanadaTrust.com`. You need to make sure that the webpage you load is indeed from TD, and not some kind of phishing website that an attacker can utilize to steal your banking credentials/information
  - Authenticating the remote host's identity is very important
- Digital signatures can be utilized to achieve authentication
  - The certificate can contain a digital signature that is signed by the certification authority (CA), and can be utilized for verifying the identity of the remote host
    - For example, when connecting to `TD.com` or `TDCanadaTrust.com`, there needs to be a public key that is only associated with TD and no one else. The public key is signed by the certificate authority (CA), so you know that you really are connecting to TD Canada
      - However, this is only part of the process
  - Authentication via digital signatures can be done via public key crypto or symmetric key crypto
- i.e. Diagram of Naive Approach

```
            |---------|-----------|--------|
            | Alice's | Encrypted | "Im    |
            | IP      | Password  | Alice" |
            | Address |           |        |
            |---------|-----------|--------|
|-------|                                         |-----|
| Alice |---------------------------------------> | Bob |
|-------|                                         |-----|
                        |-------|
                        | Trudy |
                        |-------|
```

  - Assume that Alice and Bob have some kind of pre-shared secret. This can be a passcode or phrase that was previously established through other means of communication. Can Bob utilize this common secret that has only been shared with Alice and no one else, as a means of authenticating Alice's identity?
    - This is a feasible way for authentication, which involves Alice sending a message to Bob containing information such as Alice's IP address, a message in clear-text, and the encrypted password
      - The encrypted password is the pre-shared secret between Alice and Bob, but it is encrypted because sending a password in clear-text is a

bad idea

      - In terms of authentication and security, this approach will not work, nor is it sufficient
        - The problem with this approach is that Bob does not know who sent the message. A third party, like Trudy, can copy and resend the message at a later time, and none will be the wiser; neither Bob nor Alice will know
          - This is called a replay attack
    - A third party, like Trudy, can sniff the transmission between Alice and Bob, and save all of their messages. Then, Trudy can simply re-send the saved messages to Bob, and pretend to be Alice
    - The naive approach of authentication does not work
      - It is problematic because:
        1. There needs to be a pre-shared secret passcode, and a key that's used to encrypt the passcode between Alice and Bob
        2. A third party can easily conduct a replay attack
    - To improve this authentication approach, there needs to be a mechanism that is able to prevent the reply of messages exchanged between communicating parties, such as Alice and Bob
      - Preventing a replay attack makes it harder for an attacker to be mis-identified as the true user
  - The mechanisms for preventing replay attacks are commonly used in cryptography, and in almost all security protocols
- Authentication: Symmetric Key Crypto
  - In cryptography, nonces are used to prevent replay attacks
    - A nonce is a large number that is only used once-in-a-lifetime
  - i.e. Diagram of Authentication Via Symmetric Key Crypto

```
|-------|       "I am Alice"       |-----|
| Alice |----------------------->>> | Bob |
|-------|                           |-----|

|-------|              R            |-----|
| Alice | <<<-----------------------| Bob |
|-------|                           |-----|

|-------|       ((K_A-B)R)          |-----|
| Alice |----------------------->>> | Bob |
|-------|                           |-----|
```

      - This diagram depicts authentication via symmetric key crypto
      - 'R' is a nonce
        - It is generated by Bob
      - The final message, ((K_A-B)R), is sent from Alice to Bob, and it is the encrypted form of the nonce

            - The nonce is encrypted using the pre-shared symmetric key
        - First, Alice will send a message to Bob
            - The message can be in clear-text, or plaintext
        - Then, Bob will randomly generate a number/nonce, 'R', and send it to Alice
            - Typically, nonces are associated with time and they are only used once
                - After a nonce has expired, an attacker cannot replay messages. Hence, it is critical for nonces to be only used one time, and they must be random enough so that other parties cannot guess it
        - Upon reception of the nonce, 'R', Alice can utilize the pre-shared secret between her and Bob to encrypt the nonce, and send it back to Bob
            - By doing this, Bob can infer two things:
                1. Alice is capable of receiving a nonce that Bob sent earlier
                2. """Alice""" has the shared secret and is able to encrypt the nonce, and send it to Bob
        - In this scenario, the communicating parties are not susceptible to a replay/playback attack
            - Even if Trudy can see the entire set of messages exchanged between Alice and Bob, she cannot impersonate either party. This is because the last message, ((K_A-B)R), is encrypted, and once the nonce expires it cannot be used for authentication purposes
        - This approach is reasonable and it accomplishes the goal of authentication. Alice is able to authenticate herself to Bob, without suffering from a replay/playback attack
            - However, we make two assumptions:
                1. Trudy, or any third party, cannot insert herself in the chain of communication
                    - Trudy can launch a man-in-the-middle (MITM) attack, and replay the last message. This will allow Trudy to potentially intercept subsequent messages
                2. Alice and Bob need to have a pre-shared secret key
- Authentication: Public Key Crypto
    - Public key cryptography can be used in conjunction with a nonce to achieve authentication
        - The benefit is that a pre-shared key is not required
            - This is the only difference between authentication via symmetric key crypto and authentication via public key crypto
        - The idea is similar to authentication via symmetric key crypto

        - Both authentication schemes use nonces to prevent replay attacks
            - The nonce is only used one time, and new nonce is generated for every subsequent conversation/ connection
- i.e. Diagram of Authentication Via Public Key Crypto

```
            |-------|        "I am Alice"           |-----|
            | Alice |--------------------------->>> | Bob |
            |-------|                               |-----|

            |-------|                 R             |-----|
            | Alice | <<<---------------------------| Bob |
            |-------|                               |-----|

            |-------|        (((K_A)-)R)            |-----|
            | Alice |--------------------------->>> | Bob |
            |-------|                               |-----|

            |-------|        "Send me your cert"    |-----|
            | Alice | <<<---------------------------| Bob |
            |-------|                               |-----|

                          |-----------------|
                          | Certificate for |
                          | Alice's public  |
                          | key; signed by  |
                          | certification   |
                          | authority (CA)  |
                          |-----------------|
            |-------|                               |-----|
            | Alice |--------------------------->>> | Bob |
            |-------|                               |-----|
                                                       |
                                                       |
                                                       |
                                           |-----------|
                                           |
                                           |
                                           V
            |----------|                |-----------|
            | Alice's  |                | Digital   |
            | public   | <-----------| signature | <---- Signed by
            | key      |                | (decrypt) |       CA's public
            | ((K_A)+) |                |-----------|       key, ((K_CA)
+)
            |----------|
```

        - 'R' is the nonce
            - It is generated by Bob
        - Alice's private key is: ((K_A)-)
            - This key is used to encrypt the nonce, 'R', in

the beginning

      - Alice's public key is: ((K_A)+)
        - Bob uses Alice's public key to verify Alice's identity, and to decrypt messages sent by Alice
          - This is possible because of the following property: [((K_A)+)(((K_A)-)(R))] = R
    - Assume that Alice and Bob are aware of each other's public key
    - First, Alice will send a plaintext message (i.e. "I am Alice") to Bob.
    - Next, Bob will respond to Alice's initial message with a nonce, 'R'
      - This message is also sent in plaintext
    - Then, Alice will encrypt the nonce, 'R', with her own secret key, (((K_A)-)(R)), and send it to Bob
      - Note: A superscript minus (-) is used to represent a private key, and a superscript plus (+) is used to represent a public key
      - Note: The subscript 'A' corresponds to the entity the key belongs to
        - 'A' stands for Alice
        - 'CA' stands for certification authority
    - At this step, Bob does not fully trust "Alice". Even though he does have her public key, Bob wants to make sure that the key is indeed associated with Alice
      - Bob can establish trust by requesting Alice for her certificate that is signed by the certificate authority (CA)
    - Bob sends a message request to Alice for her certificate
      - The certificate contains Alice's public key, and is signed by a certificate authority (CA)
    - Then, Alice will send her signed certificate to Bob
      - The certificate maybe digested via a hash function, and then encrypted with the private key of the certificate authority (CA)
    - Once Bob receives Alice's certificate, he is able to retrieve and verify Alice's public key. Next, he uses the public key to decrypt the encrypted nonce message, and retrieves the nonce
      - By verifying that the nonce is the same as the one he generated, Bob is able to authenticate Alice
- Outline
  - Attacks and counter measures
  - Security primer
  - Security protocols
    - SSL
    - 802.11i
    - IPsec VPN
  - Note: The discussion surrounding security primers is now complete

    - Security protocols are developed using the concepts from message integrity, message confidentiality, and authenticity of the signature as well as authenticity of the person involved in the communication
- Secure Sockets Layer (SSL)
    - In today's communications, SSL is the most commonly used security protocol
        - SSL is used by a variety of applications such as web browsers, email clients, directory services, etc.
    - Transport layer security (TLS) is utilized to support any TCP based application using SSL services
    - Based on the end-to-end design principle, it is not a good idea to encrypt messages on the network router
        - Typically, we rely on the end host to handle security primitives such as confidentiality, integrity, authentication, etc.
    - SSL sits on top of TCP
        - Different application layer protocols can be overlayed on top of SSL
            - For more information, refer to figure below
    - The most commonly used application layer protocol that is secured by SSL is HTTPS
        - It is used everytime you connect to a website, or load a webpage
        - Note: HTTPS is not a separate protocol from HTTP. Instead, it is a combination of HTTP and SSL. Simply put, HTTP sits on top of SSL to form HTTPS
            - Similarly, SMTPS is a combination of SMTP and SSL, where SMTP sits on top of SSL
            - IMAP is an email protocol that operates on top of SSL
    - TLS stands for transport layer protocol or security. This can be used interchangeably, because SSL is the basis of TLS and older version have been shown to be insecure
        - SSL is the basis of IETF Transport Layer Security (TLS)
    - Currently, the most comminly used secure transport layer security protocols are
        - TLS 2.0
        - TLS 3.0
    - SSL provides the following services:
        - Server authentication
            - SSL can also be used for client authentication, if the client can provide some kind of certificate
        - Data encryption
            - SSL allows the generation of multiple keys that can be used to encrypt the data that has been transmitted between the server and the client
                - This ensures message confidentiality
        - Data integrity
            - SSL can be used to ensure message integrity

        - This is done by generating a hash, applying a hash function, and then applying the message authentication code at the end of the message that has been transmitted
    - Client authentication
        - This is optional; often times when we communicate with a remote server, we are only interested in authenticating the server, not the client
- i.e. Diagram of SSL in The TCP/IP Protocol Stack

```
                                            |--- Protocols
                                            |    Secured
                                            |    By SSL
                                            |
                                            V
|-----------| |----------| |----------| |-------| |------|
| SSL       | | SSL      | | SSL      | | HTTP  | | HTTP |
| Handshake | | Change   | | Alert    | | LDAP  | | SMTP |
| Protocol  | | Cipher   | | Protocol | | IMAP  | | etc. |
|           | | Spec     | |          | | SMTPS | |      |
|           | | Protocol | |          | | etc.  | |      |
|-----------| |----------| |----------| |-------| |------|

|-----------------------------------------------|
|                SSL Record Layer               |
|-----------------------------------------------|

|--------------------------------------------------------|
|                           TCP                          |
|                    (Transport Layer)                   |
|--------------------------------------------------------|

|--------------------------------------------------------|
|                           IP                           |
|                    (Internet Layer)                    |
|--------------------------------------------------------|

|--------------------------------------------------------|
|                      Network Access                    |
|                     (Network Layer)                    |
|--------------------------------------------------------|
```

- SSL (Continued) (1)
    - To achieve server authentication, public key crypto is used
    - When a user connects to a remote server, the browser will alert them if they receive a certificate that is not signed by a recognized certificate authority (CA)
    - During the initial handshake between the server and the client, the certificate will be provided by the server, and the client's browser will be able to verify if the public key provided by the server is actually signed by a valid certificate authority (CA)

    - To summarize, server authentication is done in the following steps:
        - SSL enabled browser includes public keys for trusted CAs
        - Browser requests server's certificate that is issued by a trusted CA
        - Browser uses the CA's public key to extract server's public key from the certificate
    - Note: You can check your browser's security menu to see a list of trusted certificate authorities
- SSL (Continued) (2)
    - Once the server is authenticated, there are a number of ways that the client and the server can generate some kind of a symmetric session key
        - Since the client and server are using public key crypto, and both server and client have each other's authentic public key, they can continue to use public key crypto for subsequent messages
            - However, a symmetric session key is required/ generated, because public key crypto tends to be computationally heavier compared to shared key crypto. Also, multiple keys are needed, not just for the purpose of ensuring confidentiality, but another key is needed to generate the hash. If the client is communicating with multiple entities, more keys are needed
    - In SSL, in addition to verifying the server and obtaining its public key, the server and the client also need to have some kind of agreement over the symmetric session key that can be used for the purpose of data confidentiality and data integrity
        - The symmetric session key is used for subsequent data communication
            - Symmetric means that the key is shared between the server and the client
            - Symmetric keys are used, because they are computationally lighter than public keys
    - An optional step of SSL is to utilize client certificates for client authentication. However, this practice is uncommon
    - To summarize, an encrypted SSL session is done by:
        - Client's browser generates symmetric session key, encrypts it with the server's public key, and sends the encrypted key to the server
        - By utilizing its own private key, the server decrypts the message and retrieves the symmetric session key
        - Now, both client and server have the symmetric session key
            - All data sent into the TCP socket, by client or server, is encrypted with the symmetric session key
- SSL Plus RSA

```
- SSL is quite versatile, because the client and server can
  negotiate what kind of cipher suite they want to utilize
    - Depending on the cipher suite that is adopted/utilized,
      there are different mechanisms that can be utilized to
      generate the pairwise session key(s)
- The following diagram is an example of combining SSL with
  RSA for the purpose of authenticating the server, as well as
  generating the pairwise session key
- i.e. Diagram & Steps of Server-Client Key Exchange
    =========================================================
    == Step 1: Negotiation                                 ==
    =========================================================

                        -------------->>>
                        | ClientHello  |
                        |--------------|
                        | * Algorithms |
                        |   supported  |
                        | * Random     |
                        |   number #1  |
                        -------------->>>
    |--------|                                    |--------|
    | Client |---------------------------------->>> | Server |
    |--------|                                    |--------|
    - The entire process starts from the client. The client's
      web browser will send an initial message (i.e.
      "ClientHello"), to the server. This message is sent in
      cleartext/plaintext, and can be observed in Wireshark
    - The initial message, "ClientHello", contains information
      such as:
        - Nonce
            - The nonce is locally generated by the client
            - Note: A nonce is a random number
        - Cipher suites
            - A list of cipher suites that are supported by
              the client's web browser
    =========================================================
    == Step 2: Negotiation                                 ==
    =========================================================

                        <<<--------------
                        | ServerHello   |
                        |---------------|
                        | * Algorithms  |
                        |   selected    |
                        | * Random      |
                        |   number #2   |
                        | * SSL         |
                        |   certificate |
                        <<<--------------
```

```
|--------|                                      |--------|
| Client | <<<----------------------------------| Server |
|--------|                                      |--------|
- Upon reception of the initial message, "ClientHello",
  sent by the client, the server will respond with its own
  message (i.e. "ServerHello")
- The server's response, "ServerHello", contains
  information such as:
    - Cipher suites
        - These ciphers are supported by both the client
          and the server
    - Nonce
        - This nonce is locally generated by the server
        - Note: A nonce is a random number
    - SSL certificate of the server
        - The certificate contains the server's public key
=========================================================
== Step 3: Negotiation                                 ==
=========================================================

               |---------------|
               | Verifies SSL |
    |--------> | certificate  |
    |          | and extracts |
    |          | server's     |
    |          | public key   |
    |          |---------------|
|--------|                                      |--------|
| Client |----------------------------------->>> | Server |
|--------|                                      |--------|
                    ------------>>>
                    | Encrypts    |
                    | pre-master  |
                    | secret with |
                    | server's    |
                    | public key  |
                    ------------>>>
- The client will verify the server's SSL certificate
  using a certificate authority (CA) and public key crypto
    - This process authenticates the server
- Since RSA is being used, the client's browser will
  locally generate a pre-master secret/key. This key is
  encrypted using the server's public key, and then sent
  to the server
    - Because the pre-master secret is encrypted, third
      parties will not be able to see it, because it is
      not in cleartext/plaintext. Also, a replay attack
      will not work, because the only purpose of this
      message is to send the pre-master key to the server
- Note: This SSL + RSA combination is not commonly used
```

```
    - This is because if the attacker manages to get a
      hold of your computer, he will be able to see the
      pre-master key at the time it is generated, for this
      particular SSL session with the server
        - Since all subsequent keys are derived from the
          pre-master key, the attacker will be able to
          decode subsequent messages. However, the
          attacker needs to have physical access to the
          machine, or remote access, to be able to
          retrieve the pre-master key
            - Because of this vulnerability, this
              combination/option is not used nowadays
=========================================================
== Step 4: Negotiation                                 ==
=========================================================

|--------|                                      |--------|
| Client |                                      | Server |
|--------|                                      |--------|
                                                    |
                |---------------------|             |
                | Decrypts pre-master | <-----------|
                | secret using server |             
                | private key         |
                |---------------------|

- Once the server receives the pre-master secret, the
  server will decrypt it using its private key
    - The server's private key is only known to the
      server, and no one else
- After decrypting the pre-master secret, the client and
  server can now generate the master secret/key
=========================================================
== Step 5: Shared Secret                               ==
=========================================================

|--------|                                      |--------|
| Client |                                      | Server |
|--------|                                      |--------|
    |                                               |
    |                                               |
    |                                               |
    |-----------|                       |-----------|
                |                       |
                |                       |
                |                       |
                V                       V
  |-----------------------|    |-----------------------|
  | Compute master secret |    | Compute master secret |
  | using random numbers  |    | using random numbers  |
  | #1 and #2, and pre-   |    | #1 and #2, and pre-   |
  | master secret         |    | master secret         |
  |-----------------------|    |-----------------------|
- Once the pre-master key, that's generated by the client,
  is sent to the server and decrypted using the server's
  private key, both sides will independently generate
  further keys, namely the master key
- The client and server will independently compute the
  master secret/key, via the pre-master key, two nonces
  (random numbers) that were previously exchanged in
  "ClientHello" and "ServerHello"
    - Each host locally, and separately, generates the
      master key. Further communication is not needed or
      required
- Note: The Master key is utilized to generate other types
        of keys
=========================================================
== Step 6: Handshake Completion                        ==
=========================================================

                    ------------->>>
                    | ClientFinish |
                    |--------------|
                    | Create hash  |
                    | of messages  |
                    | using Master |
                    | secret       |
                    ------------->>>
|--------|                                      |--------|
| Client |---------------------------------->>> | Server |
|--------|                                      |--------|
- Upon generating a master key/secret, the client will
  send a finished message to the server (i.e.
  ClientFinish)
    - The client will also create, and send, a hash of
      messages that have been exchanged so far. This is
      another type of verification, because the master
      secret/key should be shared/common between the
      client and server
        - The Master secret/key is used to generate the
          hash of messages that have been exchanged so far
=========================================================
== Step 7: Handshake Completion                        ==
=========================================================

|--------|                                      |--------|
| Client |                                      | Server |
|--------|                                      |--------|
                                                    |
                |----------------------|            |
                | Compare ClientFinish |            |
                | hash with server     |            |
                | version of hash      | <----------|
                | created with Master  |
                | secret               |
                |----------------------|
- Once the server receives the clients finish message
  (i.e. ClientFinish), the server can compare whether the
  hash value is consistent with what it perceives
=========================================================
== Step 8: Handshake Completion                        ==
=========================================================

                    <<<-------------
                    | ServerFinish |
                    |--------------|
                    | Create hash  |
                    | of messages  |
                    | using Master |
                    | secret and   |
                    | send to      |
                    | client       |
                    <<<-------------
|--------|                                      |--------|
| Client | <<<----------------------------------| Server |
|--------|                                      |--------|
- The server generates its own hash message using the
  master secret/key, and sends it to the client
    - Also, the server will send a finished message to the
      client (i.e. ServerFinish) along with the hash
      message
=========================================================
== Step 9: Handshake Completion                        ==
=========================================================

|--------|                                      |--------|
| Client |                                      | Server |
|--------|                                      |--------|
    |
    |           |----------------------|
    |           | Compare ServerFinish |
    |---------> | hash with            |
                | ClientFinish hash    |
                |----------------------|
- Finally, the client can verify the server by comparing
  its own locally computed hash with the hash sent by the
  server (i.e. ClientFinish hash VS. ServerFinish hash)
    - Now, the client and server share of collection of
      keys that can be used for subsequent data
      communication
```

- SSL Cipher Suite
    - SSL can utilize multiple different types of ciphers
        - Different browsers and different servers may support different types of ciphers
    - The list of supported cipher suites is exchanged in the initial 'Hello' messages between the hosts (i.e. ClientHello and ServerHello)
        - Cipher suite information is encoded in a String, and then sent to the respective host
    - The initial 'Hello' messages, ClientHello and ServerHello, will contain a String that looks something like:
        - ECDHE_RSA_WITH_AES_256_GCM_SHA385

          OR
        - RSA_WITH_AES_128_CBC_SHA256
    - A cipher suite string (message) needs to specify four fields that are associated with the cipher suite
        - The four fields are:
            - Key exchange
                - What kind of key exchange mechanism is being used?
                    - i.e. RSA, Diffie-Hellman, etc.
                - The key exchange mechanism/algorithm is used to generate the shared pre-master secret
            - Authentication
                - How is authentication done? Or how can the server be authenticated?
                    - i.e. Typically, we utilize RSA to generate the public key and private/secret key pair. Also, a certificate is used for the purpose of authentication
                        - Note: There are other authentication alternatives to RSA
            - Cipher
                - What cipher is utilized to encrypt the data?
                    - i.e. 3-DES, AES, etc.
                        - A cipher suite needs to be specified for the purpose of confidentiality for subsequent data exchange
            - Hash
                - What kind of hash function is used?
                    - i.e. MD5, SHA-2, SHA-3, etc.
        - i.e. Cipher Suite Table

Key Exchange	Authentication	Cipher	Hash
RSA	RSA	3DES	MD5
Diffie-Hellman	DSA	AES	SHA

```
| ...            | ...            | ...    | ...  |
|----------------|----------------|--------|------|
```

      - Note: There are many, many, many different types of ciphers and cipher suites
  - Cipher suite string message examples:
    - ECDHE_RSA_WITH_AES_256_GCM_SHA385
      - The four fields in this string are:
        - ECDHE
          - This is used for key exchange; it is a variation of Diffie-Hellman
        - RSA
          - This is used for authentication
        - AES_256_GCM
          - This cipher is used for message confidentiality
        - SHA385
          - This is the hash function; the number corresponds to the number of bits in the digest/output
    - RSA_WITH_AES_128_CBC_SHA256
      - The four fields in this string are:
        - RSA
          - This mechanism/algorithm is used for BOTH key exchange and authentication
        - AES_128
          - Encryption algorithm
        - SHA_256
          - Hashing algorithm
  - 'GCM' and 'CBC' are a specific type of cipher that is utilized by hosts
    - 'CBC' is cyclic block code
  - The cipher suite String is part of the initial 'Hello' message
- Each key exchange mechanism generates the pre-master key in a different manner
  - In the case of RSA based key exchange, the pre-master key is locally generated by the browser
  - In Diffie-Hellman, the pre-master key is derived by following the Diffie-Hellman algorithm, where client and server communicate with one another
    - Both hosts end up with the same pre-shared-master key
- One the pre-master key is generated, the hosts utilize it along with the previously exchanged nonces to compute a master key
- From the master key, multiple keys can be derived, such as:
  - Enc Key
    - This key is used for the encryption of data
  - HMAC Key
    - This key is used for hashing

        - Stands for hash message authentication code
      - Initial Vector
        - Some protocols use ab initial vector such as cyclic block code (CBC), which requires an initial vector before it can start the encryption process
    - i.e. Diagram of Relationship Between Keys

```
                                            |---------|
                           |--------------> | Enc Key |
                           |                |---------|
                           |
                           |
                           |
|--------|           |--------|             |------|
| Pre-   |---------> | Master |-----------> | HMAC |
| Master |           | Key    |             | Key  |
| Key    |           |--------|             |------|
|--------|                 |
                           |
                           |
                           |                |---------|
                           |--------------> | Initial |
                                            | Vector  |
                                            |---------|
```

      - The master key is derived from the pre-master key
      - The encryption key, the HMAC key, and the initial vector are all derived from the master key

- April 7th, 2021
  - SSL + RSA
    - SSL is the protocol for transport layer security
      - It can provide data confidentiality, message integrity, and authentication for TCP based applications such as HTTP, SMTP, and direct services.
  - IEEE 802.11 Security
    - The IEEE 802.11 security protocols operate at the data link layer, and are responsible for ensuring message integrity in wireless local area networks (WLAN)
    - In today's world, Wi-Fi networks are widely deployed
      - Individual corporations/institutions have their own enterprise networks. They may deploy Wi-Fi in their commercial buildings/campuses
      - Homes have their own private network
        - In an apartment building, the majority of occupants/ tenants have their own individual access point, or Wi-Fi network
        - Driving around a suburb, like your neighbourhood, will reveal lots of Wi-Fi networks
    - Over the years, people have become more conscious about potential security problems related to Wi-Fi
      - In contrast, several years ago, a lot of Wi-Fi networks

were open, allowing anyone to join the nework, and anyone can see the data transmission pertaining to the Wi-Fi network in cleartext/plaintext. Attackers were able to use sniffers to extract information from over-air transmissions
        - i.e. If you drive around Bay area, you'll find that more than 9000 802.11 networks are accessible from public roadways, and 85% use no encryption/authentication
            - This is known as War-driving; it is used by attackers to find vicims for packet-sniffing and executing various attacks
    - Nowadays, it is a lot more common for people to deploy some kind of encryption mechanism over their Wi-Fi network
- The purpose of 802.11 security is:
    1. Encryption
        - Also known as data confidentiality
        - Encryption prevents attackers from knowing exactly what users are doing on the Internet, by scrambling their data
            - i.e. You don't want your neighbour to sniff your data over the air, and know exactly what you're doing on the Internet
    2. Authentication
        - Only specific users have access to a particular network
            - i.e. You don't want unauthorized users to be able to connect to your Wi-Fi network, and enjoy your Internet service for free
- Historically, and unfortunately, the attempt to secure Wi-Fi has not been a smooth process
    - Early attempts at securing Wi-Fi turned out to be a catastrophic failure
        - The first iteration of a Wi-Fi security protocol called Wired Equivalent Privacy (WEP) was created with the goal to secure Wi-Fi, and have security properties analogous to a wired network
            - It turns out, WEP had a lot of vulnerabilities in its protocol
            - WEP is now obsolete, and is no longer adopted
- Currently, the most widely adopted Wi-Fi security protocol is (called) 802.11i

- 802.11 Security Overview
    - There are 2 main factors that constitute 802.11 security:
        1. Authentication and access control
            - The user needs to prove their legitimacy to the network before they can associate with it and enjoy wireless connectivity
                - This is the primary role of authentication;

authenticating the user
        - Sometimes the access point (AP) may need to be authenticated to ensure that the access point (AP) is not a rogue access point (AP), or an access point (AP) setup by a malicious attacker
    2. Data encryption, and message integrity and confidentiality (MIC)
        - This ensures data confidentiality and message integrity
        - Data that is transmitted over the air, or through a broadcast medium should be infeasible/impossible to decode if the transmission is captured/sniffed.

Also,

data sent during the transmission should be tamper-proof, and attackers should not be able to modify it
            - The receiver should have a way to verify whether the message was corrupted or tampered with. If so, drop the packet and request retransmission
- Authentication protocols
    - The earliest attempt of authentication was Open Authentication; a very naive/humble approach
        - Open Authentication means no authentication. Users do not need to authenticate to join the network, because it is an open network and anyone can join
            - Eventually, open authentication was dropped in favour of more secure protocols
    - Pre-shared-key authentication is a simple authentication protocol that allows users/guests to connect to a Wi-Fi network via a key or password
        - Pre-shared-key is primarily used in home settings and small businesses/enterprises, because complex infrastructure is not required for authentication
    - 802.1x authentication is a more sophisticated authentication protocol than open authentication and pre-shared-key authentication
        - 802.1x authentication is widely adopted by large enterprises for their network(s). It requires some infrastructure to setup, but it allows network administrators to easily revoke a user's access/rights by modifying the corresponding entry in the backend
    - The 3 protocols used for the purpose of authentication and access conrol are open authentication, pre-shared-key authentication, and 802.1x authentication
        - Open authentication is an exception, because it has no authentication at all
- Data encryption protocols
    - The earliest attempt at 802.11 security is Wired Equivalent Privacy (WEP); it was a complete disaster
        - Wired Equivalent Privacy used RC4 as a cipher, which

        turned out to be very insecure and vulnerable to attacks
    - For home or small enterprise networks, WPA or WPA2 Personal is the standard for encryption and ensuring message integrity/confidentiality
        - WPA/WPA2 Personal utilize the TKIP cipher for encryption
    - In large enterprise settings, WPA2 is the dominating protocol
        - WPA2 utilizes the AES encryption algorithm
- i.e. Diagram of 802.11 Security Protocols

```
                        |------------------------------------|
                        |            802.11i Suite           |
                        |                                    |
    |----------------|  |----------------|  |----------------|
    | Open           |  | Pre-shared-key |  | 802.1x         |
    | Authentication |  | Authentication |  | Authentication |
    |----------------|  |----------------|  |----------------|
                        |       |                   |        |
                        |       |                   |        |
                        |       |                   |        |
                        |       |                   |        |
                        |       V                   V        |
    |---------------|   |---------------|   |----------------|
    | Wired         |   | WPA           |   |                |
    | Equivalent    |   |  or           |   | WPA2           |
    | Privacy       |   | WPA2 Personal |   |                |
    |---------------|   |---------------|   |----------------|
                        |                                    |
                        |            802.11i Suite           |
                        |------------------------------------|
```

- The authentication and access control protocols are:
    - Open Authentication
    - Pre-shared-key key Authentication
    - 802.1x Authentication
- The data encryption and message integrity/ confidentiality protocols are:
    - Wired Equivalent Privacy (WEP)
    - WPA/WPA2 Personal
    - WPA2

- Pre-shared-key authentication is used by homes and small businesses
    - The encryption cipher used by pre-shared-key authentication is TKIP
- 802.1x authentication is used by large enterprises
    - The encryption cipher used by 802.1x authentication is AES
- The Wired Equivalent Privacy contains many vulnerabilities, partly due to the weak cipher it uses, RC4

        - Currently, WPA2 is the dominating protocol
    - Question: Does WPA/WPA2 Personal have an vulnerabilities? Is it possible to sniff out a password?
        - Answer: So far WPA/WPA2 Personal and WPA2 are known to be quite secure. It is non-trivial for an attacker to be able to reverse engineer and extract the session key, or any other bit of information. The WPA security protocols were developed to be more secure than their predecessor, Wired Equivalent Privacy (WEP)
    - 802.11i refers to the following combination of authentication and data encryption mechanisms:
        - Pre-shared-key Authentication --> WPA/WPA2 Personal
        - 802.1x Authentication --> WPA2
- Open System Authentication
    - Open Authentication is a very simple approach
        - It is the foundation of all other authentication methods, including 802.11i
            - Regardless of which authentication mechanism is used, the initial messages are exchanged via open authentication
        - Note: Open Authentication is not a good approach, because it lacks security
    - There are two mechanisms for 802.11 association:
        1. A station can scan beacon messages in its surrounding area, and try to associate with a particular access point
        2. A station can proactively send probe requests, get responses from nearby stations for those requests, and then the station can decide which access point it wants to associate with
    - i.e. Diagram of Open Authentication

```
|-----|                                      |--------|
| STA |                                      | AP STA |
|-----|                                      |--------|
   |               Probe Request                     |
   |------------------------------------------------>>> |
   |                                             |
   |                 Probe Response              |
   | <<<-----------------------------------------|
   |                                             |
   |   Open System Authentication Request        |
   |----------------------------------------->>> |
   |              (STA Identity)                 |
   |                                             |
   |      Open System Authentication Response    |
   | <<<-----------------------------------------|
   |                (STA Identity)               |
   |                                             |
   |           Association Request               |
   |----------------------------------------->>> |
   |                                            |
   |             Association Response           |
   | <<<----------------------------------------|
   |                                            |
   |                                            |
```

- 'STA' stands for Station
  - i.e. A wireless device
- 'AP STA' stands for Access Point Station
  - i.e. A router
- Open Authentication corresponds to a pair of messages exchanged
  - i.e. Open System Authentication Request/Response
- Once the station receives a probe response from the access point (AP), it will send an open authentication request
  - The request contains the identity of the station
    - i.e. The MAC address of the station
- Upon reception of the open authentication request, from the station, the access point (AP) sends an open authentication response message to the station
- The open system authentication request/response messages are only exchanged between the mobile station and the access point (AP)
  - A third party is not involved
- The open system authentication messages do not authenticate the user to the network, or vice versa
  - However, open system authentication messages are still utilized for legacy reasons
- Once the open system authentication messages are exchanged between the hosts, the station will send an association request message to the access point (AP)
  - The access point (AP) will reply to the request with an association response message
    - At this point, if 802.11i is utilized - in the form of pre-shared-key authentication or 802.1x authentication - the hosts have to go through steps to establish keys for subsequent data communication, and perform proper authentication
- After association request/response messages are exchanged, it does not mean that the station has successfully joined the network
  - However, at this stage the access point (AP) is able to receive some limited messages
    - i.e. Messages pertaining to subsequent authentication request/response

- To summarize:
  - Open authentication establishes an IEEE 802.11 association with no authentication

- 802.11i: Four Phases of Operation
  - i.e. Diagram of 802.11i

```
             Client                          Access       Authentication
             Station                         Point                Server
            |-----|                         |----|                |----|
            | STA |                         | AP |================| AS |
            |-----|                         |----|      wired     |----|
                                                       network
            <--------------------------------->
              (1) Discovery of security
                  capabilities

            <-------------------------- * * * --------------------->
              (2) 'STA' and 'AS' mutually authenticate; together
                  they generate the Master Key (MK). 'AP' serves
                  as "pass through"

              (3) 'STA' derives                <----------------------
                  Pairwise Master                   (3) 'AS' derives
                  Key (PMK)                             same 'PMK',
                                                        and sends
                                                        to 'AP'
            <--------------------------------->
              (4) 'STA' & 'AP' use 'PMK' to
                  derive Temporal Key (PTK);
                  used for message encryption,
                  integrity, etc.
```

- Legend:
    - 'STA' = Client station
    - 'AP' = Access point
    - 'AS' = Authentication server
- The operation of authentication and key exchange in 802.11i can be conceptually broken down into 4 phases; this is outlined in the diagram above
    - Each phase can have multiple message exchanges
        - The diagram above illustrates what kind of information is exchanged during a particular phase
    - This operation happens after the association request/ response messages are exchanged
- The authentication process starts with the station sending a message to the access point; to discover what kind of security property the network has
- In 802.11i, 3 entities are involved in the authentication process: the station, the access point, and the authentication server
    - For pre-shared authentication, the authentication server (AS) is part of the access point (AP). In other words, it sits on the access point
    - In enterprise Wi-Fi, the access point (AP) and

authentication server (AS) are separate entities

    - The authentication server (AS) is connected to the access point (AP) through a wired network; it is hosted by some server
    - The authentication server maintains a directory that records information such as users, their passwords, etc.

- In the 2nd step of the 802.11i operation, the wireless device (STA) and the authentication server (AS) mutually authenticate
  - There are different ways this can be achieved
  - Both devices will generate a Master Key (MK)
    - This step can be skipped if the pre-shared key is already known
      - In other cases, this key needs to be generated on the fly
  - Communication between the wireless device (STA) and the authentication server (AS) is done through the access point (AP)
    - The access point (AP) relays messages to or from the wireless device (STA) and authentication server (AS)
      - The access point (AP) is not really involved in this process; it provides physical connectivity
- In the 3rd step both devices, client station (STA) and authentication server (AS) can derive their pairwise master key (PMK)
  - The pairwise master key (PMK) is dervied from the previously generated master key (MK), or from a pre-shared secret that the devices may already have
  - Since this operation is done locally on each device, the access point (AP) has no information about this
    - This information needs to be stored on the access point (AP) from the authentication server, over the wired network
      - At this point, the pairwise master key (PMK) is now available on all 3 Wi-Fi entities: the Wi-Fi client (STA), the access point (AP), and the authentication server (AS)
- During the 4th step, the Wi-Fi client (STA) and access point (AP) use the pairwise master key (PMK) to derive a pairwise temporary key (PTK)
  - This is because the main motive is to secure a connection between the Wi-Fi client (STA) and access point (AP)
  - The pairwise temporary key (PTK) is used for encryption, hashing, message encryption, integrity, etc.
    - Depending on the cipher used, the pairwise temporary

```
                  key (PTK) maybe used as an initial vector
        - This diagram is the general flow of 802.11i
            - Depending on what authentication method is used, the
              details may differ
        - Question: Why isn't the access point (AP) able to generate
the
          pairwise master key (PMK)?
            - Answer: The pairwise master key (PMK) is generally based
              on information such as the pre-shared secret or
passcode,
              which is stored on the authentication server
                - Logging into a home network requires the user to
enter
                  the passcode, which is pre-stored. This information
is
                  stored in the authentication server, which sits on
the
                  access point
                - In an enterprise network, typically there is a
waiting
                  server that contains a directory of passcode and
                  users. In this situation the access point serves as
a
                  pass through for communication. Hence, the access
                  point does not have the required information, like
                  password or certificate, to perform authentication
                    - Even though information is physically relayed by
                      the access point, it does have have the means to
                      decode the information or verify whether the
                      information is authentic.
    - PreShared Key (PSK) Authentication
        - This is typically used for home Wi-Fi authentication
            - It can also be used for a small enterprise
        - In a home or small enterprise network, a passphrase is
already
          shared between the device and authentication server (AS)
            - Note: In this situation, the authentication server (AS)
is
                    co-located with the access point (AP)
        - Pre-shared key authentication is relatively simple; it is
          already at the 2nd phase of the four phases of 802.11i
        - i.e. Diagram of PSK Authentication
                |-----|                         |----|
                | STA |                         | AP |
                |-----|                         |----|
                   |                              |
                   |            Anonce            |
                   | <----------------------------|
                   |                              |
                   |                              |
            |------------|                       |
            | STA        |                       |
            | constructs |                       |
            | the PTK    |                       |
            |------------|                       |
                   |                             |
                   |         SNonce + MIC        |
                   | ----------------------------->
                   |                             |
                   |                             |
                   |                             |
                   |                      |---------------|
                   |                      | AP constructs |
                   |                      | the PTK       |
                   |                      |---------------|
                   |                             |
                   |         enc(GTK) + MIC      |
                   | <---------------------------|
                   |                             |
                   |              Ack            |
                   |----------------------------> |
                   |                             |
                   |                             |
            - Legend:
                - 'STA' = Station (Client)
                - 'AP'  = Access Point
                - 'PMK' = Pre-Master Key
                - 'PTK' = Pairwise Temporary Key
                - 'MIC' = Message Integrity Check
                - 'GTK' = Group Temporal Key
        - The pre-master key (PMK) is generated by:
          PBKDF2(Passphrase, SSID, SSID_Length, 4096, 256)
        - The pairwise temporary key (PTK) is generated by:
          PRF512(PMK, AMAC, SMAC, ANonce, SNonce)
        - In a home network setting, the pre-shared passphrase is used
          to derive a pre-master key by utilizing a fixed algorithm
            - In addition to the passphrase, the SSID of the Wi-Fi
              network, the length of the SSID, and other values, are
              used to derive a pre-master key
                - This derivation is deterministic
            - Once the passphrase is known, it is directly mapped to
the
              PMK
                - This entire process corresponds to phase #3 of the
                  four phases of 802.11i
                    - See previous slide for more information
            - Note: In a home network setting, the master key does not
                    need to be generated, because the master key is
                    already stored on the device ('STA') and access
                    point ('AP')
```

- The next step of the PSK authentication process is to derive the pairwise temporary key (PTK). This is done via further message exchanges between the access point (AP) and the user (STA)
  - The pairwise temporary key (PTK) needs to be used one time
    - In other words, every session has it's own PTK
      - Also, the PTK should not be deterministic like the PMK; the PMK is deterministic because it depends on a fixed passphrase
  - Generating a pairwise temporary key (PTK) uses the same trick as preventing replay attacks; nonces
    - The nonces exchanged between the client (STA) and access point (AP) is used to generate the PTK
  - After the derivation of the pre-master key (PMK), the access point sends a nonce (ANonce) to the client (STA). This message is sent in cleartext
    - Then, the client (STA) takes its own nonce (SNonce), ANonce, the MAC address of the access point (AMAC), its own MAC address (SMAC), and the PMK to generate its pairwise temporary key (PTK)
    - Also, the 'SNonce' from the client (STA) is sent to the access point (AP), along with a message integrity code (MIC)
      - The message integrity code is encrypted with the pairwise temporary key (PTK)
- Upon reception of the SNonce and MIC, the access point (AP) has all the necessary information to derive the pairwise temporary key (PTK)
  - The necessary information includes: the pre-master key (PMK), MAC addresses of both devices (AMAC and SMAC), and both nonces (ANonce and SNonce)
  - Now, the access point (AP) is able to generate additional keys such as the group temporal key (GTK)
    - The GTK is generated by the access point (AP), and it is used for broadcasting
      - i.e. If you want to reach multiple stations at the same time, the GTK can be used to encrypt the message. The message that includes the GTK is encrypted via the key dervied from the PTK that both devices previously agreed upon
  - From this point on, the client (STA) is authenticated to

be connected to the Wi-Fi network, and it can proceed with data communication.

- Even though the PSK authentication protocol is relatively simple, compared to other protocols, it accomplishes two things:
  1. Authenticate the user/client to the network by utilizing the passphrase
     - The passphrase is assumed to be a secret that is only shared between legitimate devices and the access point (AP)
  2. Generate necessary keys including the pairwise temporary key (PTK) and the group temporal key (GTK)
     - Both keys are used for subsequent communication for data integrity and data confidentiality

- 802.1x Authentication
  - Enterprise networks have the ability to authenticate and revoke users
    - i.e. Revocation means that a user won't be able to access the network anymore
    - In contrast, in a home network, the only way to prevent someone from stealing your Wi-Fi is to change the passphrase for all users
      - But with 802.1x authentication, a user's access can be revoked without disrupting other users' access
  - In 802.1x authentication, there are 3 entities:
    1. Supplicant
       - This is the same as the client (STA) mentioned in a previous slide
       - A supplicant corresponds to a device that needs to be authenticated
    2. Authenticator
       - This is the access point (AP) that sort of passes through the information
         - It is also the secured channel where communication is established between the supplicant and the access point (AP)
    3. Authentication Server
       - The setup is similar to the previous slide, but now the authentication server (AS) is an independent entity
    - Note: The setup is the same as pre-shared key (PSK) authentication, but client (STA) is renamed to

supplicant

- In 802.1x the authentication server (AS) typically uses something like a RADIUS server, which keeps track of information such as username, password, etc.
- Authentication in 802.1x goes through the following steps:
  - Prior to authentication, the message exchange follows open authentication, and association; the user will send its credential information to the access point
    - i.e. If a user is trying to join McMaster's Wi-Fi, they may send their MacID (username) and password for during the association process
  - Once the authenticator, or access point (AP), receives the user's credentials, it will forward the information to the authentication server (AS)
    - The authentication server (AS) can check the credentials to see if the given MacID is a legitimate user and is authorized to use campus Wi-Fi.
    - During the association process additional information, like passwords, are sent to the authentication server
      - The password is not sent directly in the form of plaintext. The user can utilize a mechanism like challenge response or send the password via a secure (SSL) tunnel
  - At stage/phase 4, if the authentication server (AS) is able to verify the user's credentials (i.e. ID, passcode, etc.), then the authentication server (AS) will grant the user access to Wi-Fi services.
    - This information is sent through the access point (AP); and additional keys may be stored on the access point (AP)
      - The access point (AP) encapsulates the responses to the supplicant/client
    - After stage/phase 4 is successfully completed, the user can proceed with further communication
- To summarize, 802.1x authentication:
  - Is an IEEE standard for port-based network access control
  - Provides authentication for devices connected via LAN or

WLAN

- Uses a RADIUS server to check credentials
  - RADIUS stands for: Remote Authentication Dial-In User Service
- The steps required to authenticate a supplicant are:
  1. Client associates with access point (AP) using Open Authentication
  2. User's credentials are sent to the access point (AP)
  3. The authenticator forwards user credentials to the RADIUS server
  4. The user's credentials are checked against the entire database, which consists of all users' credentials
  5. If user's credentials are found in the database, then the user is granted a level of access
  6. The access point (AP) encapsulates the 802.1x reply and sends it to the supplicant
- Question: Why is the authenticator the same as the access point (AP)?
  - Answer: In this context, the authenticator is the access point (AP)
- EAP: Extensible Authentication Protocol
  - The actual implementation of 802.1x authentication is based on the extensible authentication protocol (EAP)
    - There is nothing complicated about EAP; it consists of a collection of requests and response messages
      - Depending on what authentication method is utilized, the sequences of message exchanges can be different, as well as the content of the message
  - EAP allows messages to be exchanged during the authentication and key generation process between the supplicant and authenticator, and between the authenticator and authentication server (AS)
  - EAP works over different layer-2 technologies, like wireless LAN (WLAN) or Ethernet LAN
    - It is called EAP over LAN, or EAPoL
      - Also, it can carry messages that correspond to the RADIUS server, over a wired network
    - On top of EAPoL, there may be EAP TLS, which gives an abstraction of the tunnel between the supplicant and authentication server (AS)
  - There are 3 types of authentication methods that can be used over EAP
    - i.e.
      - MD5

            - TLS
            - PEAP
    - In conclusion, EAP is:
        - An end-end client (mobile) to authentication server protocol
            - It was originally an extension of point-to-point protocol for dial-ups
        - Sent over separate "links"
            - i.e.
                - Mobile-to-AP (EAP over LAN; EAPoL)
                - Access point (AP) to authentication server (AS)
                    - i.e. RADIUS over UDP
        - EAP supports different authentication methods
            - i.e.
                - MD5
                - TLS
                - PEAP
    - i.e. Diagram of EAP

```
|-------------------------------------------------|
|                     EAP TLS                     |
|-------------------------------------------------|
|                       EAP                       |
|------------------------|------------------------|
| EAP over LAN (EAPol)   | RADIUS                 |
|------------------------|------------------------|
| IEEE 802.11            | UDP/IP                 |
|------------------------|------------------------|
```

- EAP:MD5

    - So the first kind is called EAP-MD5. So the MD5 part is actually utilized for the purpose of challenge response. So the setup is that the supplicant has some a pair of information identity and its passcode that is stored on the authentication server that can be utilized for authenticating the user. So if you adopt 802.11i with WPA2 and the EAP for authentication, then they're going to be some message exchange that from the client, from the supplicant, you send some kind of start message and we get some kind of request from the authenticator, and then you response with identity. So in assignment 6 if you open the trace that's been provided in the wireshark you can see the content of this response message. So up to this point everything is in cleartext. Upon reception of this (identity), the authenticator actually allow the, we call it the port open for EAP message, it's essentially you can think of the authenticator or the access point is actually set up a firewall where it say all messages will be dropped except for EAP message.

- So the response for EAPOL will be forwarded to the RADIUS like you are, like user identity will be sent to the authentication server. So let's say the server actually have the information of the username and passcode but clerly it's going to be a bad idea to send the passcode directly over the networking cleartext. So this, in EAP-MD5, it utilize challenge response, essentially it gonna, it essentially encrypt by utilizing MD5 to generate a digest with your passcode of message that's generated by the authentication server. So this serves a challenge and the because the supplicant has the same correct passcode it will be able to recover the message, the challenge that's been encrypted or been digested using the md5 hash function. So it will provide the challenge response and sent to the authentication server through access point or authenticator. So upon reception of this, that the authenticator is capable of verifying or to determine that the user is actually the legitimate user and the user actually has the correct passcode. So it will notify the authenticator, the access point that the user is legit and you can actually open up the port for all subsequent traffic. But at this stage, we are just finishing, we just finished the authentication, so we actually know that the user's legit so we can allow the user to transmit, but we haven't not generate the keys per say. So at this stage there will be a master session key that's been generated. This is gonna be similar to the passphrase that we like in the home network that is kind of shared between the authentication server and the supplicant, but the difference in this case is this master session key needs to be generated instead of pre-stored. So once you have the master session key you're gonna follow similar steps, how to generate the pairwise master key and you store the pairwise master key on the authenticator, on the access point, and you have other keys like GTK and the key for MIC, message integrity and etc. So from this point on its gonna be similar. So really up to this point that's what differs different authentication method, whether there's pre-shared key or there is MD5 or other things we're gonna discuss next.

- Question: What does EAP0L stand for?
    - Answer: It stands for Extensible Authentication Protocol Over LAN

- EAP:TLS

    - So the second way of doing authenticate is using TLS. So you may think this is weird, right? Yes, it's something operating on top of transport layer, it's transport layer security. Why we are actually using the protocol for the purpose of authentication for some kind of data link service. Again lets go back to this picture here. So the authentication that needs to be done is something that may traverse multiple local area network from the wireless network to the wired network that host a particular authentication server; this is just some kind of message exchange so it itself does not have to limit to be like a layer 2 messages. So in TLS, EAP-TLS the authentication is done through using certificate. So here the assumption is that the server has some certificate that is signed by some certificate authority and user also has certificate that's also signed by certification authority. And those certificates may be used for the purpose of authentication

- Similar to the previous case, your first gonna be some message exchanged between the supplicant, the wifi device, and authenticators or access point and upon reception of that information that access point gonna open the port for EAP traffic. And the AP will forward this information to the authentication server. And then the if you adopt EAP-TLS, then, by the way those options can be config'd, so if you adopt EAP-TLS there will be a message that indicates the starting of EAP-TLS, and the subsequent message; I'm kind of just lump multiple message together into that the this authentication server gonna use access point as a bypass to send the server certificate to the supplicant. The supplicant can verify the, you know it's a legit certificate from the server and the supplicant can also provide its client's certificate. And by the end of the process, essentially the supplicant and the server will mutually authenticate one another

- Up to this point, you're only finishing authentication, there will still be subsequent keys you generate the master session key and then derive additional keys for the actual data communication

- So, again the difference between this and the MD5, or this and the pre-shared key authentication is up to this point

- EAP:PEAP

  - So the last one is called EAP-PEAP. Which is actually I would say more commonly used nowadays. The primary reason for this is because in the EAP-TLS, it's common to have service certificate, but often times client do have have because you know certificate needs to be generated by the, it needs to be signed by a certificate authority, and often times we have some kind of self signed certificate that is not really acceptable for the purpose of authentication. So EAP-PEAP, I think PEAP stands for private EAP. It kind of get rid of the need for client to send there, to have certificate and will still be able to carry out the authentication. So everything up to the point that the server sends the certificate is the same as we had before, but once the server certificate is provided we can establish a TLS tunnel the similar way we have seen earlier for SSL, like after the receiving the certificate and you'll be able to generate some key that's used to encrypt the traffic between the supplicant and authentication server. So at this stage the authentication is not yet done, but you have a tunnel that all the messages will be encrypted so no attacker will be able to see those message. And within this tunnel the supplicant and the will actually provide the, to utilize the challenge response mechanism to authenticate itself to the server. So the subsequent message is gonna be similar to what we see in the MD5 case, but instead of sending the challenge response themselves in cleartext, those messages are actually sent through the TLS tunnel as such they are encrypted. So you can see compare this with MD5, it's more secure and also has this added mechanism of you know authenticate server

- So once the challenge response is done and the user's identity is verified then the TLS tunnel is torn down and then you can proceed to the next step of key generation and communication

- This is a little bit of you know slightly complicated, but conceptually if you think about what needs to be done, right? So you need to do authentication, and you need to do generate keys and for authentication there are different mechanisms that they may differ like in the ways that whether you leverage certificate whether you do channel response, whether the channel response actually sending clear text or not

- Question: What's the major difference in the challenge response?
    - Answer: It's actually very much the same. So, essentially the, so here again the assumption here is that the authentication server keep track of the user and passcode and it will be able to encode some challenges utilizing the user's password, so the user can actually respond to the challenge by applying its passcode and send it back. So the difference is not soo much in the message, but in the message that in the ways of message is sent. They are sent inside the TLS tunnel. So those message are encrypted, that the main difference.

- What Is Nework Layer Confidentiality?

    - So we are gonna talk about something in the middle, that's called IPsec. So IPsec is a pretty complex protocol, so we will not be able to get into a lot of details. But I just wanna describe it in a little bit more conceptual way and also we are gonna describe how it's connected to virtual private network

    - So previously when we talk about SSL, we really really want to secure applications, right. So the, your TCP, your IP, the all the datagram of TCP segments those are not encrypted its the payload, the application payload in the transport layer that are encrypted.

    - When we talk about 802.11i we actually just securing the last hop between the access point and the wifi clients. So the message are, will be encrypted at layer-2, so it also means that your TCP, your IP, and your application, all of them are becoming the payload of layer-2 frames such that attacker will not be able to see it. However, this is only limited to the last hop, it's not end-to-end. So once the message leaves the access point, its you know, it becomes cleartext, its really there's no additional mechanism then, it will be the same as any normal data you exchange in Internet. So what network layer security want to do kind of something in the middle. It wanna have the ability to kind of give you a illusion that this end-to-end network is secure, so you could actually, you don't have to reveal the information of the transport layer and in some cases you can even make the IP address invisible from third party. So this kind of, this network layer security kind of provide a blanket coverage. It doesn't differentiate whether this is UDP or TCP, it doesn't if it's application, what kind of application whether it is HTTP or something else.

- And this is very useful to, for the, to enable enterprise to have, for example: there are different clusters for networks to communicate with one another or like between employees of enterprise that try to access enterprise network remotely. Like previously you have to use VPN, McMaster VPN in order to connect to mosaic, but now I think after the pandemic they removed that restriction. So if you think about how to implement this notion of private network over a public Internet, it's actually non-trivial. So previously that corporates they will do they actually have their to construct a private network they have to lay down their individual hardware like fibre optical or cable network, and physically the network is actually separated from public Internet; this is very costly. So now the private network is implemented using an overlay over the existing insecure, unsecure public Internet, called virtual private network (VPN). It's not something physical that you have to have the physical hardware or dedicated routers, etc. It'll still use the public Internet infrastructure, but it will overlay on top

- Virtual Private Networks (VPNs) (1)

  - The virtual private networks are used in 2 scenarios. One is we call the site-to-site VPN. The other is what we call that the remote access VPN.

  - So in site-to-site VPN, this is the scenario that you may have, for example: you may have a branch office in different city or even countries, and you headquarters, and you wanna communicate with one another, but you wanna have the illusion that they are kind of belong to a virtual private network (VPN)

  - So site-to-site VPN is typically done through by installing something like a VPN and IPsec enabled router. And in between the routers so the traffic will be encrypted but from the devices within individual subnet to the router they can just carry on their existing communication. So in that sense that individual device do not need to store any additional software, they can just communicate as if they communicate to other outside servers. But then the data reaches the IPsec enabled router, the router will take over, and will encrypt the traffic that originated from client as part of the payload of IPsec data and further encapsulate that and insert some kind of IP header that address to the destination router. So when this message is received at the destined router, this router will take the payload from the message it received and forward the message to the respective destination of the message

        - So essentially the, this actually give from a client's point-of-view, like the server, they are actually transparent. All the magic actually happen at the router

        - So compare this to a remote access VPN, like what we typically use, for example: access campus network, we need to install some kind of software like for example cisco end connect. We need to take care of the fact that for some connection we want to use a Mac VPN. So the data that's been sent out from your device if it is connected to Mac VPN, will be kind of encrypted and part of the payload of IPsec and eventually IP datagram as sent through the public Internet, and may get received to IPsec gateway on campus, and eventually sent to the respective servers on campus

        - Here, your device need to take care of the encryption and necessary encapsulation of message, outgoing message, and incoming message that's being received so that's why there's a we call remote access VPN compared to the previous case, site-to-site VPN, the client services the, they do not need to bother with installing additional software, the IPsec router would handle everything for them

        - So this is a high level overview

    - Q/A
        - Question: So why do you choose between EAP MD5, PEAP, and TLS?
            - Answer: So generally MD5 is not as secure because the challenge responses are sent in cleartext compared to for example in PEAP the challenge response is actually sent through the secured TLS tunnel. But the TLS actually utilize, requires certificates on the user side, so that make it more complex and not often actually practical to implement. Think about you know your device don't typically have a certificate that is signed by certificate authority. So in term of the practicality, EAP-TLS is not as good as EAP-PEAP, although PEAP as you can see throughout our discussion actually utilize TLS tunnels. But in terms of security they are kind of have similar properties

- Question: Are WEP and WPA2, are these open source, can I obtain the RFC for these?
    - Answer: I think the descriptions are available like the WPA is actually some standard that's specified by fire lines. WEP we should be able to find it as well, and in fact you would should be able to find the various attack that people come up with too exploit the vulnerability in WEP

- Question: The first time I connect to an access point, does it authenticate on the server or sometimes does it just do it on the access point?
    - Answer: For home network, most likely you will use the pre-shared key authentication. So every single time you connect you have to authenticate. It's not like you authenticate once, you have a token and you don't need to do it further in the future, and the whole process only involve the your device and access point. The access point actually have the dual role of both authenticator and authenticator service. So it actually has to store the pass code and follow this, you know those functions to generate PMK and message exchanges to generate the pairwise temporary key

- Question: Is the nonce something generated on demand?
    - Answer: So this should be random. So it should be different every single time. It's just a random large random number, just one time.