DES332 Final Summary

Computer and Network Security - Nguyen Duy Hung
SIIT DE-ASD Y3T2/2021 – By Paphana Yiwsiw (@waterthatfrozen)

Lecture 7 - Introduction to Cryptography

Encryption for Confidentiality

- Data Confidentiality: make sure that data is not available to unauthorized individuals.
- Encrypt the original data and anyone can see it, but only authorized individual can decrypt it.
- Used for transmitting and storing data throughout the network and computer system.

Terminology

- Plaintext = Original message
- Ciphertext = Encrypted message
- Encryption = From plaintext to ciphertext
- Decryption = From ciphertext to plaintext
- Key = information used in ciphering only known to sender and receiver.
- Cipher = cryptographic algorithm
- Cryptography = study of encryption algorithms

Requirements and Assumptions

- Secure symmetric encryption requirements:
    - strong encryption algorithm: attacker cannot gain key or plaintext, given algorithm and ciphertext.
    - only sender and receiver know secret key and keep it secret.
- Assumption: known cipher algorithm and secure key distribution channel.

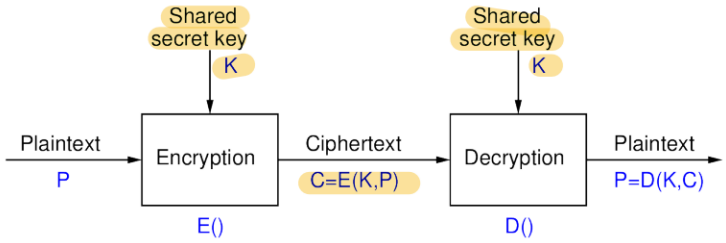
Cryptographic System Characteristics

- Operations: substitution (replace one with another)/ transposition (rearrange)/ product system (multiple stages of subs, and trans.)
    - Substitution:
        - Ceasar cipher: shift plaintext letters K positions to right (wrapping too).
    - Transposition:
        - Rail-Fence: plaintext in diagonals over K rows; ciphertext by reading row by row.
    - Product system:
        - Encrypt: Caesar cipher K1 → Rail-fence cipher K2
        - Decrypt: Rail-fence decipher K2 → Caesar decipher K1
- Key used: symmetric (both sides use same key; shared key) / public key (each side used a different key/asymmetric.)

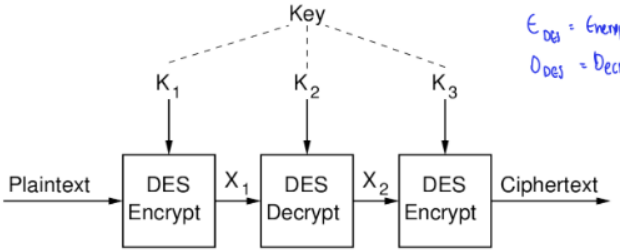
Attacking

- Attacker's goal: discover the plaintext and/or key.
    - Assume knowledge: ciphertext, algorithm, pairs of (plaintext, ciphertext) of same key.
- Methods: Brute force (try every possible key) / Cryptanalysis (exploit algorithm to deduce)
    - Assume: attacker can recognize correct plaintext

Symmetric Key Encryption for Confidentiality



- Requirements
  - Strong encryption algorithm (attacker should be unable to find plaintext or key)
  - Shared secret keys (no one else apart from sender and receiver knows the key)
- DES: Data Encryption Standard (currently not recommended)
  - 64 bits block, 56 bits + 8 parity bits key (key size is now insecure, algorithm is secure)
  - Process: Initial and final permutations, then 16 rounds of permutations and substitutions
  - Encryption Operations
    - same algorithm for both encryption and decryption
      result of current cycle (L_i, R_i) depends on that of previous cycle (L_{i-1}, R_{i-1}):
      - $L_i = R_{i-1}$ / $R_i = L_{i-1}$ XOR $F(R_{i-1}, K_i)$ where F is function computed by expand-shift-substitute-permute cycle and K is key.
      - rewrite as $R_{i-1} = L_i$ / $L_{i-1} = R_i$ XOR $F(L_i, K_i)$
  - 3DES: Triple-DES (still available as option)
    - 64 bits block, 168 bits key (option for 112 and 56) / 3 times slower than DES



    - C = $\mathbf{E}_{DES}(K_3, \mathbf{D}_{DES}(K_2, \mathbf{E}_{DES}(K_1, m)))$
      - 3 distinct keys give the strength of 112 bits key because M&H Double DES attack defeats the strength 1 of 3 keys (168-56)
    - C = $\mathbf{E}_{DES}(K_1, \mathbf{D}_{DES}(K_2, \mathbf{E}_{DES}(K_1, m)))$ [case $K_3 = K_1$]
      - strength of 80 bits key (I have no idea what this number comes from)
    - C = $\mathbf{E}_{DES}(K_1, \mathbf{D}_{DES}(K_2, \mathbf{E}_{DES}(K_1, m)))$ [case $K_3 = K_2 = K_1$]
      - same as single DES, as it uses the same key so this 3DES can be used to decrypt information that encrypted by single DES. because it used the same key in first encrypt and decrypt.
- AES: Advanced Encryption Standard (used in many products across networks)
  - 128 bits block, 128/192/256 bits key
  - used Substitution-permutation network

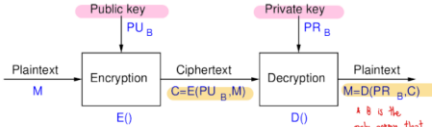
Principles of Public-Key cryptosystems

- Asymmetric algorithm in public key cryptography used one key for encryption and another but related key for decryption. Different from symmetric algorithms which used same secret key.
- Characteristics: Require a computational infeasible way to calculate decryption key given algorithm and encryption key only. Either of 2 used for encryption but other used for encryption.

Public and Private Keys

- Public key
    - Encryption for secrecy / Decryption for authentication
- Private key
    - Encryption for authentication / Decryption for secrecy
- Public-Private key pair for each user (PU, PR)

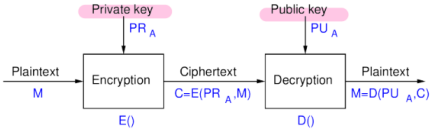
Confidentiality



(in this case, B = receiver)

- encrypt use PU / decrypt use PR / only person with PR can decrypt successfully

Authentication



- authentication is to verify that A is the one who really sends message to B
- encrypt use PR / decrypt use PU / only person with PR can encrypt → authenticity proven.

Analysis of Public Key cryptography

- Brute force? avoid by using large key, less efficient with larger key, mainly used for key management and digital signature.
- Compute PR from PU? no feasible methods in standard computing

PGP: Pretty Good Privacy

- Goals: uses both asymmetric and symmetric to protect messages on end-to-end
    - Confidentiality, Authentication, Integrity, Non-repudiation of origin
- PGP authentication of sender steps:
    - sender creates plaintext
    - sender creates SHA-1 hash of plaintext
    - sender encrypts SHA-1 hash using RSA with sender PR → create digital signature and appended to plaintext
    - receiver uses RSA with sender PR to decrypt the digital signature and recover SHA-1 hash
    - receiver generates SHA-1 hash from obtained plaintext then compare against decrypted SHA-1 hash from digital signature. If match, then it is authentic and accepted.
- PGP confidentiality of message steps:
    - sender generates random 128 bits session shared secret key (SSSK) of this message only
    - sender encrypts plaintext and append digital signature using sym. encryption with SSSK.
    - sender encrypts SSSK using RSA with each receivers' PU and append encrypted unique copy of SSSK to ciphertext (black-text message)
    - each receiver uses RSA with its PK to decrypt and recover their SSSK copy.
    - receiver uses decrypted SSSK decrypt the ciphertext and recover the plaintext.

Lecture 8 - Number Theory for Public Key

Divisors: b divides a / a is divisible by b → b | a → a = mb; b is divisor of a
Prime numbers: any integer p > 1 is prime if and only if its only divisors are ±1 and ±p
Greatest common divisor (gcd): c = gcd(a,b)
Relative prime numbers: two integer a and b are relative prime if gcd(a,b) = 1
Theory

- If a and b are integers, not both zero, then gcd(a,b) exists and unique
    - Euclidean GCD algorithm: gcd(a,b) = gcd(b,a%b) if b > 0; otherwise gcd(a,0) = a
- Linear combinations of a and b are multiples of gcd(a,b) and vice versa.
    - If gcd(a,b) = d then there exist integers x,y such that ax+by = d
    - If there exist integers x,y such ax+by = d, it's necessary and sufficient that gcd(a,b) | d.
- Extended Euclidean algorithm
    - find solutions of the equation of ax ≡ c mod b
    - Let d = gcd(a,b), then the equation ax+by = c has a solution if and only if d | c.

```
Solution of gcd(114, 42) : a = 114, b = 42
gcd(114, 42) = gcd(42, 114 % 42) = gcd(42, 30) | 114 = 2(42)+ 30  -> 30 = 114 - 2(42)
             = gcd(30, 42 % 30)  = gcd(30, 12) | 42 = 1(30)+ 12  -> 12 =  42 - 1(30)
             = gcd(12, 30 % 12)  = gcd(12, 6)  | 30 = 2(12)+ 6   ->  6 =  30 - 2(12)
             = gcd(6, 12 % 6)    = gcd(6, 0)   | 12 = 2(6) + 0   ->  0 =  12 - 2(6)
             = 6
find x and y that ax+by = gcd(a,b);
114(x)+42(y) = 6
           6 = 30 - 2(12)           = 30 - 2(42 - 1(30))
             = 3(30) - 2(42)        = 3(114 - 2(42)) - 2(42)
             = 3(114) - 8(42)       Therefore, gcd(114,42) = 6, x = 3, y = -8.
```

- Modular arithmetic
    - a = floor(a / n) * n + (a % n)
    - a ≡ b mod n if a mod n = b mod n
        - a ≡ b mod n if n | (a - b)
        - if a ≡ b mod n and b ≡ c mod n, then a ≡ c mod n
    - Modular arithmetic operations
        - [(a mod n) +/-/* (b mod n)] mod n = (a +/-/* b) mod n
        - there are commutative (สลับที่), associative (เปลี่ยนกลุ่ม), distributive (กระจาย) laws
        - Identities laws: (0 + a) mod n = a mod n; (1 * a) mod n = a mod n
    - Additive inverse (-a): for integer a, there is a unique integer b such that (a+b) ≡ 0 mod n
    - Multiplicative inverse of a: For a in Z_n, that is relative prime to n, there exists b in Z_n such that (a*b) ≡ 1 mod n
- In RSA key generation,
    - we want to compute d = e^{-1} mod ϕ(n) where n = p*q with p,q is two prime numbers.
    - Naïve approach

```
Data: p, q, e
Result: d such that de ≡ 1 mod ϕ(n)

for i in range(1, ϕ(n)) do
    if ie ≡ 1 mod ϕ(n) then
        d := i
        return d
```

    - Requires ϕ(n)/2 tests which is not computationally feasible when n is big

- Fermat's little theorem
    - If p is prime and a is a positive integer not divisible by p, then $a^{p-1} \equiv 1 \bmod p$
    - Totient function $\phi(n)$: the number of positive integers less than n and relative prime to n.
        - For prime number p, $\phi(p) = p-1$
        - For prime number p and q, $\phi(p*q) = \phi(p)*\phi(q) = (p-1)(q-1)$
- Euler's theorem
    - For every a and n that are relative prime, $a^{\phi(n)} \equiv 1 \bmod n$
- Primitive roots
    - k is the order of a modulo m if the k is the smallest positive integer such that $a^k \equiv 1 \bmod m$
    - There is $k>0$ such $a^k \equiv 1 \bmod m$ if and only if $\gcd(a,m) = 1$
    - If the order of b modulo m is k and $b^n \equiv 1 \bmod m$ then k | n.
    - g is called a primitive root modulo m if the order of g modulo m is $\phi(m)$
    - For a prime number p, there are $\phi(p-1)$ primitive roots modulo p
        - Example, p = 13, there are 4 primitive roots because $\phi(13-1) = 4$ → {1, 5, 7, 11} makes $\gcd(a,12)=1$
    - Example question: what is the order of a = 3 mod 10
        - $a^k \equiv 1 \bmod 10$
        - Solution
            - k = 1; $3^1 \equiv 1 \bmod 10$? No.
            - k = 2; $3^2 \equiv 1 \bmod 10$? No.
            - k = 3; $3^3 \equiv 1 \bmod 10$? No.
            - k = 4; $3^4 \equiv 1 \bmod 10$? Yes.
            - Therefore, order = 4
- Discrete logarithm
    - Find x such that $a^x \equiv n \bmod m$
    - It is the inverse operation of modular exponentiation.
    - No efficient classical algorithm is known for computing discrete logarithm.
    - It is become computational infeasible

Lecture 9 – Public Key Algorithm/RSA

- Public key algorithms rely on one key for encryption and a different but related key for decryption
- It is computationally infeasible to determine the decryption key given knowledge of algorithm and encryption key.
- In RSA, plaintext is encrypted in block with each block having a binary number less than some n.
- Operations
    - For some plaintext block M and ciphertext block C:
        - $C = M^e \bmod n$
        - $M = C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n$
    - Both sides know the value of n.
    - The sender knows e. → Public key is {e, n}
    - Only receiver knows d. → Private key is {d, n}

RSA Characteristics

- It is possible to find values of e, d, n such that $M^{ed} \equiv M \bmod n$ for all $M < n$
- It is relatively easy to calculate M^e and M^{ed} for all values of $M < n$
- It is **computationally infeasible** to determine d given e and n

Determining e, d, n

- RSA Core: Euler's theorem
    - Let p, q be two prime numbers, $n = p*q$ and $0 < m < n$, then $m^{k\phi(n)+1} \equiv m \bmod n$
    - ed could be determined by $ed = k\,\phi(n) + 1$ → $ed \equiv 1 \bmod \phi(n)$ or $d \equiv e^{-1} \bmod \phi(n)$
    - This is true if e and $\phi(n)$ are relative prime.

Basic steps in RSA algorithms

1. Privately choose two prime numbers, p and q, and keep it secret.
2. Calculate $n = p*q$, n is made public.
3. Choose $e < \phi(n)$, such that $\gcd(e, \phi(n)) = 1$, e is public.
4. Privately calculate $d \equiv e^{-1} \bmod \phi(n)$. d is secret.

Encryption and Decryption

- Both operations involve computing result of $a^m \bmod n$.
- The more efficient method using binary representation of m is as follows:

Algorithm 1: Compute $a^m \bmod n$.
Data: $b_k b_{k-1} \ldots b_0$, the binary representation of m
Result: $a^m \bmod n$

```
d := 1
for i := k downto 0 do
    d := (d * d) mod n
    if b_i ≠ 0 then
        d := (d * a) mod n
return d
```

Key Generation

- Determine two prime numbers, p and q.
- Selecting either e or d and then calculate the other one.

RSA Security

- The feasibility of computing one key given another key is computationally equivalent to that of factoring a number into its two prime factors.
- RSA is considered to be secure if n is the size between 1024 and 2048 bits, given current power.

RSA for bit sequences

- Encryption
    - Consider an RSA key private key PR = {e, n} = {77, 143} and
      16-bit sequence: `1011 0001 1010 1011`
      //spaced out for more readability
        1. Calculate block size of plaintext
           floor($\log_2(n)$) = floor($\log_2(143)$) = 7 bits/block
        2. Divide sequence into blocks
           `101 1000 | 110 1010 | 11`
        3. If the last block doesn't fill up, pad one and zeros to ensure it also has 7 bits.
           `101 1000 | 110 1010 | 111 0000` → 88 | 106 | 112
        4. Encrypt it with {e, n} using $C = M^e \bmod n$
           Block 1: 88 → $88^{77} \bmod 143 = 121$
           Block 2: 106 → $106^{77} \bmod 143 = 6$
           Block 3: 112 → $112^{77} \bmod 143 = 73$
           Ciphered number → 121 | 6 | 73
           Ciphered binary → `111 1001 | 000 0110 | 100 1001`
        5. Converted to 8-bits binary block sequences
           //Why 8? Ciphertext block size is plaintext block size + 1 (aka. floor($\log_2(n)$) + 1)
           `0111 1001 | 0000 0110 | 0100 1001` ← cipher bits sequence.

           --- Encryption is finished ---
- Decryption
    - Consider an RSA key public key PU = {d, n} = {53, 143} and
      24-bit sequence: `0111 1001 0000 0110 0100 1001`
      //spaced out for more readability
        1. Calculate block size of ciphertext
           floor($\log_2(n)$)+1 = floor($\log_2(143)$)+1 = 8 bits/block
        2. Divide sequence into blocks
           `0111 1001 | 0000 0110 | 0100 1001` → 121 | 6 | 73
        3. Decrypt it with {d, n} using $M = C^d \bmod n$
           Block 1: 121 → $121^{53} \bmod 143 = 88$
           Block 2: 6 → $6^{53} \bmod 143 = 106$
           Block 3: 73 → $73^{53} \bmod 143 = 112$
           plaintext number → 88 | 106 | 112
        4. Converted to 7-bits binary block sequences
           `101 1000 | 110 1010 | 111 0000`
        5. Remove padding bits if any (look at zeros and one)
           `101 1000 | 110 1010 | 111 0000`
           → Result is `1011 0001 1010 1011`
           //spaced out for more readability

           --- Decryption is finished ---

Lecture 10 – Authentication in Distribution System/Key Management

Authentication in Distributed System

- Each agent has his own private key and the public key of the other.
- Let K_A and K_B are public key (PU) of A and B, K_A^{-1} and K_B^{-1} are private key (PR) of A and B.
    - A sends message M to B by first signing M with PR of A (K_A^{-1}), then encrypting the signed message with PU of B (K_B)
      $A \rightarrow B : A, \left\{ \{M\}_{K_A^{-1}} \right\}_{K_B}$
    - On receiving encrypted message, B perform decrypting the message with PR of B (K_B^{-1}), then verify signature of it with PU of A (K_A)
      $\left\{ \left\{ \left\{ \{M\}_{K_A^{-1}} \right\}_{K_B} \right\}_{K_B^{-1}} \right\}_{K_A}$
    - In this way, no one apart from B can read message since only B has PR of B. If B reads the message, B knows that it sent by A since message must be encrypted with PU of A.
    - This protocol is secure when no one but B could know about M if A does not tell M to anybody else.
    - However, message can be lost on internet, so A is not sure whether B has received its message. So, the protocol could be extended as
      1. $A \rightarrow B : A, \left\{ \{M\}_{K_A^{-1}} \right\}_{K_B}$
         M signed with PR of A then encrypt with PU of B
      2. $B \rightarrow A : \left\{ \{M\}_{K_B^{-1}} \right\}_{K_A}$
         M signed with PR of B then encrypt with PU of A
         - On obtaining M,
         - B performs the following operation on the message received in step 1:
           $\left\{ \left\{ \left\{ \{M\}_{K_A^{-1}} \right\}_{K_B} \right\}_{K_B^{-1}} \right\}_{K_A}$
         - On receiving in step 2, A performs this operation to determine if B has received M.
           $\left\{ \left\{ \left\{ \{M\}_{K_B^{-1}} \right\}_{K_A} \right\}_{K_A^{-1}} \right\}_{K_B}$
    - Is it secure?/Is there a way for R (outsider) to know about M between A and B? → Yes.
        - R could do:
            - A sends message to B
              $A \rightarrow B : A, \left\{ \{M\}_{K_A^{-1}} \right\}_{K_B}$
            - Then, R intercept and keep the message part and send
              $R \rightarrow B : R, \left\{ \{M\}_{K_A^{-1}} \right\}_{K_B}$
            - When B gets the message, it will decrypt the second component with K_B^{-1}. Then, with PU of R, B signs the result with his private key and encrypts with PU of R. Thus,
              $B \rightarrow R : \left\{ \left\{ \left\{ \{M\}_{K_A^{-1}} \right\}_{K_R} \right\}_{K_B^{-1}} \right\}_{K_R}$
            - R can now gain the knowledge of M by decrypting the obtained message in this order: K_R^{-1}, K_B, K_R^{-1}, K_A

- $R \rightarrow A : \left\{ \{M\}_{K_B^{-1}} \right\}_{K_A}$
  Note that R cannot pretend to be B to send message to A, because R does not know about PR of B. Therefore, A shall believe that his message might have failed to reach B.

Public Key Exchange

- In PU cryptography, the most important is the integrity of public keys.
- For example, A wants to talk to B, but R manage to give A his PU and A believes this is PU of B. Now, when A wants to talk to "B", R could always pretend to be B and read all communications since A believes PU of R is PU of B.
- Then, How can A obtain the PU of B first? → Trusted key distribution centre S which in charge of distribution of Pus.
  - Systems works as follows

    $1. A \rightarrow S : A, B, Na$

    $2. S \rightarrow A : S, \{S, A, Na, K_B\}_{K_S^{-1}}$
  - Explanation: When A wants to talk to B, sends plaintext to PU server S ask for PU of B. Then, In message from A to S, a random string nonce (Na) is included to indicate freshness, means that Na has just been generated by A and never been generated before. When A get a message signed with PR of S, A knows that it is sent by S. Since the message contains fresh random Na. A knows that S has got Na and just generated the replying message.
  - It is relied on assumptions of: A knows the PU of S, and S knows the PU of everyone.

Key Hierarchy and Lifetimes

- Master keys used to exchange session keys, often asymmetric keys like RSA. It is changed not so often.
- Session keys used to exchange data, often symmetric keys like AES or DES. It is changed automatically and regularly.
- Lifetime of session keys
  - Shorter → more secure / increase exchanges overhead
  - TCP/Connection-oriented: new session key for each connection.
  - UDP/Connection-less: new session key after fixed period or certain number of packets.
- Approaches to exchange session keys:
  - Option 1:
    - Assumption: entities can obtain correct PU of others.
    - Process: random session key is generated when A talks to B, then encrypts this key with PU of B and sends to B.
    - Pros and Cons: Simple, too strong assumptions, practically difficult to achieve
  - Option 2:
    - Assumption: entities can exchange master keys with trusted central entity/Key distribution centre (KDC).
    - Process: KDC automatically exchanges session keys between users.
    - Pros and Cons: Each entity doesn't need to know the correct PU of others, but security and performance at KDC.
  - Option 3:

- End-to-end protocols for session keys exchanges.
- **Diffie-Hellman key exchange protocol**
- **Oakley key determination protocol** (improve shortcoming of Diffie-Hellman)

Session Key Exchange

- Protocol for distribution of a fresh session key K generated by KDC S, to users A and B.
  1. A request S to generate new K for talking with B
  2. S reply to A, send newly generated K
  3. S inform B about generated K by S between A and B
  4. B request A to confirm K is a fresh session key between A and B generated by S
  5. A reply to B to confirm K is a fresh session key between A and B generated by S
  6. B inform A to confirm that B know that K is fresh session key between A and B generated by S

  At the end, both A and B are expected to get K and know that other side got K as well.
- Implementation level: concrete messages exchanged between parties as follows:

1 : $A \rightarrow S$: $req, newkey, A, Init, S, Server, N_a, B$
That is, A sends to S a message with:
- keywords $req, newkey$ indicating that this message represents a request for new key.
- keywords $Init, Server$ indicating the roles of A, S
- N_a: a fresh nonce

2 : $S \rightarrow A : \{rep, newkey, S, Server, A, Init, N_a, K, B\}_{K_{AS}}$
That is, S replies by sending a newly generated session key K to A. This reply message contains:
- keywords $rep, newky$ indicating that this is a reply to an earlier request for a new key.
- Nonce N_a so that after getting this reply, A knows that K is a fresh session key generated by S.

3 : $S \rightarrow B$: $\{inf, newkey, S, Server, B, Resp, K, B, A, S\}_{K_{BS}}$
S informs B (playing the role of a Responder) about key K.
- Keywords $inf, newkey$ indicate that the message is intended to inform B about a new key.
- After receiving the message, B could only say that it has been informed about K without being sure whether K is fresh or not. This is because B has no knowledge about K before receiving the message, and hence it could not verify whether this message is just sent recently or it has been replayed by some malicious user.

To verify the information it has just obtained, B requests A:

4 : $B \rightarrow A$: $\{req, keyconfim, B, Resp, A, Init, N_b, Hash(K), S\}_{PU_A}$
In this message, keywords $req, keyconfirm$ indicate that the message represents a request for confirming the status of the included key. N_b is a fresh nonce.

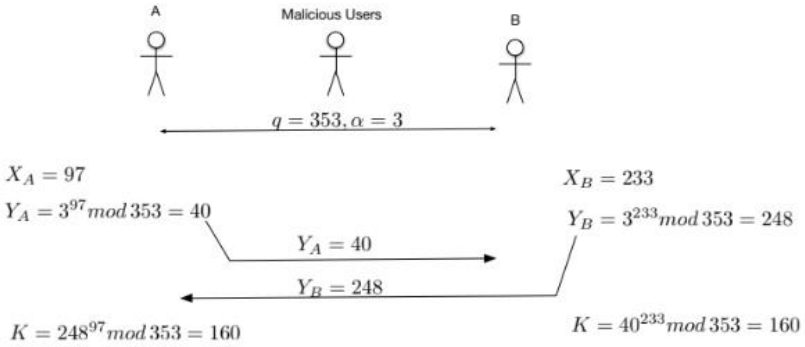
5 : $A \rightarrow B$: $\{rep, keyconfirm, A, Init, B, Resp, N_b\}_{PU_B}$
That is, A replies B by sending a message to confirm that K is indeed a fresh session key between A and B. After getting this message, B knows that what it has been informed before is correct.

6 : $B \rightarrow A$: $\{inf, keyconfirm, B, Resp, A, Init, S\}_K$
B sends A the above message to confirm that B now knows that K is a secrete session key between A and B. The keywords $inform, keyconfirm$ indicate the intention of the message is for B, to confirm to A that it knows about K.

Diffie-Hellman Key Exchange

- Protocol for two parties to exchange a session key. This protocol forms the foundation of the key management in IP and web security.
- Used maths of
    - For every a and n that are relative prime, $a^{\phi(n)} \equiv 1 \bmod n$
    - In general, for any two positive integer, a and n, the smallest positive number m such that: $a^m \equiv 1 \bmod n$, called the order of a (modulo n).
    - If the order of a (modulo n) is $\phi(n)$ and $a < n$, then a is primitive root of n.
    - For a prime number p, there are $\phi(p-1)$ primitive roots modulo p.
- Process of Diffie-Hellman key exchange

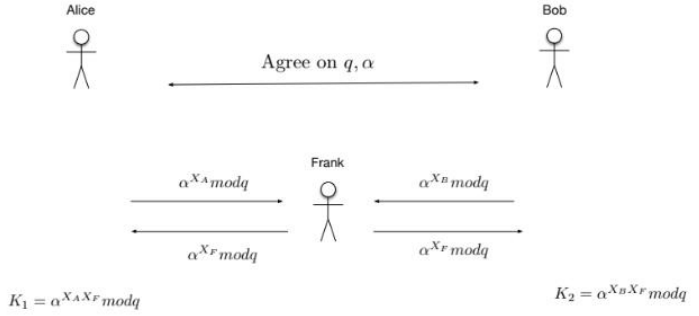
1. **Global Public Elements**
   The protocol assumes that two global public numbers are known:
   - q: a prime number
   - α: $\alpha < q$ and α is a primitive root of q
2. **Key generation of A**
   - Select private X_A such that $X_A < q$
   - Calculate public Y_A by $Y_A = \alpha^{X_A} mod\ q$
3. **Key generation of B**
   - Select private X_B such that $X_B < q$
   - Calculate public Y_B by $Y_B = \alpha^{X_B} mod\ q$
4. **Generation of Secret Key by A**
   $$K = Y_B^{X_A} mod\ q$$
5. **Generation of Secret Key by B**
   $$K = Y_A^{X_B} mod\ q$$



- A and B obtain the same K because:

$$Y_B^{X_A} mod\ q = ((\alpha^{X_B} mod\ q)^{X_A}) mod\ q = \alpha^{X_A X_B} mod\ q =$$
$$((\alpha^{X_A} mod\ q)^{X_B}) mod\ q = Y_A^{X_B} mod\ q$$

- If attackers can compute the discrete logarithms to obtain X_A and X_b from Y_A and Y_B. Then, he can learn about K. Hence, prime number q needs to be very **big** number.
- Weaknesses: No information about identities of parties, subjected to man-in-the-middle attack.
- Man-in-the-middle attack on DH protocol:
    - DH doesn't provide source authentication because man in the middle could pretend to be B to send Y_B to A. So, A established key with man in the middle but thinks does it with B.



Oakley Key Determination Protocol

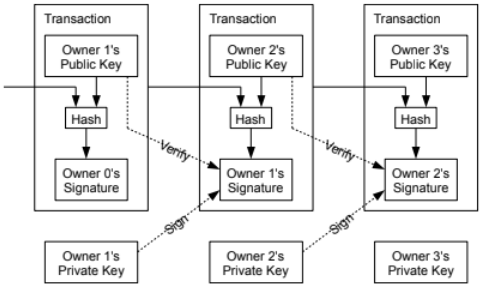
- Improvements of Oakley over Diffie-Hellman:
    - Enables 2 parties to negotiate the global parameters called group
    - Nonce are used against replay attack
    - Prevents man-in-the-middle attack
- However, Oakley requires parties to know PU of others.

Message Authentication

- It is a procedure to ensure that messages come from the intended source and not modified.
- Needs to counter these attacks:
    - Masquerade: Insertion of message to network by someone who present itself as authorized entity.
    - Modification:
        - Content modification: change message contents.
        - Sequence modification: change message sequences in protocols.
        - Timing modification: delay or replay of message.
- Authentication methods:
    - Message encryption: ciphertext of messages as authenticator.
        - M from A to B is encrypted using secret K, therefore $\{M\}_K$ arrives at B, B knows that A is the only one who encrypted it.
        - Conventional encryption? No. If bad guy C sends B an arbitrary message X, B decrypts X using K, resulting in $\{X\}_K$ that might be gibberish. If A and B are agents with no understanding of common language, then B will not realize that $\{X\}_K$ is meaningless. This also happened when public key cryptography is used.
    - MAC/Message Authentication Code: A public function of a message and secret key to produces a fixed length value as authenticator.
        - A, B assumed to share K. When A sends M to B, it calculated MAC.
        - MAC = $C_K(M)$; $C_K()$ is a hash function map message to any fixed length message.
        - Then A sends to B: M, MAC
        - When B receives: X, V
            - B needs to calculate $C_k(X)$ and compares to V.
            - If it matches, B can summarize X = M and V = MAC.
        - Because unauthorized parties don't know about K, so they cannot calculate $C_k(M')$ if they modify M to M'

Representation : Bit Coin

- Electronic coins as a chain of digital signatures.
- Each owner transfers the coin to others by creating hash of previous transaction's digital signature and the public key of the next owner adding these to the end of the coin.

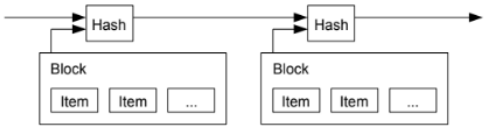


- T_0: owner O_0 owns the coin because he receives from ... or mines himself ...
- $T_1 = \{H(T_0||K_{O_1})\}_{K_{O_0}^{-1}}$ representing that O_0 transfers the coin to O_1 who hence owns the coin.
- $T_2 = \{H(T_1||K_{O_2})\}_{K_{O_1}^{-1}}$ representing that O_1 transfers the coin to O_2 who hence owns the coin.
- $T_3 = \{H(T_2||K_{O_3})\}_{K_{O_2}^{-1}}$ representing that O_2 transfers the coin to O_3 who hence owns the coin.

A payee can verify the signatures to verify the chain of ownership. Unfortunately he can't verify that one of the owners in the chain did not double-spend the coin. For example, O_2 may have transferred the above coin to O_3' before transaction T_3, by

$$\{H(T_2||K_{O_3'})\}_{K_{O_2}^{-1}}$$

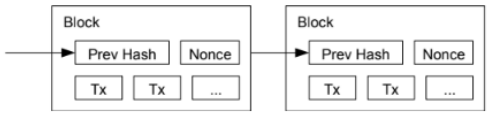
- Proof of work
  - Timestamp server
    - **Timestamp server**: consider a sequence of (data) items $d_0, d_1, \ldots, d_n, \ldots$, and let
      - block $b_0 = d_0$
      - block $b_1 = d_0||d_1$
      - ...
      - block $b_n = d_0||d_1||\ldots||d_n$



Timestamp server construct the timestamp h_n for block b_n as follows

$h_0 = H(b_0)$
$h_1 = H(h_0||b_1)$
$h_2 = H(h_1||b_2)$
...
- $h_n = H(h_{n-1}||b_n)$

  - Proof of work
    - **Proof of work**:



Suppose that $T_0, \ldots, T_{n-1}$ are verified transactions (none of these owners double-spent) and T_n is the new transaction that needs to be verified. A proof of work for T_n is such a Nonce that the timestamp of block $Nonce||T_0||T_1||\ldots||T_n$ begins with a required number of zero bits.

Lecture 11 –Network Layer Security

****NOTED THAT IT MIGHT NOT BE IN THE FINALS****

OSI Internet Model

- Network/Internet layer handles the interconnection of networks. In the internet model, it uses IP protocols (IPv4/v6) to address and route between devices throughout the network. It could be end devices or intermediate devices.

IP Security Deployment

- IPsec used to securely connect: LANs over untrusted intermediate networks, and a single computer connected into two LANs over the untrusted interconnecting network.
- Network layer security is closely related to protocol nature being used at network layer and synonymous with IP security.
- Benefits of IPsec:
    - Generality: IPsec can be used for entire network, transparent to applications since IPsec is at IP level below transport layer protocols.
    - Convenience: IPsec has IKE which is a protocol for automatic key management. So, users do not need to pay attention to application network's security issues, and system admin don't need to handle key management.
- 3 Functional areas of IPsec:
    - Authentication: peer-entity and data-origin authentication (using PK)
    - Data Integrity via MAC (optional and confidentiality): (using symmetric encryption)
    - Key Management: negotiation of session key and related parameters (using DH variants)
- Components of IPsec architecture which provides functional areas:
    - Key management protocol: **IKE**
    - Two transforms applied to IP packets to be protected
        - **AH/Authentication header**: protects authenticity and integrity. No confidentiality protection.
        - **ESP/Encapsulating security payload**: provides confidentiality, but probably integrity and authenticity.
    - Two modes:
        - **Transport:** IPsec security transform (AH or ESP) is applied directly to IP packets to be protected.
        - **Tunnel:** IP packets to be protected is placed within an outer IP packet as PDU, then IPsec security protocol is applied to inner IP packets.
        - 4 combinations

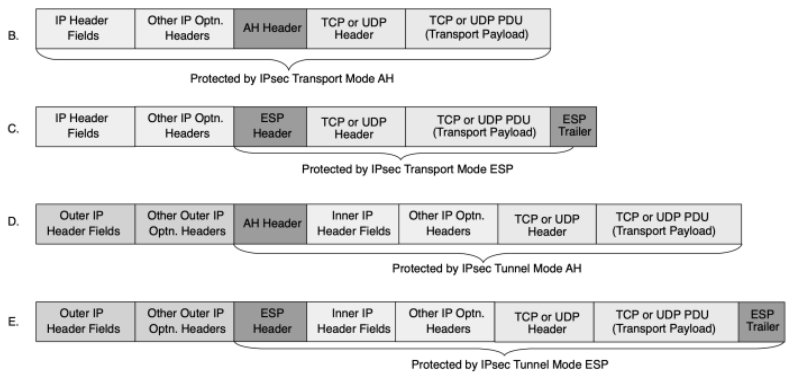


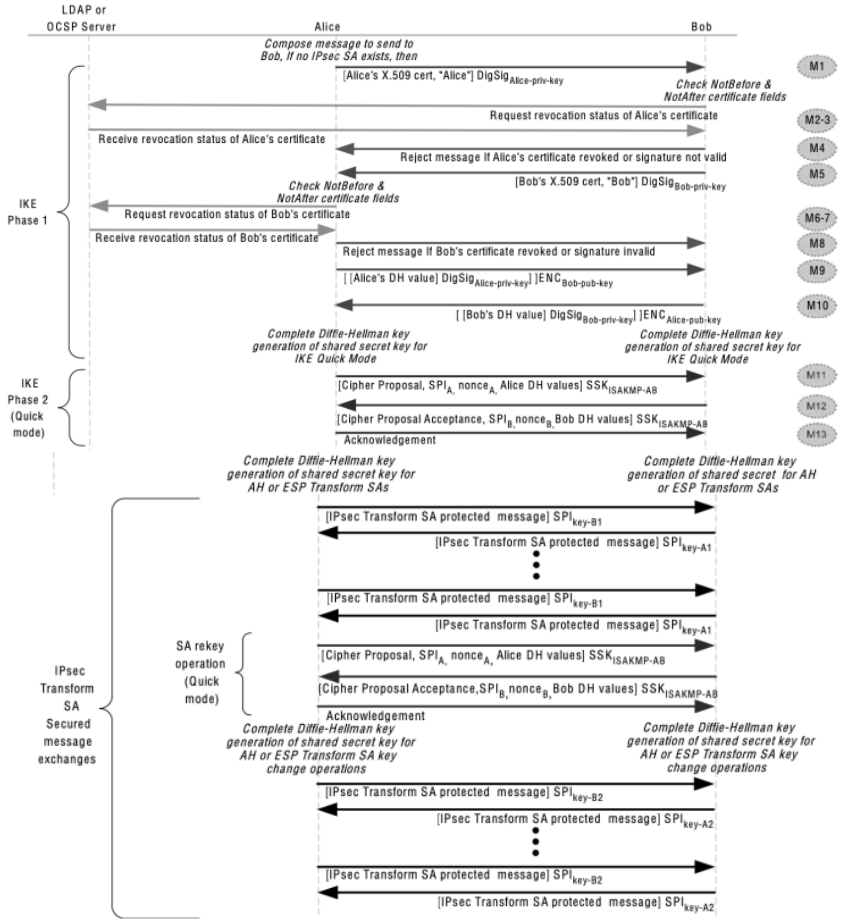
Table 10.2. IPsec modes and transforms compared

Transform	Mode	
	Transport	Tunnel
AH	Authenticates IP payload and selected portions of IP header (and IPv6 extension headers)	Authenticates entire inner IP packet plus selected portions of outer IP header
ESP with encryption (ESP-3DES or ESP-AES)	Encrypts IP payload	Encrypts inner IP packet(and any IPv6 extension headers)
ESP without encryption (ESP-nul)	Authenticates IP payload	Authenticates inner IP packet

- Typical usage
    - IPsec transport mode is used by machines that are the endpoints of communications which are the acutal devices establishing the secure channel.
    - Tunnel mode protects the entire original IP packet and is commonly implemented by IPsec gateways, such as firewalls or routers. So, the secure channel is established between gateways and not by end machines.

IKE: Key Management

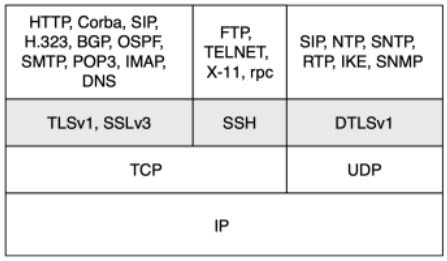
- IKE borrows ideas from Oakley's protocol which is a variant to DH (Diffle-Hellman) protocol to counter man-in-the-middle attacks.
- Security Association (SA): SA defines the security mechanisms algorithms for MAC, cryptographic keys, authenticated data size, that will be employed when sender sends data to receiver. It is one-way. If want to communicate two ways, two SAs are needed.
- IKE provides automated shared secret-key management capabilities, supporting these IPsec SAs.
- IKE message flow (not going to talk about this).



Lecture 12 –Transport Layer Security

- Transport layer facilitates data exchange between application on both end machines. This layer uses two protocols, either TCP or UDP.
    - TCP provides connection-oriented, reliable connection, and error checking.
    - UDP is connectionless, unreliable connection, not guaranteed delivery.

Existing Transport Security Protocols



- Used over TCP
    - TLSv1.2: Transport Layer Security protocol
    - SSLv3: Secure Sockets Layer protocol
    - SSH: Secure Shell protocol
- Used over UDP
    - DTLSv1: Datagram Transport Layer Security protocol
- From application layer:
    - SMTP/POP3/IMAP (email protocols) are now used over TLS rather than TCP.
    - E-commerce HTTP request/response (WWW traffic) used over TLS to form HTTPS.
    - TLS can also use to create a VPN.

Two layers of SSL/TLS

- Lower layer protocol is SSL/TLS record protocol, responsible for establishing a secure and reliable channel between machines from insecure TCP connections.
    - TLSv1 record protocol provides reliable channel to upper layer protocols.
    - Provides security components:
        - Data-origin authentication and Data integrity: using MAC with shared secret key.
        - Confidentiality: using symmetric encryption algorithms.
- Upper layer protocols consist of: Handshake, Change Cipher Spec, Alert protocol, Application protocol such as XHTTP, SOAP.

Session Establishment

- SSL/TLS is a session-based connection-oriented protocol. Connection is established for each session and created by handshake protocol.
- During the process, encryption algorithms, keys, necessary digital certificates are specified.
- Having a session is nice because when there is more than one connection carried, it prevents repeated establishment of channels and handshake procedure.
- Each connection is distinguished by shared secret keys set and sequence numbers. The shared secret keys for multiple connections have the same origin, derived from a unique master shared secret key created during the initial handshake protocol.

SSLv3.0 handshake protocol (current form) and its version history – look at the lecture.

HTTP

- Client-server application protocol operates between user client software referred as user agent (UA) and server system.
- UA initiates an HTTP request; it establishes TCP connection to a specific TCP port on the server (80 by default). Server listens on that port, waits for UAs to send a request.
- Upon the receipt of requests, the server sends back a status line and a body message which is requested resource, error message, or other information.

HTTPS

- HTTP over SSL = HTTPS
- Requested resources via HTTP are identified using URIs (Uniform Resource Identifiers), specifically URLs (Uniform Resource Locators), using http URI schemes.
- HTTPS is HTTP carried over SSL. The https scheme is a URI scheme which syntax is identical to basic http scheme (used for normal http connection) that signals the UA to use SSL to protect traffics exchanged between the server and the UA.
- HTTPS Features
    - URL uses https://
    - Listen on port 443
    - Encryption: URL of requested documents, document contents, browser forms contents, cookies, HTTP header contents.
    - Authentication of server using digital certificates.
    - Authentication of client using password.

Good luck.