UNIT II

Block Ciphers & Public Key Cryptography

Data Encryption Standard (DES) - 2
Block Cipher Principles - 6
Block Cipher Modes of Opn - 6
Advanced Encryption Standard (AES) - 12
Triple DES - 17
Blowfish - 19
RSA Algorithm - 20
Public Key Cryptography - 23
Principles of Public Key Crypto Systems - 26
The RSA Algorithm - 27
Key Management - 32
Diffie-Hellman Key Exchange - 36
Elliptic Curve Arithmetic - 40
Elliptic Curve Cryptography - 41
UNIT II

Block Ciphers

Data Encryption Standard (DES):-

- Proposed by NIST adopted in 1975.
- It is a block cipher that encrypts 64-bit data using 56-bit key.

**DES Encryption:**

64-bit plaintext

Initial Permutation

\[ 64 \]

- Round 1: $K_1$

\[ 48 \]

- Round 2: $K_2$

\[ 32 \text{-bit swap} \]

Inverse Initial Permutation

64-bit ciphertext

- Initial Permutation (IP): 1st step of the data computation.

* IP rearranges the 1st data bits.

www.studentsfocus.com
\[
\text{Ex: } IP((675a\ 69\ b7 \ 5e\ 5a\ b6\ b5a) \oplus \\
= (ff\ b6\ a9 \text{ and } 004\ df6\ fb)
\]

**DES Round Structure:**
- Uses two 32-bit L & R halves.
- Feistel Cipher,
  \[L_i = R_{i-1}\]
  \[R_i = L_{i-1} \oplus F(R_{i-1}, k_i)\]
- \(F \) takes 32-bit R half & 48-bit subkey.
  \(\rightarrow\) Expands R to 48-bits using Perm E
  \(\rightarrow\) Adds to subkey using XOR
  \(\rightarrow\) Passes through 8 S-boxes to get 32-bit result
  \(\rightarrow\) Finally permutes using 32-bit Perm P.
**Substitution Boxes 8**: 
- Each of the eight 8-boxes is different.
- Each 8-box reduces 6 bits to 4 bits.
- So, the 8 8-boxes implement the 48-bit to 32-bit confusion substitution.

**DES key schedule**:
- Forms subkeys used in each round.
  - Initial permutation of the key (pc1) which select 56-bits in two 28-bit halves.
  - 16 stages consisting of:
    - rotating each half separately either 10 or 2 places depending on the key rotation schedule.
    - selecting 24-bits from each half & permuting them by pc2 for use in round fnE.

**Decryption**: reverse order (s1 k16 ... s1 k1).

**Avalanche effect**:
- Key desirable property of encryption alg.
- Where a change of one 8-bit (or) key bit results in changing approx half 8-bit bits.
- Malung attempts to "home-in" by guessing key components.
Strength of DES - key size.

- 56-bit keys have $2^{56} = 7.2 \times 10^{16}$ values.
- brute force search looked hard.

Analytic Attacks:
- differential cryptanalysis
- linear cryptanalysis
- related key attacks

Block Cipher Principles:

   - Number of Rounds:
     - More is better, Exhaustive search best attack.
   - Function $f$:
     - provides "confusion" is non-linear, avalanche.
   - have issues of how S-boxes are selected.
2. key schedule:
   - complex subkey creation, key avalanche..
**Block Cipher Modes of Operation:**

1. Block ciphers encrypt fixed size blocks.
   - Example: DES encrypts 64-bit blocks.
2. NIST SP 800-38A defines 5 modes:
   - 4 block & stream modes
   - Modes of Operation:
     - Electronic code book (ECB)
     - Cipher Block chaining (CBC)
     - Cipher Feedback (CFB)
     - Output Feedback (OFB)
     - Counter (CTR)

Electronic code book (ECB):

- Message is broken into independent blocks that are encrypted.
- Each block is a value which is substituted like a codebook, hence the name.
- Each block is encoded independently of the other blocks. \( C_i = E_k (P_i) \).
- Uses: Secure transmission of single values.

Decryption: \( P_i = D_k (C_i) \)
Advantages & Limitations of ECB:

- Msg repetitions may show in ciphertext.
- If aligned with msg blk.
- Partially with data such graphs
- For with msg that change very little,
  which become a code-book analysis pbm.

* Weakness is data independent.
* Not vulnerable to cut-and paste attack.
* Main use is sending a few blls of data.

Ciphers Block chaining (CBC):

* Msg is broken into bllks.
* Each previous cipher block is chained
  with current plaintext blk.

\[ C_i = E_k (P_i \oplus C_{i-1}) \]

\[ C_1 = IV \]

Uses:
- Bulk data encryption, authentication.

Diagram:

- Encryption: IV \[ P_i \]
  \[ \downarrow \]
  \[ \oplus \]
  \[ K \]
  \[ \text{Encrypt} \]
  \[ C_i \]

- Decrypt: \[ K \]
  \[ \text{Decrypt} \]
  \[ \oplus \]
  \[ K \]
  \[ \downarrow \]
  \[ \text{IV} \]
  \[ P_i \]

www.studentsfocus.com
Adv & dis adv:

* A cipher text blk depends on all blks before it. Any change to a blk affects all following cipher text blks... avalanche effect.

Disadv:

* Need Initialization Vector (IV).

Stream modes of opn:

* Blk modes encrypt entire blk.

* May need to operate on smaller units.

* Real-time data.

* Convert blk cipher into stream cipher.

* Cipher Feedback (CFB) mode.

* Output "OFB".

* Counter (CTR).

* Use blk cipher as some form of pseudorandom number generator... Vernam cipher.

Cipher Feedback (CFB):

* Key is treated as a stream of bits.

* Added to the OF of the blk cipher.

* Side 1 1 1 1 F B ( OF ) 54 ( OF ) 1 2 8 J.
FB is independent of msg
uses: Stream encryption on noisy channels.
why noisy channels?

Advantages:
- Needs an IV which is unique for each use.
- If ever re-use attacks can recover IV.
- OTP
- Can pre-compute
- Bit errors do not propagate
- More vulnerable to msg stream modification
- Change arbitrary bits by changing ciphertext.
- Sender & receiver must remain in sync
- Only use with full offline FB.

www.studentsfocus.com
Counters (CTR):

- a "new" mode, though proposed early on
- IIAI to OFB but encrypt counter value
  rather than any fb value.

\[ C_i^0 = P_i \oplus C_i^0 \]

- must have a diff key & counter value
  for every plaintext block (never reused)

\[ \text{OTP usage} \]

- uses high-speed NEW encryption.

**Encryption:**

\[ k \rightarrow \text{Encrypt} \rightarrow \text{ciphertext} \]

\[ P_i \rightarrow k \rightarrow \text{Encrypt} \rightarrow c_i \]

**Advantages & Limitations:**

- Efficiency: can do parallel encryption in the
  form

\[ (a) \text{can} \quad (b) \text{can\ preprocess\ in\ advance\ ahead.} \]
1. AES differs proposed by Rijmen - Denne

x, Can use Triple-DEE - but slow

- Have demonstrated exhaustive key

x, Break it

- Have "practical" attacks that can

- A replacement for DES was needed

AES appeared

Advanced

Cipher (AES)


- Valuable alternative towards break (cf. Rijndael)

x, Must ensure never reuse key

x, Never have more than 26

- Provide security (good as other modes)

1. Random access to encrypted data bits

12. Good for bursty high-speed links
AES Encryption Process:

Plaintext - 16 bytes (128 bits)

Key (16 bytes)

Initial transformation

Round 1 (4 transformations)

Round N-1 (1 transformation)

Round N (3 transformations)

Ciphertext - 16 bytes

Key Expansion

No. of Rounds

Key Length (bytes)

1, 10, 24, 32

Resistance

Speed & code compactness

Design simplicity
AES Structure:

1. Data block of 1 col of 4 bytes is state
2. Key is expanded to array of words
3. Has 9/11/13 rounds in which states:
   - byte substitution (1 8-box used on every byte)
   - shift rows (permute bytes into groups of cols)
   - mix columns (substitute using matrix multiply of groups)
   - add round key (XOR state with key material)
   - view as alternating XOR key & scramble data bytes.

Comments on AES:
1. An iterative rather than Feistel cipher.
2. Key expanded into array of 32-bit words.
3. Four words from round key in each round.
4. Four different stages as shown.
5. Has a simple structure.
6. Only AddRoundKey uses key.
7. AddRoundKey a form of Vigenère cipher.
8. Each stage is easily reversible.
9. Decryption uses keys in reverse order.
10. Each round has only 3 stages.
Aes arithmetic:

* Uses arithmetic in the finite field

\[ GF(2^8) \] with irreducible polynomial

\[ m(x) = x^8 + x^4 + x^3 + x + 1 \]

which is \((100011011)\) or \(\xi_{11} b\).

\[ \text{Ex:} \]

\[ \{02\} \cdot \{84\} \mod \{11\} b = (100001110) \mod \{11\} b \]

\[ = (100001110) \ XOR (10010111) \]

\[ = (00010101) \]

AES key expansion:

* Takes 128-bit (16-byte) key & expands into 82-bit words.

1. Start by copying key into first 4 words.
2. Then loop creating words that depend on values in previous 4 places back.
3. In 3 of 4 cases just XOR these together.
4. First word in \(t\) has rotate + S-box + XOR round constant on previous, before XOR \(t^{th}\) bit back.
Implementation Aspects:
* Can efficiently implement on 8-bit CPU.
  > Byte substitution works on bytes using a table of 256 entries
  > Shift rows is simple byte shift
  > Add round key works on byte XOR's
  > Mix columns requires matrix multiply in $GF(2^8)$ which works on byte values, can be simplified to use table lookups & byte XOR's.

* Designers believe this very efficient implementation was a key factor in its relation as the AES cipher.
Triple DES:

Multiple Encryption & DES:

* Clear a replacement for DES was needed
  > theological attacks that can break it
  > demonstrated exhaustive key search attacks.

* AES is a new cipher alternative.

* Prior to this alternative was to use multiple
  Encryption with DES implementations.

* Triple-DES is the chosen form.

Why not Double-DES?

* Could use 2 DES encrypt each blk.
  > \( C = E_{k2}(E_{k1}(P)) \).

* Concern at time of reduction to single stage.
  > Heek in the middle attack.

* Heek in the middle attack.
  > works whenever use a cipher twice.
  > \( X = E_{k1}(P) = D_{k2}(C) \).

* attack by encrypting \( P \) with all key

* then decrypt \( C \) with keys & match
can show takes $O(2^{56})$ steps.

Requirements:

- Requires known plain text.

**Triple-DES with Two Keys:**

- Hence must use 3 encryptions—need 3 distinct keys.
- But can use 2 keys with E-D-E sequence:
  - $C = E_{k1}(D_{k2}(E_{k1}(P)))$.

- Encrypt & decrypt equivalent in security.
- If $k_1 = k_2$, then can work with Single DES.


- No current known practical attacks.
- Several proposed impractical attacks might become basis of future attacks.

**Triple-DES with Three Keys:**

- Although no practical attacks no two-key
- Triple-DES have some indications.
- Can use Triple-DES with three keys to avoid even those:
  - $C = E_{k3}(D_{k2}(E_{k1}(P)))$.

- Has been adopted by some Internet applns.
Blowfish:

1. A symmetric block cipher Blowfish.

Characteristics:
- Fast implementation on 32-bit CPUs.
- Compact in use of memory.
- Simple structure for analysis.
- Variable security by varying key size.
- Uses a 32 to 448 bit key.

Key schedule consists of:
1. Initialize P-away & then 4 8-boxes using P.
2. XOR P-away with key bits (reuse as needed).
3. Loop repeatedly encrypting data using current P & S places.
   Replace successive pairs of P then S values.
4. Reverses 321 encryptions, hence slow in re-keying.
5. Uses 8 primitives: addition & XOR.
   Data bytes divided into two 32-bit halves Lo & Ro.
for $i = 1$ to 16 do
    $R_i = L_{i-1} \ XOR \ P_i$;
    $L_i = F(R_{i-1} \ XOR \ R_{i-1})$;

Ex:
    $L_{17} = R_{16} \ XOR \ P_{18}$;
    $R_{19} = L_{16} \ XOR \ P_{14}$;

where:

$$F(a,b,c,d) = ((s_1 \ a + s_2 \ b) \ XOR \ s_3, c) + s_4, a$$

- key dependent 8 boxes and subkeys makes cryptanalysis very difficult.
- Changing both halves in each round increases security.
- Provided key is large enough, brute-force key search is not practical, especially given the high-key schedule cost.

**RC5 algorithm**

- A proprietary cipher owned by RSA Data.
- Designed by Ronald Rivest (of RSA).
- Used in various RSA Data products.
- Can vary key size/data size/number of rounds.
- Very clean, simple, design, easy to implement on various CPUs.
RC5 is a family of ciphers RC5-w/l/b.

- $w$ = word size in bits (16, 32, 64)
- no data $= 2w$
- $r$ = no. of rounds (0, 655)
- $b$ = no. of bytes in key (0, 255)

Normal version is RC5-32/12/16.

We $= 32$-bit words so encrypts $64$-bit data blocks using $12$ rounds with $16$ bytes ($128$ bits) secret key.

RFC 2040 defines 4 modes used by RC5:

- RC5-Block cipher is ECB mode
- RC5-CBC is CBC mode
- RC5-CBC-PAD is CBC with padding by bytes with value being the no. of padding bytes
- RC5-CTS, a variant of CBC which is the same size as the original msg, uses ciphersalt stealing to keep size same as original

RC5 key expansion and encryption:

- RC5 $\text{key expansion}$ and encryption
- RC5 uses $2r+2$ subkey words ($w$-bits)
- subkeys are stored in array $S[0], \ldots, S[r]$
key schedule consists of

- Initializing \( S \) to a fixed pseudorandom value, based on constants \( e \) and \( \phi \).
- The byte key \( k \) is copied (little-endian) into a c-word away \( 2 \).
- A mixing opn then combines \( k \) and \( S \) to form the final \( S \) away.

- Split \( k \) into two halves \( A \) & \( B \):
  
  \[
  \begin{align*}
  L_0 &= A + S[0] \\
  R_0 &= B + S[1] \\
  \end{align*}
  \]

- For \( i = 1 \) to \( n \) do
  
  \[
  \begin{align*}
  L_i &= (L_{i-1} \oplus R_{i-1} \ll \cdot R_{i-1}) + S[2 \cdot i] \\
  R_i &= (R_{i-1} \oplus L_{i-1} \ll \cdot L_{i-1}) + S[3 \cdot i+1] \\
  \end{align*}
  \]

- Each round is like 2DES rounds.
- Note rotation is main source of non-linearity.
- Need reasonable no. of rounds (32: 12-16)
Public key cryptography:

1. **Key Distribution**
   - Developed to address two key issues:
     - How to have secure communication without having to trust a KDC with your key.

2. **Digital Signatures**
   - How to verify a msg comes intact from the claimed sender.

Public invention due to Whitfield Diffie & Martin Hellman at Stanford in 1976.

Public key cryptography:

- **Public Key / Two-key / Asymmetric Crypt**
  - Involves the use of two keys:
    - Public key, which may be known by anybody, and can be used to encrypt msgs.
    - Private key, known only to the recipient, used to decrypt msgs. & sign (create) signatures.
Infeasible to determine private key from public is asymmetric box, those who encrypt msgs (or) verify signature cannot decrypt msgs (or) create signatures.

Encryption with public key:

- Bob's public key
- Alice's public key
- Transmitted ciphertext
- Encryption alg (ex: RSA)
- Alice & private key
- Plaintext
- Encryption alg (ex: RSA)
- Transmitted ciphertext
- Decryption alg
- Plaintext
- Alice

Conventional Encryption

Needed to work:
1. The same alg with the same key is used for encryption & decryption
2. The S & R must share the alg & the key

Security:
1. The key must be kept secret
2. It must be impossible (or) impractical to decrypt any

Public-key Encryption

Needed to work:
1. One alg is used for encryption & decryption with a pair of keys, one for encryption & one for decryption
2. The S & R must each have one of the matched pairs of keys (not the same one)
3. One of the 2 keys must be kept secret
4. Knowledge of the alg is impractical to recover any
Principles of public key crypto systems

Key Pair:
- Key Pair, source:
- Key Pair, destination:

Public-key Applications:
- Can classify uses into 3 categories:
  - Encryption / decryption (provide secrecy)
  - Digital signatures (provide authentication)
  - Key exchange (of session keys)
- Some algorithms suitable for all uses
- Others are specific to one

<table>
<thead>
<tr>
<th>Algm</th>
<th>Encryption / Decryption</th>
<th>Digital Signature</th>
<th>Key Exchange</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Elliptic Curve</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Diffie-Hellman</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>DSA</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

Public-key Requirements:
to find decryption key knowing only align
& encryption key.
* It is computationally easy to en/decrypt
msg when the relevant (en/decrypt)
key is known.
* Either of the two related keys can
be used for encryption, with the other used
for decryption (for some alg).
* These are formidable requirements
which only a few alg have satisfied.
* Need a trap-door one-way fun.
* One-way fun has,
  $y = f(x)$ easy
  $x = f^{-1}(y)$ infeasible.
* A trap-door one-way fun has,
  $y = f_k(x)$ easy, if $k & x$ are known
  $x = f_k^{-1}(y)$ easy, if $k \& y$ known
  $x = f_k^{-1}(y)$ infeasible, if $y$ known but
  $k$ not known.
* A perfect trap-door key scheme depends
  on a suitable trap-door one-way fun.
Security of public-key schemes:

- Like private key schemes, brute-force exhaustive search attack is always theoretically possible.
- But keys used are too large (> 512 bits).
- Security relies on a large enough difference in difficulty between easy (encrypt, decrypt) vs hard (cryptanalyse) problems.
- More generally, the hard problem is known, but is made hard enough to be impractical to break.
- Requires the use of very large numbers.
- Hence, is slow compared to private key schemes.

The RSA algorithm:

- Best-known & widely used public-key scheme.
- Based on exponentiation in a finite field, specifically modulo a prime.
Exponentiation takes \( O((\log n)^3) \) ops (easy).

Uses large integers (ex. 1024 bits).

Security due to cost of factoring large no.

Factorization takes \( O(e \log n \log \log n) \) ops (hard).

**RSA Encryption/Decryption:**

To encrypt a msg \( M \), the sender:

1. Obtain public-key of recipient.
   
   \( P_U = \{ e, n \} \).

2. Computes: \[ C = N^e \mod n \] where \( 0 \leq N < n \).

To decrypt the ciphertext \( C \), the owner:

1. Uses their private key \( P_R = \{ d, n \} \).

2. Computes: \[ M = C^d \mod n \]

Note that the msg \( M \) must be smaller than the modulus \( n \) (blek if needed).

**RSA Key Setup:**

Each user generates a public/private key pair by selecting two large primes at random: \( p, q \).
Computing the system modulus,

\[ n = p \cdot q \cdot \frac{\varphi(n)}{\gcd(e, \varphi(n))} \]

Selecting at random the encryption key \( e \), where \( 1 < e < \varphi(n) \) and \( \gcd(e, \varphi(n)) = 1 \).

Solve the following eqn to find decryption key \( d \):

\[ e \cdot d = 1 \pmod{\varphi(n)} \quad \text{and} \quad 0 \leq d \leq n \]

Publish the public encryption key:

\[ PU = \{ e, n \} \]

Keep secret private decryption key:

\[ PR = \{ d, n \} \]

Why RSA works –

Booz of Euler's theorem:

\[ a^{\varphi(n)} \equiv 1 \pmod{n} \]

where, \( \gcd(a, \varphi(n)) = 1 \)

In RSA have:

\[ n = p \cdot q \]

\[ \varphi(n) = (p-1)(q-1) \]

Choose \( e \) and \( d \) to be inverses.
Hence,\[ C = M^{e \cdot d} \mod N = M^{1+k \cdot \phi(n)} = M^{\phi(n) \cdot k} = N^k \cdot (N^{\phi(n)})^k = N^k \cdot (1)^k = N^k = M \mod n.\]

**RSA Example - Key Setup:**

1. Select Primes: \( p = 17 \) & \( q = 11 \).
2. Calculate: \( n = pq = 17 \times 11 = 187 \).
3. Calculate: \( \phi(n) = (p-1)(q-1) = 16 \times 10 = 160.\)
4. Select \( e \): \( \gcd(e, 160) = 1 \), choose \( e = 7 \).
5. Determine: \( d : de = 1 \mod 160 \) and \( d < 160.\)
   Value is \( d = 23 \) since \( 23 \times 7 = 161 = 2 \times 160 + 1 \).
6. Publish public key: \( PU = \{ 7, 187 \} \).
7. Keep secret private key: \( PR = \{ 23, 187 \}.\)

**RSA Example - Encryption / Decryption:**

- Example msg: \( M = 88 \) (nb. \( 88 < 187 \)).
  - Encryption: \[ C = 88^7 \mod 187, C = 1 \]
  - Decryption: \[ M = 17^{23} \mod 187 \]

www.studentsfocus.com
RSA Security

Possible approaches to attacking RSA are:

1. Brute-force key search:
   - Infeasible given size of nos.
   - Trying all possible private keys

2. Mathematical attacks
   - The approaches to factor the product of two prime numbers

3. Timing Attack:
   - Depends on the running time of the decryption algo.

Defense to Brute Force Attack:

- Use large key space
- Larger no. of bits in even of the better secured but prbms are,
- Increased computing power
- Factoring prblm

Timing attacks:

- Use constant exponentiation time
- Add random delays
- Blind values used in calculation
Key Management:

Key mgmt & Distribution:

Symmetric schemes require both parties to share a common secret key.

Key Hierarchy:
- Session key:
  - Temporary key:
    - Used for encryption of data between users.
    - For one logical session, it is discarded.
  - Master key:
    - Used to encrypt session keys.
    - Shared by users & key distribution center.

Key Distribution Scenario:

1. ID_A || ID_B || N
2. E(k_A, E(k_S, ID_A || ID_B || N))
3. E(k_B, E(k_S, ID_A))
4. E(k_S, N_2)
5. E(k_S, S_1(N_2))
Key distribution issues:

* KDCs required large network but must trust each other.

* Session key lifetimes should be limited for greater security.

* Use of automatic key distribution on behalf of users, but must trust sys.

* Use of decentralized key distribution.

* Controlling key usage.

Symmetric key distribution using public keys:

* Public key cryptosystems are inefficient.

  → So almost never use for direct data encryption.

  Rather use to encrypt secret keys for distribution.

Simple secret key distribution:

* Allow and secure comm.

* No key before after exist.

* Proposed by Merkle.

(1) $PU_n \rightarrow JDA$

(2) $E(Pub, {K})$
Secret key distribution with confidentiality.

Do Authentication?

(1) $E(Pub, Enill IDA)$

(2) $E(Pub, Enill NA)$

(3) $E(Pub, N2)$

(4) $E(Pub, E(Pub, K3))$

Distribution of public keys:

* Public announcement
* Publicly available directory
* Public key authority
* Public key certificates

Public Announcement:

* Users distribute public keys to recipients
  (a) broadcast to community at large

Major weakness is forgery:

Anyone can create a key claiming to be someone else's broadcast if

Until forgery is discovered can masquerade as claimed user.

Publicly available directory.
Properties:
- Contains entries for participants registering securely with directory.
- Can replace key at any time.
- Directory is periodically published.
- Can be accessed electronically.
- Still vulnerable to tampering or forgery.

Public Key Authority:
- Improve security by tightening control over distribution of keys from directory.
- Requires users to know public key for directory, then users interact with directory to obtain any desired public key.
- Does require real-time access to directory when keys are needed.
- May be vulnerable to tampering.

\[\text{Public Key Authority} \Rightarrow (\text{Public Key Authority}) \Rightarrow \ldots \Rightarrow (\text{Public Key Authority})\]
Public-key certificates:

- Allow key exchange without real-time access to public-key authority.
- A certificate binds identity to public key.
- All contents signed by a trusted public-key (or certificate) Certificate Authority (CA).
- Can be verified by anyone who knows the public-key authority's public key.

Diffie-Hellman key exchange:

- Public-key scheme proposed by Diffie & Hellman in 1976.
- This method is a practical method for public exchange of a secret key.
- Used in a no of commercial products.
Algorithm:

1) Global public elements:
   - q: Prime no.
   - $\alpha$: $\alpha < q$ and $\alpha$ is a primitive root of $q$.

2) User A key generation:
   - Select private $\text{XA}$. $\text{XA} < q$.
   - Calculate public $\text{YA}$. $\text{YA} = \alpha^{\text{XA}} \mod q$.

3) User B key generation:
   - Select private $\text{XB}$. $\text{XB} < q$.
   - Calculate public $\text{YB}$. $\text{YB} = \alpha^{\text{XB}} \mod q$.

4) Calculation of secret key by User A:
   - $K = (\text{YB})^{\text{XA}} \mod q$.

5) Calculation of secret key by User B:
   - $K = (\text{YA})^{\text{XB}} \mod q$.

Example:
- Users Alice & Bob who wish to swap keys.
- Agree on prime $q = 353$ and $\alpha = 3$.
- Select random secret keys:
  - Alice chooses, $\text{XA} = 97$.
  - Bob chooses, $\text{XB} = 233$.
Compute respective public keys:
\[ y_A = 3^9 \mod 353 = 248 \quad (\text{Alice}) \]
\[ y_B = 3^{233} \mod 353 = 160 \quad (\text{Bob}) \]

Compute shared session key as:
\[ K_{AB} = y_B^{x_A} \mod 353 = 160 \quad (\text{Alice}) \]
\[ K_{AB} = y_A^{x_B} \mod 353 = 40 \quad (\text{Bob}) \]

Key Exchange Protocols:

**User A**
- Generate random \( x_A \leq q \);
- Calculate \( y_A = \alpha^{x_A} \mod q \);
- Calculate \( K = (y_B)^{x_A} \mod q \).

**User B**
- Generate random \( x_B \leq q \);
- Calculate \( y_B = \alpha^{x_B} \mod q \);
- Calculate \( K = (y_A)^{x_B} \mod q \).
Man-in-the-Middle Attack:

1. Darth prepares by creating two private/public keys.
2. Alice transmits her public key to Bob.
3. Darth intercepts this & transmits his 1st public key to Bob. Darth also calculates a shared key with Alice.
4. Bob receives the public key & calculates the shared key [with Darth instead of Alice]
5. Bob transmits his public key to Alice.
6. Darth intercepts this & transmits his 2nd public key to Alice. Darth calculates a shared key with Bob.
7. Alice receives the key & calculates the shared key [with Darth instead of Bob].
   Darth can then intercept, decrypt, forward all msgs, & re-encrypt, Alice & Bob.

www.studentsfocus.com
Elliptic curve Arithmetic

ECC offers equal sec for a far smaller key size.

Confidence level in ECC is not yet as high as that in RSA.

Abelian Group:

- A set of elements with a binary opn, denoted by \( \cdot \) that associates to each ordered pair \((a, b)\) of elements in \( G \), an element \((a \cdot b)\) in \( G \),

(A1) Closure: If \( a \) and \( b \) belong to \( G \), then \( a \cdot b \) is also in \( G \).

(A2) Associativity: \( a \cdot (b \cdot c) = (a \cdot b) \cdot c \)

for all \( a, b, c \) in \( G \).

(3) Identity Element: There is an element \( e \)

in \( G \), such that, \( a \cdot e = e \cdot a = a \).

(4) Inverse Element: For each \( a \) in \( G \), there is an element \( a' \) in \( G \), such that,

\( a \cdot a' = a' \cdot a = e \).

(5) Commutative: \( a \cdot b = b \cdot a \) for all \( a, b \).
Example: \[ y^2 = x^3 - x. \]

Elliptic curve cryptography:

- ECC addition is analog of modulo multiplication.
- ECC repeated addition is analog of modulo exponentiation.
- Need "hard" problem equivalent to discrete log.
- \[ Q = kP, \] where \( Q \) and \( P \) belong to a prime curve.
- It is "easy" to compute a given \( k \) \( P \) but hard to find \( k \) given \( Q \) at the elliptic curve.
ECC Diffie-Hellman key exchange

Global Public Elements:

Eq(a, b) - elliptic curve with parameters a, b, and q, where q is a prime or an integer of the form $2^m$.

$G_1$ - point on elliptic curve whose order is a large value $n$.

Use A key generation:

Select private $n_A$, $n_A < n$.

Calculate public $P_A$, $P_A = n_A \times G_1$.

Use B key generation:

Select private $n_B$, $n_B < n$.

Calculate public $P_B$, $P_B = n_B \times G$.

Calculation of secret key by User A:

$k = n_A \times P_B$.

Calculation of secret key by User B:

$k = n_B \times P_A$.

ECC encryption / decryption:
Must first encode any msg \( M \) as a point on the elliptic curve \( P_m \).

Select suitable curve & point \( G_1 \) as in D-H.

Each user chooses private key \( nA < n \) and computes public key \( P_A = nA G_1 \).

To encrypt \( P_m : C_m = \{ kG_1, P_m + kP_B \} \),

\( k \) random.

Decrypt \( C_m \) compute:

\[
P_m + kP_B - nB(kG_1) = P_m + k(nB G) - nB(kG_1) = P_m
\]

Comparable key sizes for equivalent security.

Symmetric scheme

<table>
<thead>
<tr>
<th>Key size (in bits)</th>
<th>RSA/DSS (modulus size in bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>112</td>
</tr>
<tr>
<td>80</td>
<td>160</td>
</tr>
<tr>
<td>112</td>
<td>224</td>
</tr>
<tr>
<td>128</td>
<td>256</td>
</tr>
<tr>
<td>192</td>
<td>384</td>
</tr>
<tr>
<td>256</td>
<td>512</td>
</tr>
</tbody>
</table>