CS 6260 - APPLIED CRYPTOGRAPHY

MODULE 1 - INTRODUCTION AND SYMMETRIC ENCRYPTION

- Cryptography → Secret writing
  → Ancient: Caeser, Enigma m/c etc. but only last 40 years as a science
- Main goals: 1) Data privacy / Confidentiality : Nobody else can read msg
  2) Data Integrity : Message has not been modified in comm^n/storage
  3) Data Authenticity : Message came from where it claims
- Data Privacy: Protected data from unintended eavesdroppers
- About 1/2 internet traffic encrypted.
- Main players and Settings:
  - 2 main players Alice, Bob - Sender, Receiver (S,R) + Eve or A: Attacker
  1) Symmetric Key Setting: Both S and R share same key
  * some secret key must be known to both S an R unknown to A - else there can be no security because A can do everything S and R can do.
  - Efficient and secure, but S and R have to agree on key a priori
  2) Asymmetric/Public Key Setting: Only receiver holds secret key → Asymmetric
  corresponding public key of receiver known to everyone (sk_R, pk_R)
  - Might use some public directory of available keys...
  - New, last 40 years - Counter intuitive, hard to achieve?!
- [See Table]

Goal	Symmetric	Asymmetric
Privacy	" Encryption	" Encryption
Integrity/ Auth	MAC	Dig. Signature

- How good is a scheme/Protocol:
  1) Trial and error: Easy to find attack → Fix attack ← IF no attack found, don't know if secure
  2) Provable Security Approach:
  Use proof by contradiction → Prove that if an attack to this scheme is found, some other hard problem can be solved or some trusted assumptions broken
  - Need a good definition of security
    - Helps to find attacks more easily + Prove security

Symmetric Encryption

- Syntax : 1) Message Space $MsgSp$ 2) Key gen algo K or Key Space $KeySp$ and randomly pick from it
  3) Encryption, decryption algorithms E, D

  $C = E(M, K, ?)$ $\quad M = D(C, K)$ $\quad M \in MsgSp$ $\quad K \in KeySp$

  - Key gen is often a random pick.
  - Encryption may have random i/ps but decryption does not! (or a state i/p)

* Basic correctness Requirement:

For every msg $M \in MsgSp$ and key $K \in KeySp$, $D(K, E(K, M)) = M$.

- One Time Pad (Vernam Cipher, 1919)

  Message Space: Set of all bit strings of length n $\quad MsgSp = \{0,1\}^n$

  Key Space: Same as Message Space - choose random $\quad KeySp = \{0,1\}^n$

  $E(K, M)$: $C \leftarrow M \oplus K$ return C

  $D(K, C)$: $M \leftarrow C \oplus K$ return M

  * Should use 1 key only once (or) need new key for every n-bit chunk

  ◦ Simple and efficient

  ◦ Intuition for security → should not give adversary any new information they didn't already know

- Perfect / Shannon Security (1949)

  An encryption scheme $SE(K, E, D)$ is Shannon secure if for every ciphertext C and messages M_1, M_2: $\quad P(E(K, M_1) = C) = P(E(K, M_2) = C)$

  where the probabilities are over a random choice of key.

  Alternately: $\quad P(E(K_1, M_1) = C) = P(E(K_2, M_2) = C)$ for some K_1, K_2.

- For any ciphertext C, any message is equally likely when key is unknown
  → Very strong security, No leakage of information.

- Theorem / Proof - One Time Pad is perfectly secure or Shannon Secure.

  1) Fix $C \in \{0,1\}^n$ $\quad$ 2) choose $M_1, M_2 \in \{0,1\}^n$

  For every M, $\quad P(E(K, M) = C) = 1/2^n = P(K = M \oplus C)$

  - If M and C are fixed only one key works $\Rightarrow P(\cdots = C)$ is $1/2^n$ keys

  $\Rightarrow$ One Time Pad is perfectly secure or Shannon Secure.

- Then why use any other scheme: Can only send msg as long as key!

- Optimality of One Time Pad for Shannon Security:

If a scheme is Shannon secure $|KeySp|$ cannot be smaller than $|MsgSp|$

[Size of keysp cannot be smaller than size of msg space]

Proof:

1) Fix ciphertext C by picking msg M_1 and key K [Not just OneTimePad]

$\Rightarrow P(E(K,M)=C) > 0$ for some K (Because we used 1 K to get C)

2) Assume there exists $M_2 \in MsgSp$ such that $P(D(K,C)=M_2) = 0$.

there is no key for which C decrypts to M_2.

$\Rightarrow P(E(K,M_2)=C) = 0$ by correctness requirement

Thus $P(\cdot\cdot M_1)\cdot\cdot \neq P(\cdot\cdot M_2)$ → Violates Shannon Secrecy

$\Rightarrow$ For scheme to be Shannon secure, for every message M_2, there exists $K' \in KeySp$ so that $D(K',C) = M_2$

[For every message, there is at least 1 key that results in C. But 2 messages & same key can't encrypt to C]

$\Rightarrow |KeySp| \geq |MsgSp|$

- But we want key space to be $\leq$ Msg Space for a scheme to be practical.

→ End of Information-theoretic cryptography

→ Relax security requirement (not Shannon secure) - assume attackers are computationally bounded • Computational Cryptography • complexity

- Assumptions: 1) Existence of hard problems 2) Secret keys kept secret

George Notes Extra

- $\forall m \in MsgSp, \forall k \in KeySp$: $D(k, E(k,m)) = m$ *might as well store msg (repeated secret sharing)
- $k \xleftarrow{\$} KeySp$ - Notation for k randomly chosen from KeySpace

- Perfect Security: $\forall m \in MsgSp, \forall c$: $P(M=m|c) = P(M=m)$

- Given a ciphertext, all messages are equally probable

* Not: A ciphertext occurs & equal prob. for a given message - $P(C=c|m) \neq P(c=c)$

Actually: Any message can map to any ciphertext & equal prob. (CipherSpace is not limited here)

- What happens if we reuse key for OTP → $C_1 \oplus C_2 = m_1 \oplus m_2$

Misc 1: XOR only binary fn & equal prob 2: Prob & sharing new key each time

Module 2: Symmetric Encryption

- Goal: How to achieve data confidentiality in symmetric key setting

  Tool: Symmetric Encryption Scheme - is a cryptographic primitive.

  Process: Definition / Syntax of primitive → its security → Examples of specific schemes that use the primitive → Security of those schemes

- Syntax of symmetric Encryption:

  1) Key Generation Algorithm ($\mathcal{K}$): Randomized algo - i/p: random o/p: key
     - May simply involve choosing a key at random from a set KeySp
  2) Encryption Algorithm ($\mathcal{E}$): Run by sender; Message from set MsgSp (set of all possible messages)
     - Can be randomized or have state inputs (counter)
     - o/p: Ciphertext - Assumed to be sent over the channel
  3) Decryption algorithm ($\mathcal{D}$): Run by receiver; i/p: ciphertext, Key o/p:
     - Deterministic: No random i/ps or states.

  - To be useful, satisfy correctness i.e. $D(K, E(K,M)) = M \quad \forall M \in MsgSp, K \in KeySp$
  - Common KeySpace = $\{0,1\}^k$ for some integer k → Key chosen randomly from here

- Block Ciphers: Main tool for building practical SE schemes

  Eg: DES, 3DES, AES etc.

  - Used to encrypt short strings
  - Function family $E: \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$
  - Defined as $E_K : E(K,M)$ for $K \in \{0,1\}^k \rightarrow E_K$ is a permutation i.e. 1to1 and onto function on $\{0,1\}^n$
    * For every $C \in \{0,1\}^n \; \exists$ single $M \in \{0,1\}^n : C = E_K(M)$

    → Each M maps to 1 C and only 1 C + Every M has a corresponding C

    → Each C is generated from 1 and only 1 M " C has " M.

[Note: Onto = Multiple M can map to 1 C. Into: Can have C with no M
[Multiple C from 1 M is not a function: Many M without C - also not a fn]
4 categories: Many to one vs One to One / Onto vs Into]

→ Well defined inverse: E_K^{-1}: $E_K(E_K^{-1}(C)) = C$ and $E_K^{-1}(E_K(M)) = M \; \forall M, C$

- Block Cipher Modes of Operation → How to use the blockcipher (which encrypts short strings) to encrypt long strings → Assume messages are of

sizes = multiple of block length (can pad message if needed)

1) Electronic Code Book (ECB) Mode:

- Simplest Block cipher - Simply use block cipher directly for each block
- "encryption modes are encryption schemes that use block ciphers as a tool"
  - These encryption schemes also called symmetric blockcipher based modes of operation.
- Practical SE scheme: $ECB = (\{0,1\}^k, E, D)$
- Scheme: Divide msg M (length nk) block by block - encrypt each block with blockcipher using key picked at random from KeySp
  C = concatenation of all the block ciphertexts.
  Decryption " ... (similar)
- Correctness: (Obviously correct)
- Cannot be Shannon Secure - Because of Impossibility result i.e. short key to encrypt long messages
- (Goal: Data confidentiality) Intuition for Security -
  Adversary can't find key or decrypt C. But if 2 blocks of message are same → Ciphertexts of those blocks (deterministic) will also be the same
  - Some leakage of information

2) Cipher Block Chaining (CBC) Mode with random IV: (CBC$)

$CBC\$ = (\{0,1\}^k, E, D)$

- Choose a random string as IV = initial Vector → XOR with 1st message block. C from 1st block XOR'ed with next message block ...
- Decryption: Receiver gets (IV, C) → Decrypt first block of C, XOR with IV, then XOR decryption of next block with C_0, etc.
- Correctness by properties of XOR
- Security: 1) Because of chaining, 2 message blocks with same value don't produce same C.
  2) Need new IV for each new message.

3) Stateful CBC with counter IV (CBCC)

- Similar to previous scheme, but instead of random IV, use counter starting at 0^n (Increment for next message)
  - Assume counter does not wrap around: for n = 128, need to send

2^{128} messages - At that point, can renegotiate key

⇒ Each msg, new counter used + counter sent as part of C

- Correctness - Similar to previous
- Security: Attacker can learn counter by looking at C (See later section)

4) Randomized Counter Mode (CTR$) or XOR Mode: ↘

- No chaining, Parallelizable (III to ECB) $F: \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$
- Pick a random string of length l
- * The encryption function here does NOT have to be a block cipher!
  - ⇒ Does not have to be a permutation (does not have to be invertible!)
  - ⇒ I/p and o/p strings need not be same length!!
- Run function F on Random String (NOT message block) then XOR result with message to get C[i]. Repeat for each block
- To decrypt, repeat same process, then XOR with C[i] to get M[i] (Receiver knows R because it is the public part of C) (F uses the key)
- Correctness: By properties of XOR
- Security: ??

5) Stateful Counter Mode (CTRC) or Stateful XOR Mode

- Same block F as before. Use counter instead of random string
- Start ctr: 0^l. For next message, ctr will get incremented. Again assume ctr does not wrap around ...
- Sender maintains counter as state. Receiver gets counter along with C
- Still correct.

What is a Secure Encryption Scheme

- Assume attackers are computationally bounded i.e. cannot do exhaustive key search
- Scheme is secure when it is not insecure.
- Insecurity: Compute Secret key, Compute messages or any part of message, or any fn. of bits of message, see when equal messages are encrypted (If types of msgs is small and you learn 1 message, you can learn all!)

→ No adversary with reasonable resources who sees several ciphertexts can compute any partial info about PTs, besides apriori information

- Attacker may know message space or freq of messages (say Eng. words)
- Attacker knows length of PT → Public info

IND-CPA : Indistinguishability under Chosen Plaintext Attacks

- Definition (Practical) of Security for Symmetric Encryption Schemes
- Fix scheme, choose random key · Attacker knows all the algorithms, does not know key

  Experiment with 2 rooms with encrypting computer inside. You can enter room (don't know which room) then choose 2 messages from Msgsp of same length. The computer returns C for one of the messages depending on which room you are in. (C0 for room 0, C1 for room 1) → can keep entering message pairs.

  Scheme is secure if you cannot deduce the room number from the ciphertexts.

- Formal definition: Consider 2 experiments ind-cpa-0 and ind-cpa-1

  The attacker is given a left-right encryption oracle that takes 2 messages and return M_b under K (ind-cpa-0 returns M0 and ...)

  Attacker can repeat experiment ...

  IND-CPA advantage of attacker A = difference in probabilities that attacker was correct vs attacker was wrong when he o/p 0.

  $Adv^{ind\text{-}cpa}(A) = P[A \Rightarrow 0 \text{ in ind-cpa-0 exp}] - P[A \Rightarrow 0 \text{ in ind-cpa-1 exp}]$

* Symmetric Encryption Scheme SE is IND-CPA secure if for any adversary A with reasonable resources, its IND CPA advantage is small i.e. close to zero.

- What is the advantage of random attacker: $1/2 - 1/2 = 0$

  → Obviously not show that scheme is secure. → For this need to prove 0 for any (or every) attacker.

- Advantage for very good attacker (Always decrypt C and find K): $1 - 0 = 1$

→ Advantage lies between 0 and 1, for security, should be close to 0.

- Resources of adversary: (Attacker is an algorithm)

  1) Time complexity of algorithm: Measured using both worst case

running time + size of code. (usually running time more important)
- code may be very simple but sometimes running time may be too high.

2) Number of queries A makes → May be an attack that is fast but needs too many ciphertexts

3) Total length of all queries → May be a fast attack with 1 message but message is very long.

How to prove a scheme is not secure

1) Construct an adversary (pseudocode) that breaks the scheme under the definition of security

2) Calculate adversary's advantage → should not be too close to 0 for valid attack ($1/2^{60}$ is too small for valid attack, anything above this is acceptable.

3) Calculate adversary's resources → They have to be "reasonable" for a valid attack

→ This will mean that there is at least 1 attacker whose advantage is not too small under reasonable resources.

Examples for IND CPA definition:

1) Consider $SE = (K, E, D)$: $Msgsp = \{0,1\}^n$, $E_k(M) = M$, $D_k(C) = C$

To prove: SE is not IND-CPA (secure)

Proof: (1) Adversary $A^{E_k(LR(\cdot,\cdot,b))}$ makes query $E_k(LR(M_0, M_1, b))$: $M_0, M_1 \in Msgsp$, $M_0 \neq M_1$

Let's choose $M_0 = 0^n$ $M_1 = 1^n$. $C \xleftarrow{\$} E_k(..)$

If C == M_0 return 0 else: return 1

(2) $Adv^{IND-CPA}_{SE}(A) = Pr[A \Rightarrow 0 \text{ in ind-cpa-0}] - Pr[A \Rightarrow 0 \text{ in ind-cpa-1}] = 1 - 0 = 1$

(3) Resources: t = time to compare n bits
$q = 1$
μ = length of message = $2n$

2) Consider $SE = (K, E, D)$: $E_k(M) = M[1] \| E'_k(M$ where $SE' = (K, E', D')$ = secure scheme
$D_k(b \| C) = D'_k(C)$

To prove: SE is not IND-CPA

Proof: (1) Construct adversary $A^{E_k(LR(\cdot,\cdot,b))}$ that makes query M_0, M_1 : $M_0[1] \neq M_1[1]$

E.g. $M_0 = 0^n$, $M_1 = 1^n$ $C \xleftarrow{\$} E_k(LR(0^n, 1^n, b))$

If C[1] == $M_0[1] = 0$ then return 0 else 1

(2) $Adv_{SE}^{IND\text{-}CPA}(A) = 1 - 0 = 1$

(3) Resources: t = time to compare 1 bit $q = 1$ $\mu = 2n$

$\Rightarrow$ This scheme is not secure

2.5) Can generalize (2) to different schemes $\rightarrow$ if a bit or function of bits, attacker can distinguish b/w left and right on the oracle.

3) Analysis of ECB Mode:

Given: Blockcipher mode of operation $E: \{0,1\}^n \times \{0,1\}^k \rightarrow \{0,1\}^n$

$ECB = \{\{0,1\}^k, \mathcal{E}, \mathcal{D}\}$:

To Prove: ECB is not IND-CPA secure

Proof: Construct adversary $A^{\mathcal{E}_K(LR(\cdot,\cdot,b))}$ who queries with (M_0, M_1) then

$C_1 || C_2 \leftarrow \mathcal{E}_K(LR(0^{2n}, 0^n || 1^n, b))$: $M_0 = M_a || M_a$ $M_1 = M_b || M_c$

if $C_1 == C_2$ return 0 else 1

$Adv_{SE}^{IND\text{-}CPA}(A) = 1 - 0 = 1$

t = time to compare n bits
$q = 1$ $\mu = 4n$

$\Rightarrow$ ECB Mode is not IND-CPA secure.

Theorem: Any stateless deterministic scheme is not IND-CPA secure

Proof: [No state or randomness $\Rightarrow$ equal messages produce equal CTs]

Construct adversary $A^{\mathcal{E}_K(LR(\cdot,\cdot,b))}$. For simplicity, assume $MsgSp = \{0,1\}^n$

Claim can be shown to be true for arbitrary message space.

Note that this statement is about entire messages, not blocks within a msg.

$C_1 \leftarrow \mathcal{E}_K(LR(M_0, M_1, b))$ and $C_2 \leftarrow \mathcal{E}_K(LR(M_0, M_2, b))$: $M_1 \neq M_2$

Eg: $M_0 = M_2 = 0^n$ $M_1 = 1^n$ $C_1 \leftarrow 0^n, 1^n$ $C_2 \leftarrow 0^n, 0^n$

If $C_1 = C_2$ then return 0 else return 1

$Adv_{DE}^{IND\text{-}CPA}(A) = 1 - 0 = 1$

+ For distinct messages, C cannot be same by correctness.

For same M, C cannot be distinct because stateless, deterministic

Resources: t = time to compare 2 CTs of n bits each.

$q = 2$ $\mu = 4n$

4) Analysis of CBCC

Given: $\mathcal{E}(C_0 \oplus M_0) \rightarrow C_1$, $\mathcal{E}(C_1 \oplus M_1) \rightarrow C_2$ ... here C_0 = counter - sent along w/ C.

Proof: Construct adversary $A^{E_K(LR(\cdot,\cdot,b))}$ whose first query is:

$C_0 || C_1 \leftarrow E_K(LR(0^n, 1^n, b))$ $\quad C_0' || C_1' \leftarrow E_K(LR(0^n, 1^n, b))$

$C_0 = 0$ since first call to Oracle. Next time, $C_0' = 1$

If left message was encrypted $C_1 = E_K(0^n)$ else $C_1 = E_K(1^n)$. Now if we construct M_0 for next query s.t. $M_0 \oplus C_0'$ = all zeroes then $C_1 = E_K(0^n)$ M_1 will be diff.

[can use any $M_0 = \cdots 0$ and $M_0' = \cdots 1$ (or) can use $M_0 = C_0$, $M_0' = C_0'$, $M_1 \neq M_0$]

If $C_1 = C_1'$ return 0 else 1

$Adv_{CBC}^{IND\text{-}CPA}(A) = 1 - 0 = 1$

Resources: t = time to compare n bits, $q = 2$, $\mu = 4$

[$C_1 = E_K(C_0 \oplus M_0) = E_K(0^n \oplus 0^n) = E_K(0^n)$; $C_1' = E_K(C_0' \oplus M_0') = E_K(0^{n-1}1 \oplus 0^{n-1}1) = E_K(0^n)$

For $M_1 = M_1'$, $C_2 \neq C_2'$ because counter changes, i/p to b.c. is different. By correctness of blockcipher C_2 and C_2' can never be same]

* Very common technique, assume b.c. is secure → come up with M_0 : points entering b.c are same. vs M_1, message entering blockcipher are different.
- The other schemes shown above are secure if underlying BlockCipher is secure

Alternate Interpretation of IND-CPA (CPA-CG: CPA Choose and Guess)

Fix SE, Pick key. Consider an experiment cpa-cg between adversary & challenger (Only 1 experiment here). Pick a random challenge bit b at beginning of experiment. Does not change during experiment. (Based on b, left or right is returned) ⇒ Prob. of returning left or right = 1/2

Again, attacker wins if they guess b correctly → Return 1 if $b' = b$

$$\text{IND-CPA Adv}(A) = 2 * Pr[\text{expt. cpa-cg returns } 1] - 1$$

↳ using same definition of IND CPA adv. as before, not redefining advantage.

$$Pr[Exp_{SE}^{CPA\,CG} \Rightarrow 1] = Pr[b=b'] = Pr[b=b' \mid b=0]\,Pr[b=0] + Pr[b=b' \mid b=1]\,Pr[b=1]$$

$$= Pr[b'=0 \mid b=0] \cdot 1/2 + Pr[b'=1 \mid b=1] \cdot 1/2$$

$$= Pr[b'=0 \mid b=0] \cdot 1/2 + (1 - Pr[b'=0 \mid b=1])\,1/2$$

$$= \frac{1}{2} + \frac{1}{2}\left(Pr[b'=0 \mid b=0] - Pr[b'=0 \mid b=1]\right) = \frac{1}{2} + \frac{1}{2}(\text{IND-CPA advantage})$$

...

MODULE 3 : BLOCKCIPHERS AND THEIR SECURITY

- Main tool of symmetric cryptography Eg: DES, 3DES, AES etc.
  (Refer to previous module) k, n are parameters for block ciphers
  For each key K, there is a function mapping $M \to C$ - permutation.

DES - Data Encryption Standard

- Old standard from 70s : k = 56, n = 64
- History: NBS announced search for data prot. algo in 1973 → IBM submitted 'Lucifer' in 1974 → DES published in 1975 → DES approved as federal standard in 1976
  (NSA might have suggested some modifications to IBM's algorithm)
- $\approx 2.5 \times 10^7$ operations/s → Very fast

Security of Blockciphers

1) For any blockcipher (and any symmetric encryption scheme) an attacker can do exhaustive key search i.e. given $M_1, C_1 = E(K, M_1)$, $M_2, C_2 = E(K, M_2)$ .... $M_q, C_q = E(K, M_q)$ attacker can recover key as follows:
   for $i = 1, 2, .. 2^k$: if $E(T_i, M_1) = C_1$: if $E(T_i, M_j) = C_j \; \forall \; 2 \le j \le q$, return T_i
   ⇒ To break an algorithm with key size k, need worst case 2^k computations
   Average case: 2^{k-1} computations
- For DES, special property: $DES_K(x) = \overline{DES_{\bar{K}}(\bar{x})}$ → Speeds up key search by factor of 2
  ⇒ For 56b key → $2^{55} / 2 \times 2.5 \times 10^7$ s ≈ 23 years
2) Differential cryptanalysis: Finds the key given about 2^{47} chosen plaintext-CT pairs
   2^{47} is for DES
3) Linear cryptanalysis: Find with smaller number of PT-CT pairs (2^{42} for DES)
   → Not necessarily chosen.

- Can assume (2) and (3) need lot of data and are impractical. Assume (1) is the best known attack
- DES is not secure → 1) Regular m/c takes this much time, can build diff. m/c
  2) Parallelism.
- 1998: 2 m/cs - one did E.K.S. in 56 hours, another in 22hrs (1999)
  → Because key is not very long

Triple DES (3DES) $3DES(K_1 || K_2, M) = DES(K_2, DES^{-1}(K_1, DES(K_2, M)))$
uses double length key: 112b → Needs 3 computations.

- IS more secure than DES.

AES : Advanced Encryption Standard

- 1998 : NIST new search for block cipher → 15 algos (worldwide) → 2001 Rijndael algo chosen as winner (Belgian : Rijmen, Daemen)
- n = 128 k = 128, 192 or 256 b
- EKS, differential and linear cryptanalysis - infeasible

Security of Block Ciphers (Contd).

- Obviously key recovery should be infeasible - But this is not good enough
  → Need a security property to show security of SE scheme built from BC
- Intuition: 1) Key search should be hard 2) CT should not leak bits or fn. of PT
  Why can't we use security defn of encryption schemes for BCs?
  → Too strong! No block cipher can satisfy, because BC have to be deterministic
- Main idea: B.C is good if CTs "look" random ← Because random strings cannot leak information.
  In other words: B.C. is good if indistinguishable from a random fn. for an efficient attacker.
- Let Func(l, L) denote set of all functions from $\{0,1\}^l \to \{0,1\}^L$
  [For BC. $l = L$] A random function = random instance f from Func
  $g(\cdot) \xleftarrow{\$}$ Func(l, L). if T[x] not defined, then $T[x] \xleftarrow{\$} \{0,1\}^L$ return T[x]
  → Function means for same i/p need same o/p.
  → This is P
- To define a function, we only need to define the o/ps for all possible i/ps
  If we concatenate all possible strings → gives 1 representation of function

[How many functions possible for given domain and range?]

- Any long string would define a function + any func would have a long string

⇒ Picking a function at random from set of all possible functions ≡ Picking a long bitstring at random (Pick each bit with prob 1/2)

Pseudo random functions (PRFs)

A function family F is a PRF if the i/p-o/p behavior of its random instance is indistinguishable from a random function (computationally)

- My B.C. is secure if it behaves like a PRF. (For us F = Blockcipher)
  - Via oracle access of the function family (eg: Picking a key K gives the function)
  - there one world = b.c. other world = random func.
- Blackbox access: Attacker has access to a box or oracle to which he can send i/ps and observe o/ps [Implements random function described above]

PRF Security for Block-Ciphers.

Fix a b.c. or function E. Consider 2 experiments: Experiment-prf-0 (random) and experiment-prf-1 (real)

(1) Pick a random key K for E and this specifies blockcipher E_K
For every i/p, o/p = E_K

(2) Inside Oracle → random function g → function picked randomly from set of all functions with domain and range as E (see pseudocode above)
Both sides, same i/p → same o/p

Goal of attacker is to figure out which room they are in → b

$Adv^{PRF}(A) = Pr[A \Rightarrow 1 \text{ in } E_{pp\text{-}prf\text{-}1}] - Pr[A \Rightarrow 1 \text{ in } E_{pp\text{-}prf\text{-}0}]$

E is a secure PRF if for any adversary (& reasonable resources), PRFadv is small

* This definition works for ANY deterministic function with fixed domain and range, not just a blockcipher → Function doesn't have to have equal domains and ranges, doesn't have to be invertible (But we will use this definition mostly for blockcipher analysis)
- Why can't we simply use the random fn instead of trying to find PRF?
  → Problem is: How do you share a random function? Easy for a single party to compute random fn. as you go, hard to describe the fn to recvr (vs) Blockciphers - share short key that defines the fn. efficiently.
- Can we not use a similar definition for encryption schemes? (instead of IND-CPA, use Indistinguishability from Random Strings?) → Will need to modify PRF s.t. g is not deterministic → Yes, Possible to define.
  But...
  Might be too strong a definition, requires all bits to be random → Possible to have secure schemes even if this is not true.

→

- Testing this definition:

(1) B.C. that o/p msg in the clear each time: Obviously not secure

Attacker: if C==M, return 1 else 0.

Advantage: $P[A=1 \text{ in exp-prf-1}] - P[A=1 \text{ in exp-prf-0}] = 1 - 1/2^N \approx 1$

[If attacker is in exp-prf-1, he always returns 1. If he is in exp-prf-0 = Prob o/p of random fn $g(x) = x$ if $g(x), x$ are N bit strings. Then $P[x = g(x)] = 1/2^N$]

Obviously, not secure and a reasonable adversary can break it

⇒ Similar attacks possible if any info from message is leaked. Basically any info leaked will be noticed by attacker in real expt but not in random.

- Resources: Same as for E.S.

* Since i/p o/p length of B.C. is fixed → Total length of queries - fixed by num queries.

[For attack above: t = time to compare 2 nb. strings $q=1$ $\mu=2^n$

Security of Practical Blockciphers

- conjecture: AES is PRF secure

Concrete security statement (it is believed that): For any adversary A running time t making q queries:

$Adv^{PRF}(A) \leq const * (t/T_{AES}) * 1/2^{128} + (q^2/2^{128})$ [Upper Bound]

T_{AES} = Time to compute AES once, t = running time, q = no. of queries

Term 1: Exhaustive key search: if attacker had time $t << 2^{128}$ ⇒ very small

Term 2: Birthday bound attack: [Can be done against any blockcipher]

⇒ Attacker queries q distinct messages to oracle → if o/ps are distinct, respond with 1, else if at least 2 o/ps same respond with 0 ("random")

⊛ ↳ All o/ps will be distinct ⇒ $Adv(A) = 1 - q^2/2^{128}$.

- From the equation above, the advantage of an attacker is very small practically

MODULE -5 - MESSAGE AUTHENTICATION CODES FOR AUTHENTICITY/INTEGRITY

- Authencity e.g: signing Integrity eg. laminating
- Any good encryption scheme does NOT provide authenticity & integrity E.g. OTP - Shannon secure but an attacker can do $C' = C \oplus M'$ so that receiver will decrypt M' instead of M → can flip bits of the message
- MAC - Tool for auth./integ. in symmetric key setting.
  - Defined by: Message space, Key gen algorithm, MAC (Tagging/signing) algorithm, Verification Algorithm.
- MAC Syntax

$\Pi = (K, MAC, VF)$, $MsgSp$ Key K → usually randomly chosen from $\mathcal{K}$

MAC - Input: M, K Output: (M)Tag / MAC/signature

* A MAC does not have to be randomized or stateful to be secure

Most practical MACs are deterministic.

- For now, assume receiver gets both M and Tag.

VF - Input: Tag, M Output: 1/0 for accept or reject

- Correctness requirement:

$$\forall K, VF(K, M, MAC(K, M)) = 1$$

- If MAC is deterministic, then VF need not be defined - Receiver can simply MAC the received message and compare with Tag.

- MAC Security
- What can attacker do: May be possible to get sender to send chosen messages + receiver to verify tags it receives.
  - Necessary but not sufficient to not know the key.
  - If attacker can compute forgery i.e. new message and tag that pass VF.
  - Not concerned about replay attacks here.
- Formal definition: unforgeable under chosen message attack (UF-CMA)

Attacker given 2 oracles: MAC → Takes 1 input + VF → 2 inputs: Message + Tag

Goal of attacker: output a forgery Message, Tag i.e. Tag pass VF

constraint: M was not previously queried to oracle.

MODULE - 10 ASYMMETRIC ENCRYPTION or PUBLIC KEY ENCRYPTION

- Symmetric - parties share same key.

  Asymmetric - Sender, receiver or both parties holds secret key unknown to anyone else but public key is available. (can look up through PKI (public infrastr) public key

- Encryption : Receiver holds public key pair, sender needs to know recvrs
- Useful to communicate with sites with no apriori secrets shared
- Harder to design (Merkle in 1970 did undergrad project → ... Hellman)

Asymmetric Encryption Scheme - Syntax

$AE = (K, E, D)$, $MsgSp(pk)$

Message space may depend on public key!

Keygen: Output is a pair of keys: pk = Public key, Sk = Secret key - held secretly by receivr

Encryption Algorithm: Run by sender. Inputs: Msg, pk = Public key of receiver

- can be randomized (has to be), o/p = ciphertext.

Decryption Algorithm: Run by receiver. Inputs: CT, secret key O/P: Message

- May also run some check to see if CT is valid → o/p is ⊥ if invalid

Correctness Requirement: For every (pk, sk) output by KeyGen and any $m \in MsgSp(pk)$ if $C = E(pk, M)$ then $D(sk, C) = M$

- Assumptions:

1) Sender knows receivers public key and must be assured through PKI that it is authentic → Not assured by encryption.
2) Encryption algo is NEVER stateful - since any party can send messages to receiver simultaneously

- Security Intuition:

- No useful information must be leaked from CT → Same applies here.
- Similar to INDCPA in symmetric encryption.
- ... Run key gen to generate key pair. ... (M_0, M_1 of same length)

  Attacker is given the public key and not given the secret key

  Oracle encryption is done under public key.

  Adv = same as before = $Pr[A \Rightarrow 0 \text{ in indcpa0}] - Pr[A \Rightarrow 0 \text{ in indcpa1}]$

+ An asymmetric scheme AE is indis ... if for any adversary A with polynomial resources, its INDCPA advantage is negligible in security parameter

- More precise: Security param - typically length of public key in bits.

- Is any P.K. scheme insecure under this scheme? Since any adversary has P.K and can encrypt msg using P_K, then compare CT with oracle CT → NOT true because assumes encryption is deterministic!

[Same attack possible in SE setting (no key) - M_0, M_1 then M_0, M_0 ... only successful if scheme is deterministic.]

- INDCPA is not always enough!

1998 - Daniel Bleichenbacher - attack against Encrption Scheme part of SSL

-

MODULE - II - BASICS OF NUMBER THEORY

$\mathbb{Z}$ - Set of integers, $d|b \Rightarrow d$ divides b

$\mathbb{Z}_+$ - Set of +ve integers $\mathbb{Z}_N = \{0, 1, \cdots N-1\}$ for $N \in \mathbb{Z}_+$

$\mathbb{N}$ - Set of natural numbers (includes 0)

for $a, N \in \mathbb{Z}$, let $\gcd(a, N)$ be largest $d \in \mathbb{Z}_+ : d|a$ and $d|N$.

$\mathbb{Z}_N^* = \{i \in \mathbb{Z}_n : \gcd(i, N) = 1\}$ → only numbers from 1 to N-1, relatively prime to N

$\phi(N) = |\mathbb{Z}_N^*|$

Eg: $\mathbb{Z}_{12}^* = \{1, 5, 7, 11\}$ $\phi(12) = 4$

Division and Modulo

- For any $a \in \mathbb{Z}, N \in \mathbb{Z}_+$ $\exists$ unique r, q : $a = Nq + r$ $0 \le r < N$

$a \bmod N = r \in \mathbb{Z}_N$

- $a \equiv b \pmod N$ iff $a \bmod N = b \bmod N$ equivalence mod N

Eg: $17 \equiv 14 \pmod 3$

Groups

Let G be a non empty set with binary operation . defined on it. G is a group if :

1) Closure: $\forall a, b \in G$, $a \cdot b \in G$. (result of operation should be in set)

2) Associativity: $\forall a, b, c \in G$ $a \cdot (b \cdot c) = (a \cdot b) \cdot c$

3) Identity: $\exists 1 \in G : a \cdot 1 = 1 \cdot a = a$ $\forall a \in G$ (1 need not be the number 1)

4) Invertibility: $\forall a \in G$ $\exists b \in G : a \cdot b = b \cdot a = 1$ (b should be unique $= a^{-1}$)
↳ Again identity, not 1.

Group Examples (Prove 4 properties above)

$\mathbb{Z}_N$ - group under addition modulo N (Set alone does not define group, NEED operation)

- Inverse: $N - a$

$\mathbb{Z}_N^*$ - group under multiplication modulo N

1) $a \cdot b \bmod N$ - Since a, b rel. prime to N, cannot get 0. [Look into proof]

2) ... 3) 1 (number 1) 4) TBD.

Exponentiation Operation (for any group)

$a^i = \begin{cases} 1^* & \text{if } i = 0 \\ a \cdot a \cdots a & \text{if } i > 0 \ (i \text{ times}) \\ a^{-1} \cdot a^{-1} \cdot a^{-1} \cdots & \text{if } i < 0 \ (-i \text{ times}) \end{cases}$ * identity, not 1 — group opern, not mult.

$\forall a \in G$ and $i, j \in \mathbb{Z}$: $a^{i+j} = a^i a^j$, $a^{-i} = (a^i)^{-1} = (a^{-1})^i$

$(a^i)^j = a^{ij}$

- Order of a group = its size - no. of elements in group.

* If G is a group and $m = |G|$, $\boxed{a^m = 1 \ \forall \ a \in G}$.
* " $\boxed{a^i = a^{i \bmod m}}$ $\forall \ a \in G$ and $i \in \mathbb{Z}$ (Corollary)

E.g. $\mathbb{Z}_{21}^* = \{1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20\}$ under mult. mod 21

$m = 12$. $5^{86} \bmod 21 = 5^{86 \bmod 12} \bmod 21 = 5^2 \bmod 21 = 4$

- subgroup: set $S \subseteq G$ is called a subgroup (when G is a group) if it is a group by itself under the same operation as G.
  - If we already know G is a group, easy way to test if S is a subgroup: S is a subgroup iff $\boxed{x \cdot y^{-1} \in S \ \ \forall \ x, y \in S}$

* $|S| \mid |G|$ (order of S divides order of G)

Example: $G = \mathbb{Z}_{11}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ $S = \{1, 3, 4, 5, 9\}$

Running Times of algorithms

- Want crypto algos to be efficient and fast. But also want to have some algos not fast = hard problems. → for this, work with BIG numbers ($2^{512}, 2^{1024}, \ldots$)
- Look at worst case running times and Big-O
- Parameter: inputs to algo i.e. (function of) bit length of a number. (not the number itself)
  → Called Security Parameter.

Basic Algorithms (See Table)

Addition: By usual carry algorithm: $O(|a| + |b|)$ - Linear

Multiplication: " algo $O(|a| \cdot |b|)$ <multiplied> - Quadratic.

Integer Division: $O(|a| \cdot |N|)$ <multiplied> - Quadratic

Modulo: Same as division - Quadratic

GCD: For $a, N \in \mathbb{Z}$ and $a, N \neq (0, 0)$, gcd is the smallest integer in the set:

$\{a \cdot a' + N \cdot N' : a' N' \in \mathbb{Z}\}$

Corollary: if $d = \gcd(a, N)$, then there are weights $a', N' \in \mathbb{Z}$ s.t.

$\boxed{d = a \cdot a' + N \cdot N'}$ ↳ always exist.

E.g: $d = \gcd(20, 14)$ $a', N' = 2, -3$

Extended GCD: Outputs 3 numbers: (d, a', N')

(*) if $(q, r) = \text{INT-DIV}(a, N)$, $\boxed{\gcd(a, N) = \gcd(N, r)}$

E.g: $\text{INT-DIV}(17, 3) = (5, 2)$ $\gcd(17, 3) = \gcd(3, 2) = 1$

Running time of ext-GCD: (non trivial analysis) $O(|a| \cdot |N|)$ - Quadratic

Modular Inverse:

For a, N : $\gcd(a,N) = 1$, $a^{-1} \bmod N \equiv a^{-1}$ in $\mathbb{Z}_N^*$ → for mult mod N → unique $a' \in \mathbb{Z}_N^*$: $aa' \equiv 1 \pmod N$

But, $d = 1 = a \cdot a' + N \cdot N'$ and $N \cdot N' = 0 \pmod N \Rightarrow a \cdot a' = 1 \pmod N$

* Note that here identity = 1 because $\cdot$ = multiplication mod N.

MOD-INV(a,N): $d, a', N' \leftarrow$ EXT-GCD(a,N); return $a' \bmod N$ - Quadratic.

(Modular) Exponentiation:

For group G and $a^n \in G$, $n \in \mathbb{N}$, to compute $a^n = a \cdot a \cdot a \cdots$ (n times) → simple algo

- Obviously simple and correct. But, running time → $O(n) = O(2^{|n|})$ → Exponential
  < n is basically = $2^{|\text{bitsize of } N|}$ >
- Obviously slow and prohibitively expensive.

→ Example of simple algorithm that is slow.

But to use this to construct crypto. schemes → use better algo for ...

* Note: discussion above is for generic group, not nec for groups & operation = $\cdot \bmod N$
  i.e. discussion above is for exponentiation, not nec modular exponentiation

Fast Exponentiation

Repeated squaring: i steps for a^n if $n = 2^i$

For n not a power of 2 → any n can be written as $b_0 b_1 \cdots = 2^i b_0 + 2^{i-1} b_1 \cdots$

compute (say $2^4 b_4 + 2^3 b_3 + 2^2 b_2 + \cdots$) → $y_5 = 1$, $y_4 = y_5^2 \cdot a^{b_4}$, $y_3 = y_4^2 \cdot a^{b_3}$ ...

$b_i = 1$ or 0 here. (Basically multiply by b_i and left shift sort of - but doing this for the exponent, so square and multiply by a^0 or a^1.)

* This translates to $2ab_4 + b_3$, $4ab_4 + 2ab_3 + b_2 \cdots$

Algo: $y = 1$ for $i = a : k-1$ $y = y^2 a^{b_i}$

Running Time: Group operation + a^{b_i} is not an expon. (only y^2). Since $k = |n|$

$O(|n|)$ group operations - exponentiation

For $\mathbb{Z}_N^*$ defined with $\cdot$ = multiplication modulo N → Both quadratic running time ∴ Total running time = Cubic. = $O(|n| \cdot |N|^2)$ - Mod-exp.

* Modular Exponentiation often slowest / most expensive operation in crypto schemes.
- we want schemes to have polynomial running time. For efficient adversary, resources must be polynomial in security parameter.

- Negligible advantage - $g: \mathbb{Z}_p \to \mathbb{R}$ is negligible if it goes to 0 faster than any polynomial i.e. $\exists$ integer n_c for every positive integer c. $g(n) \leq n^{-c}$ $\forall n \geq n_c$
  [$1/2^k$ is negligible. $1/k^2$ is not]

MODULE - II - PART 2

- Order of a group element: (Not of the group itself!) - Smallest integer $n \geq 1$: $g^n = 1$
  - Because g is finite, the sequence $g^0 = 1, g^1, g^2 \cdots$ should repeat
    $\rightarrow \exists\ i, j$ with $i < j$: $g^i = g^j$
    But $1 = g^0 = g^{-i} \cdot g^i = g^{-i} g^j = g^{j-i}$ $\Rightarrow$ for some $m \geq 1$, $g^m = 1$
  - How to find order: Eg $\mathbb{Z}_{11}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$
    $o(2) = 10$ $o(5) = 5$ (Just write down the powers)
- Subgroup generated by g for $g \in G$:
  If $\langle g \rangle = \{g^0, g^1 \cdots g^{o(g)-1}\}$ $\rightarrow$ Subgroup of g whose order = $o(g)$ of G
  - $|S|$ always divides $|G|$ $\Rightarrow$ $o(g)$ of $g \in G$ always divides $|G|$ [See eg above]
    $\langle 2 \rangle = \{1, 2, 4, 8, 5, 10, 9, 7, 3, 6\}$ $\langle 5 \rangle = \{1, 5, 3, 4, 9\}$
* Generator - $g \in G$ is a generator if $\langle g \rangle = G$.
  Obviously g is a generator iff $o(g) = |G|$
  A group is cyclic if it contains a generator
- Discrete Log : If there is a cyclic group $G = \langle g \rangle$, for every $a \in G$, $\exists$ unique exponent $i \in \{0, 1 \ldots |G|-1\}$: $g^i = a$ $\rightarrow$ i is the discrete log of a to base g
  $\text{Dlog}_{G,g}(a)$
  * Inverse of the exponentiation function. [Example - Straightforward]
- Finding cyclic groups:
  (1) If p is a prime, $\mathbb{Z}_p^*$ is cyclic
  (2) If G is a group where $m = |G|$ is a prime number, G is cyclic
  * $\mathbb{Z}_p^*$ does not include 0, so has $m = p - 1$. $\therefore$ (2) does not imply (1)!
- Computing Discrete Logs: Given $G = \langle g \rangle$, $x \in G$, To find: $\text{Dlog}_{G,g}(x)$
  Algo 1: for $x = 0, \cdots |G| - 1$ do $x' = g^x$ if $x' = x$ return x.
  Complexity: $|G| \approx n$ (which repr in bits would be $\log_2 N = k$ $\Rightarrow$ $O(2^k)$
  Correct but exponential running time

- There are better algos $O(\sqrt{|G|})$ but still exponential and prohibitive
- Need: $O(n^c)$ for constant c and $n = \log|G|$ $(= k)$
  → Some algos for some groups but no poly time algo known.
- Good because this may be needed by adversary to break crypt. schemes
  i.e. we show best way to break algo is to compute discrete log
- Computational intractability is conjectured.

- Computing discrete log in $\mathbb{Z}_p^*$ for prime p:
  → There is an algo which can compute in $e^{1.92(\ln q)^{1/3}(\ln\ln q)^{2/3}}$ →
  where q is the largest prime factor of $p-1$ → we choose p s.t. $p = 2q+1$,
  $p \approx q$ when p is very large (q is a prime here.)
- Computing discrete log in Elliptic Curve Groups:
  If G is a prime order group of points over an elliptic curve → best known
  algo to compute Dlog → $O(\sqrt{p})$ where $p = |G|$
  - To compare algos that require Dlog in $\mathbb{Z}_p^*$ vs E.C.G. → Say we want
    80 bits of security i.e. algo should take 2^{80}
    For Dlog in $\mathbb{Z}_p^*$ with ||, we need $|\mathbb{Z}_p^*| = p-1 \approx 2^{1024}$ (for the order to be 2^{80})
    For Dlog in E.C.G of prime order p → since we need $O(\sqrt{p})$ we can set $p \approx 2^{160}$
- Why are smaller groups preferable:
  If we work with group size 2^{160} (160 bit numbers) exponentiation cost is
  cubic in $\log|G|$ → 160^3. This would be 1024^3 for group size 2^{1024}
  ⇒ Encryption/Decryption for $|G| = 2^{1024}$ is $(6.4)^3$ slower ≈ 262 times slower
  Elliptic Curve Groups preferable to create faster crypt. schemes with same level of security
- Constructing cyclic groups and finding generators:
  - If $|G|$ is prime, then every $g \in G - \{1\}$ is a generator (i.e. every non iden. element)
  - If $G = \mathbb{Z}_p^*$ where p is prime → in general hard to find generator,
    but easy if prime factorization of $p-1$ is known.

Algo: Pick $g \xleftarrow{R} G - \{1\}$ until (Test-Gen$_g$ = True)

i) How to test if g is a generator:

p is a safe prime if $p = 2q+1$ where q is a prime (for safe prime, Dlog of $\mathbb{Z}_p^*$ is slow) → Note that, here we know P.F. of $p-1$!
⇒ Easy to check if generator

- Observation: g is a generator iff $g^2 \neq 1$ and $g^3 \neq 1$ for $\mathbb{Z}_7^*$

  $\Rightarrow$ $g \in \mathbb{Z}_p^*$ is a generator iff $g^2 \neq 1$ and $g^q \neq 1$ $\rightarrow$ Use for TEST CHECK above

* $\mathbb{Z}_p^*$ has $q-1$ generators $\Rightarrow$ for the random choice step:

$$\Pr[\langle g \rangle = \mathbb{Z}_p^*] = \frac{q-1}{p-2} = \frac{q-1}{2q-1} \approx \frac{1}{2}$$

$p-2$ because 1 is not included.

on average need only 2 iterations to find a generator!

TEST CHECK step $\rightarrow$ g^2 - quadratic running time, $g^q \rightarrow$ cubic running time

- How to find large safe prime:

  FindPrime(k): $p \xleftarrow{R} \{2^{k-1} \ldots 2^k - 1\}$ // all numbers of k bits

  until p is prime and $(p-1)/2$ is prime; return p

- How to test p is prime:

1) Naive algo: for $i = 2, \ldots \sqrt{N}$, do $N \bmod i = 0$ then return false

   Correct but slow. $\rightarrow$ linear in N $\Rightarrow$ exponential in k.

Soln: $O(|N|^3)$ randomized algos and 1 deterministic algo.

Randomized used more commonly because constants for det running time large

- Density of Primes

obviously Pr of choosing prime randomly $= \pi(N)/N$, where

$\pi(N)$ = Number of primes in the range 1 to N $= \frac{N}{\ln N} \cdot \frac{1}{N} \approx \frac{1}{\ln(N)}$

E.g. $N = 2^{1024}$ Pr $\approx 1/1000$ $\rightarrow$ Not too bad!

- Squares

a is a square or quadratic residue modulo p if $\exists\, b: b^2 \equiv a \pmod p$

[b is the square root]. Then define:

$$J_p(a) = \begin{cases} 1 & \text{if } a \text{ is a square mod } p \\ 0 & \text{if } a \bmod p = 0 \\ -1 & \text{otherwise} \end{cases}$$

$\rightarrow$ Legendre or Jacobi symbol of a modulo p

- Set of Squares. QR = Quadratic Residue

$$QR(\mathbb{Z}_p^*) = \{a \in \mathbb{Z}_p^* : a \text{ is a square mod } p\} = \{\ldots : \exists b \in \mathbb{Z}_p^* : b^2 \equiv a \bmod p\}$$

E.g.: For $p = 11$: 5 squares, 5 non squares, each square has exactly 2 square roots

- Relation to Discrete Log:

  $J_p(a) = 1$ iff $Dlog_{\mathbb{Z}_p^*, g}(a)$ is even where g is a generator

$\because g^{2j} = (g^j)^2$. Since g^j always $\in \mathbb{Z}_p^*$, always a square.

for to be a square, if $x = g^n$, n has to be even!

* If $p \geq 3$ is a prime and g is a generator of $\mathbb{Z}_p^*$, then

$$QR(\mathbb{Z}_p^*) = \{g^i : 0 \leq i \leq p-2 \text{ and } i \text{ is even}\}$$ proof

- Computing J_p:

$$J_p(a) \equiv a^{\frac{p-1}{2}} \pmod p \quad \text{for } p \geq 3 \text{ is a prime for any int } a$$

E.g: $a=5$, $p=11$. $J_{11}(5) = a^5 = 5^5 = 1$

$a=6$, $p=11$ $J_{11}(6) = 6^5 = 5 \cdot 3 \cdot 6 = -1$

→ To compute Legendre symbol i.e. to test if square:

Test-Sq(p,a): $s \leftarrow a^{p-1/2} \bmod p$ if $s=1$ return 1 else return -1

Efficiency: modular exponentiation: Cubic running time. → Efficient

- Fact: 1) For prime $p \geq 3$ and g generator of $\mathbb{Z}_p^*$:

$J_p(g^{x \cdot y} \bmod p) = 1$ iff $J_p(g^x \bmod p) = 1$ or $J_p(g^y \bmod p) = 1$

$\forall x, y \in \mathbb{Z}_{p-1}$

→ $p-1$ is the order of the group $\mathbb{Z}_p^*$

2) For any a,b: $J_p(ab) = J_p(a) \cdot J_p(b)$ E.g. check with $J_{11}(a)$

3) For any $a \in \mathbb{Z}_p^*$: $J_p(a^{-1}) = J_p(a)$

4) For $QR(\mathbb{Z}_p^*)$ → if $\mathbb{Z}_p^*$ is cyclic and g is a generator,

A generator is always a non square. (Not all non squares are generators)

5) For $p \geq 3$ and g a generator of $\mathbb{Z}_p^*$ (see 1 fact above):

$|QR(\mathbb{Z}_p^*)| = (p-1)/2$ → because we are using $0 \leq i \leq p-2$

⇒ half elements are squares and half are not squares

MODULE - 12 - NUMBER THEORETIC PRIMITIVES

- cannot use blockciphers and PRFs as building blocks in public key setting because no shared key → Use discrete log based building blocks instead
- Discrete Log Related Problems:
  - Discrete Logarithm: Given g^x Find x
  - Computational Diffie Hellman (CDH): Given g^x, g^y Find g^{xy}
  - Decisional Diffie Hellman (DDH): Given g^x, g^y, g^z Find is $z \equiv xy \bmod |G|$
- Relation between these problems:

If we can solve DL → we can solve CDH → we can solve DDH

Hardness of DDH ⇒ Hardness of CDH ⇒ Hardness of DL

- For general tree/ graph search, cannot terminate yet because there may be a shorter path in the frontier → NOT needed for BFS.
- BFS WILL find shortest path in terms of number of steps.

UNIFORM-COST SEARCH (Cheapest first)

- Guaranteed to find path with cheapest cost → Choose lowest path-cost node from frontier
- Again don't terminate till goal is popped off the frontier
- Depth first search: always expand longest path

BFS and UCS are optimal → guaranteed to find ...
DFS - not optimal, may find longer paths
Why use DFS: storage requirements for frontier - BFS and UCS have $\approx 2^n$ nodes in frontier DFS only has n nodes.
→ Not that much savings if we track both explored and frontier

Completeness: If the goal exists somewhere, will the algo find it:

- BFS: Always complete (for infinite tree)
- UCS: complete if non-zero action cost separate from path cost. If there is a path $1 + 1/2 + 1/4 \cdots <= 2$, and shortest path >2, will be incomplete. (and no node has infinite number of successors)
  DFS: will follow the infinite path and never complete!

- Note that UCS expands in terms of equidistant contours i.e. is not directed towards goal → May have to explore half the space
- Heuristic: Estimate of distance from start to goal
- Greedy Best first Search: Expands the path that appears closest to the goal
  - Directed towards goal. But if there is an obstacle, may find a longer path than needed (Might still expand small num of nodes?!)
- A* - Combination of UCS and Greedy BFS
  - Expands path with minimum f, where g(path) = path cost
    h(path) = h(s) = estimated distance to goal.

[Best estimated total path cost first!]

* Actual dist >, Heuristic distance i.e. straight line distance

- DL ⇒ CDH: Given DL solver A_1, can easily construct A_2 which:
  1) uses A_1 to get x from g^x 2) Performs $(g^y)^x \to g^{xy}$.
- CDH ⇒ DDH: Given CDH solver A_2, can easily construct A_3 which:
  1) use A_2 to compute g^{xy} 2) Compare $g^{xy} == g^z$ → if yes, $z = xy$.

Formal definition of DL

for G - cyclic group of order $m = |G|$, g - generator:

$Exp^{dl}_{G,g}(A)$ $x \xleftarrow{\$} Z_m$; $X \leftarrow g^x$ $\bar{x} \leftarrow A(X)$. If $g^{\bar{x}} = X$ return 1 else 0

Randomly pick exponent → do not give attacker x, give attacker g^x

Attacker tries to predict exponent x

$$Adv^{DL}_{G,g}(A) = Pr[Exp^{dl}_{G,g}(A) = 1]$$

Dlog problem said to be hard in G if dl-adv of any adversary with polynomial resources is negligible (Resources here = Running Time)

Formal Definition of CDH

<Same as above>

$Exp^{CDH}_{G,g}(A)$ $x \xleftarrow{\$} Z_m$; $y \xleftarrow{\$} Z_m$ $X \leftarrow g^x$; $Y \leftarrow g^y$ $Z \leftarrow A(X,Y)$.

If $Z = g^{xy}$ then return 1 else return 0

Randomly pick x, y and give attacker g^x, g^y. Attacker predicts g^{xy}

$$Adv^{CDH}_{G,g}(A) = Pr[Exp^{CDH}_{G,g}(A) = 1]$$

<same as above>

Formal definition of DDH

Attacker not asked to compute but distinguish, so 2 experiments

$Exp^{DDH1}_{G,g}(A)$ $x \xleftarrow{\$} Z_m$, $y \xleftarrow{\$} Z_m$, $z \leftarrow xy \bmod m$; $X \leftarrow g^x$, $Y \leftarrow g^y$, $Z \leftarrow g^z$

<Real> $d \leftarrow A(X,Y,Z)$; Return d.

$Exp^{DDH0}_{G,g}(A)$ $x \xleftarrow{\$} Z_m$, $y \xleftarrow{\$} Z_m$, $z \xleftarrow{\$} Z_m$; $X \leftarrow g^x$, $Y \leftarrow g^y$, $Z \leftarrow g^z$

<Random> $d \leftarrow A(X,Y,Z)$; Return d

Attacker knows description and order of the group, and the generator

$$Adv^{DDH}_{G,g}(A) = Pr[Exp^{DDH-1}_{G,g}(A) = 1] - Pr[Exp^{DDH-0}_{G,g}(A) = 1]$$

... May call the o/ps of attacker as real Diffie Hellman tuple Vs random Diffie Hellman tuple

- **Hardness of these Problems:**

* The hardness of these problems is not absolute, depends on the underlying group!

- Hardness is conjectured, not proven
- DL, CDH in $\mathbb{Z}_p^*$ - hard $e^{1.92 (\ln p)^{1/3} (\ln \ln p)^{2/3}}$ in E.C. - harder $O(\sqrt{p})$

  DDH in $\mathbb{Z}_p^*$ - harder $\rightarrow O(\sqrt{p})$ But DDH in $\mathbb{Z}_p^*$ - easy! -

- If prime factorization of order of group is known, then DL can be solved in: $\sum_{i=1}^{n} \alpha_i (\sqrt{p_i} + |p|)$ where $p-1 = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_n^{\alpha_n}$ for $\mathbb{Z}_p^*$

  → For DL problem to be hard, we need at least 1 prime factor to be large:

  If we have $p = 2q+1$ then if p is large q is large, making algo above slow.

- **DDH in $\mathbb{Z}_p^*$**

  Can distinguish squares vs non squares in $\mathbb{Z}_p^*$

* If $p \geq 3$ is a prime and $G = \mathbb{Z}_p^*$, g is a generator of g. ∄ efficient DDH adversary A with advantage ≥ 0.5!!

Adversary $A(X, Y, Z)$.

If $J_p(X) = 1$ or $J_p(Y) = 1$,

  $s = 1$ else $s = -1$

If $J_p(Z) = s$ return 1 else 0

Idea here is to compute Legendre symbol and use it to distinguish between squares, non squares

In real experiment Z is a square iff X and/or Y is a square.

- $Adv^{DDH}_{G,g}(A) = Pr[\cdots = 1] - Pr[\cdots = 1] = 1 - 1/2 = 1/2$

  In experiment 1, always right. In expt 0, can be wrong at random

- Resources: compare bits + computing Legendre symbol 3 times:

  cubic time in $|p|$ → Polynomial running time → Efficient

- Luckily, DDH problem hard (conjectured) for several groups:
- $QR(\mathbb{Z}_p^*)$ → Subgroup of quadratic residues of $\mathbb{Z}_p^*$ where $p = 2q+1$
- Elliptic curves

MODULE-13- DL BASED PUBLIC KEY ENCRYPTION SCHEMES

- **DIFFIE HELLMAN KEY EXCHANGE:**

Let $G = \langle g \rangle$ be a cyclic group of order m. [these are public info]

Goal: Alice and Bob should agree on a shared secret

- Let each party pick a random exponent x, y, they exchange g^x, g^y

⇒ Both can compute g^{xy} $(= (g^y)^x = (g^x)^y =$ (what they receive)$^{\text{exponent}})$

To break this adversary has to compute $g^{xy} \leftarrow g^x, g^y$ (CDH!)

- DL ⇒ CDH: Given DL solver A_1, can easily construct A_2 which:
  1) uses A_1 to get x from g^x 2) Performs $(g^y)^x \to g^{xy}$.
- CDH ⇒ DDH: Given CDH solver A_2, can easily construct A_3 which:
  1) use A_2 to compute g^{xy} 2) Compare $g^{xy} == g^z$ → if yes, $z = xy$.

Formal definition of DL

for G - cyclic group of order $m = |G|$, g - generator:

$Exp^{dl}_{G,g}(A)$ $x \xleftarrow{\$} \mathbb{Z}_m$; $X \leftarrow g^x$ $\bar{x} \leftarrow A(X)$. If $g^{\bar{x}} = X$ return 1 else 0

Randomly pick exponent → do not give attacker x, give attacker g^x

Attacker tries to predict exponent x

$$Adv^{DL}_{G,g}(A) = Pr[Exp^{dl}_{G,g}(A) = 1]$$

DLog problem said to be hard in G if dl-adv of any adversary with polynomial resources is negligible (Resources here = Running Time)

Formal Definition of CDH

<Same as above>

$Exp^{CDH}_{G,g}(A)$ $x \xleftarrow{\$} \mathbb{Z}_m$; $y \xleftarrow{\$} \mathbb{Z}_m$ $X \leftarrow g^x$; $Y \leftarrow g^y$ $Z \leftarrow A(X,Y)$.

If $Z = g^{xy}$ then return 1 else return 0

Randomly pick x, y and give attacker g^x, g^y. Attacker predicts g^{xy}

$$Adv^{CDH}_{G,g}(A) = Pr[Exp^{CDH}_{G,g}(A) = 1]$$

<same as above>

Formal definition of DDH

Attacker not asked to compute but distinguish, so 2 experiments

$Exp^{DDH1}_{G,g}(A)$ $x \xleftarrow{\$} \mathbb{Z}_m$; $y \xleftarrow{\$} \mathbb{Z}_m$, $z \leftarrow xy \bmod m$; $X \leftarrow g^x$, $Y \leftarrow g^y$, $Z \leftarrow g^z$

<Real> $d \leftarrow A(X,Y,Z)$; Return d.

$Exp^{DDH0}_{G,g}(A)$ $x \xleftarrow{\$} \mathbb{Z}_m$, $y \xleftarrow{\$} \mathbb{Z}_m$, $z \xleftarrow{\$} \mathbb{Z}_m$; $X \leftarrow g^x$, $Y \leftarrow g^y$, $Z \leftarrow g^z$

<Random> $d \leftarrow A(X,Y,Z)$; Return d

Attacker knows description and order of the group, and the generator

$$Adv^{DDH}_{G,g}(A) = Pr[Exp^{DDH-1}_{G,g}(A) = 1] - Pr[Exp^{DDH-0}_{G,g}(A) = 1]$$

... May call the o/ps of attacker as real Diffie Hellman tuple vs random Diffie Hellman tuple

- Hardness of these Problems:

* The hardness of these problems is not absolute, depends on the underlying group.

- Hardness is conjectured, not proven
- DL, CDH in Z_p^* - hard $e^{(1.92+o(1))(\ln p)^{1/3}(\ln\ln p)^{2/3}}$ in E.C. - harder $O(\sqrt{p})$
  DDH in Z_p^* - harder $\rightarrow O(\sqrt{p})$ But DDH in Z_p^* - easy.

- If prime factorization of order of group is known, then DL can be solved in: $\sum_{i=1}^{n} \alpha_i (\sqrt{|p_i|} + |p|)$ where $p-1 = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_n^{\alpha_n}$ for Z_p^*

→ For DL problem to be hard, we need at least 1 prime factor to be large: if we have $p = 2q+1$ then if p is large q is large, making algo above slow.

- DDH in Z_p^*

Can distinguish squares vs non squares in Z_p^*

* If $p \geq 3$ is a prime and $G = Z_p^*$, g is a generator of g. $\exists$ efficient DDH adversary A with advantage of 1!!

Adversary $A(X, Y, Z)$.
If $J_p(X) = 1$ or $J_p(Y) = 1$,
  $b = 1$ else $b = -1$
If $J_p(Z) = b$ return 1 else 0

Idea here is to compute Legendre symbol and use it to distinguish between squares, non squares
In real experiment Z is a square iff x and/or Y is a square.

$Adv^{DDH}_{G,g}(A) = Pr[\cdots = 1] - Pr[\cdots = 1] = 1 - 1/2 = 1/2$

In experiment 1, always right. In expt 0, can be wrong at random

Resources: compare bits + computing Legendre symbol 3 times:
cubic time in $|p|$ ⇒ Polynomial running time → Efficient

Luckily, DDH problem hard (conjectured) for several groups:
$QR(Z_p^*)$ → Subgroup of quadratic residues of Z_p^* where $p = 2q+1$
Elliptic curves

MODULE-13- DL BASED PUBLIC KEY ENCRYPTION SCHEMES

DIFFIE HELLMAN KEY EXCHANGE:

Let $G = \langle g \rangle$ be a cyclic group of order m. [these are public info]
Goal: Alice and Bob should agree on a shared secret
Let each party pick a random exponent x, y, they exchange g^x, g^y
Both can compute g^{xy} $(= (g^y)^x = (g^x)^y$ - (what they receive)$^{exponent})$
Break this adversary has to compute $g^{xy} \leftarrow g^x, g^y$ (CDH!)

MODULE - 14 - RSA FUNCTION AND ENCRYPTION

- **RSA Math**

$\phi(N) = \mathbb{Z}_N^*$

Suppose $e, d \in \mathbb{Z}^*_{\phi(N)}$: $ed \equiv 1 \pmod{\phi(N)}$, then for any $x \in \mathbb{Z}_N^*$ we have

$(x^e)^d \equiv x \bmod N$

Because $x \in \mathbb{Z}_N^*$ we know that $x^{ed} = x^{ed \bmod \phi(N)} \equiv x^1 = x$!

- **RSA Function** $f : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ defined by: $\forall x \in \mathbb{Z}_N^*$

$$f(x) = x^e \bmod N$$

Decryption Exponent - $d \in \mathbb{Z}^*_{\phi(N)}$: $ed \equiv 1 \pmod{\phi(N)}$

* RSA function defined as above is a permutation with inverse given by:

$$f^{-1}(y) = y^d \bmod N$$

Proof: $f^{-1}(f(x)) \equiv (x^e)^d \equiv x \bmod N$.

E.g: $N = 15$ [Note for DL problems, group was $\mathbb{Z}_p^*$, here $\mathbb{Z}_N^*$ - n is not a prime]

$\mathbb{Z}_N^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$

$\phi(N) = 8$

$\mathbb{Z}^*_{\phi(N)} = \{1, 3, 5, 7\}$

Let $e = 3$ and $d = 3$ $\quad ed \equiv 9 \equiv 1 \bmod 8$

$f(x) = x^3 \bmod 15$

$g(x) = y^3 \bmod 15$

Note: e and d need not be same

Testing for inverse - Simply compute $f(x)$ and $g(f(x)) \to$ should give x

- **RSA Idea** $pk = N, e$ $\quad sk = N, d$

$E_{pk}(x) = x^e \bmod N = f(x)$; $D_{sk}(y) = y^d \bmod N = f^{-1}(y)$

Security relies on finding f^{-1} without knowing d.

RSA is actually a trapdoor i.e. a one way permutation

→ Given d, it is easy to invert but not easy given only N, e.

- **Generating RSA parameters** (from input - security parameter k)

Returns: 1) p, q = distinct odd primes 2) $N = (pq)$ - called RSA modulus

3) $|N| = k$ i.e. $2^{k-1} \le N \le 2^k$

4) $e \in \mathbb{Z}^*_{\phi(N)}$ - encryption exponent 5) $d \in \mathbb{Z}^*_{\phi(N)}$ - decryption exponent

- More RSA Math:

* If p, q are distinct primes and $N = p, q$ then $\phi(N) = (p-1)(q-1)$

$\phi(N) = |\{1, \dots, N-1\}| - |\{ip : 1 \le i \le q-1\}| - |\{iq : 1 \le i \le p-1\}|$

$= N - 1 - (p-1) - (q-1) = pq - p - q - 1 = (p-1)(q-1)$

- If $\phi(N)$ and $e \in \mathbb{Z}^*_{\phi(N)}$, d can be computed easily: $d \leftarrow$ MOD-INV$(e, \phi(N))$
  (Quadratic running time)
  Also good chance of picking prime at random

Building RSA Parameter Generator

Choose $e=3$ (commonly chosen for efficient exponentiation)

For simplicity assume k is even.

1) Pick at random p, q of bit length $k/2$ i.e. from $\{2^{k/2-1}, \ldots, 2^{k/2}-1\}$

$N = pq \quad M = (p-1)(q-1)$

- Repeat 1 until p, q are prime, $N \geq 2^{k-1}$ and $\gcd(e, M) = 1$*

* we want $e \in \mathbb{Z}^*_{\phi(N)}$, so e must be relatively prime to $\phi(N)$

2) Compute $d \leftarrow$ MOD-INV(e, M)

return N, p, q, e, d.

This algo is efficient because 1) enough primes available 2) MODINV is eff.

- Basic RSA Security

- We want finding x given N, e, y: $f(x) = x^e \bmod N$ to be hard

(or) $x \rightarrow f(x)$ easy, $f(x) \rightarrow x$ easy with d, $f(x) \rightarrow x$ hard with d

One wayness of RSA → one wayness under known (encryption) exponent attack

$\text{Exp}^{\text{ow-kea}}_{k,rsa}(A)$ → 1) use RSA gen algo to generate parameters → algo

N, e: public p, q, d: secret | shows e as generated but commonly e is fixed..

2) choose x randomly from $\mathbb{Z}_N^*$ (PT), encrypt and give N, e, y to adversary

Goal of adversary: decrypt y and returns x'

if $x' = x$, return 1 (attacker wins) otherwise 0

$$\text{Adv}^{\text{owkea}}_{K_{rsa}}(A) = \Pr[\text{Exp}^{\text{owkea}}_{k,rsa}(A) = 1]$$

- RSA believed to be one way under kea.
- Know N $\xrightarrow{1}$ Know p, q $\xrightarrow{2}$ Know $\phi(N)$ $\xrightarrow{3}$ Know d $\xrightarrow{4}$ Invert RSA

  4 - easy - know decryption works 2 - $\phi(N) = (p-1)(q-1)$ - compute directly

  3 - easy - basically MOD-INV(e) if e and $\phi(N)$ are known

  1 - would be easy if prime factorizn(N) is easy.

⇒ For security of RSA, 1 should be hard.

- Relation between inverting RSA and hardness of factoring: if factoring

- ElGamal Scheme Idea: Variant of DH Key Exchange for encryption
  - Alice has public key g^x and secret key x → Receiver
  - Bob (sender) - Picks y and sends g^y to Alice
    - Encrypts message M under $g^{xy} = (g^x)^y$ and sends CT to Alice
  - Alice decrypts using $g^{xy} = (g^y)^x$

  → Adversary still faced with CDH.

- Syntax / Details : $EG = (K, E, D)$ associated with cyclic group G order m

  K : $x \xleftarrow{\$} Z_m$, $X \leftarrow g^x$, return X, x

  $E_X(M)$: $y \xleftarrow{\$} Z_m$, $Y \leftarrow g^y$; $K \leftarrow X^y$; $W \leftarrow K \cdot M$; return (Y, W)

  $D_X(Y, W)$: $K = Y^x$; $M \leftarrow W \cdot K^{-1}$; return M

  Msg Sp → G

- Security - 1) To get x from X, attacker has to solve DL problem

  2) To get M given X (and Y, W), attacker must compute $K = g^{xy}$ i.e solve CDH.

  3) Leakage of Partial information i.e INDCPA or INDCCA security

* ElGamal is not IND-CPA in Z_p^* → Same reason as DDH being "easy" in Z_p^* i.e. squares vs non squares.

Proof : [Note that $m = |Z_p^*| = p-1$, so exponents chosen from 1 to p-1]

x = Secret Key X = Public Key y = randomly chosen exponent for X.

$Y = g^y$ → Publicly Transmitted W = ciphertext with key $K = g^{xy} = Y^x = X^y$

We consider Y, W to be the ciphertext Here $W = g^{xy} \cdot M = Y^x \cdot M = X^y M$

- Given X and Y, W, attacker can compute $J_p(M)$ [Recall: $J_p(g^{xy}) = 1$ iff $J_p(g^x) = 1$!!!]

  → (Partial) loss of information.

  FIND-J - efficient algorithm that can find $J_p(M)$ from X and (Y, W)

Attack

Let M_0, M_1 : $J_p(M_0) = -1$, $J_p(M_1) = 1$ → Attacker takes M_b and (Y, W) then finds $J_p(M_b)$. If FINDJ $(X, Y, W) = 1$ return 1 else return 0

Can choose $M_0 = g$, $M_1 = 1$!!!

Algorithm FIND-J (X, Y, W) If $J_p(Y) = 1$ or $J_p(X) = 1$, [$\Rightarrow K = g^{xy}$ is a square!]

then $s = 1$ else $s = -1$; Return $J_p(W) \cdot s$

Efficiency: Computing J_p + Regular multiplication (not Group Multiplicn)
- cubic running time ⇒ Efficient

* ElGamal is IND-CPA secure over any group G where DDH is hard ‼
  → Prime order subgroups for prime p of $\mathbb{Z}_p^*$ (Eg: Subgroup of squares)
  → Elliptic curve groups of prime order
* If DDH is hard in G, then AE_G is INDCPA secure
  [DDH is not hard in $\mathbb{Z}_{p^n}^*$ for prime p]

  Proof: (by contraposition)
  Assume A is an IND-CPA attacker for ElGamal scheme, construct adversary B for DDH that uses A.
  DDH adversary B is given $g, G, (X, Y, Z)$ where $g^x = X$, $g^y = Y$ and $Z = g^{xy}$ or g^z
  A is ElGamal adversary (INDCPA) is given $pk = g^x$; so let's give X.
  and use g, G same.
  Assume A makes a single query against E.G. → M_0, M_1 and it gets $CT = Y$, $W = g^y, g^{xy} \cdot M_b$ (which B has to now give A) Say B picks random b then B gives $(Y, Z \cdot M_b)$ to A → would be correct pair if B is given $Z = g^{xy}$. NOW A returns its guess for bit b (say d)
  If A is successful, B assumes $Z = g^{xy}$ so experiment is real.
  [See video for B's algo] i.e. if b=d return 1 else return 0

$$Adv^{DDH}_{g,G}(B) = Pr[Exp^{DDH1}_{G,g} \Rightarrow 1] - Pr[Exp^{DDH0}_{G,g} \Rightarrow 1] = P_1 - P_2.$$

P_1 = Prob of returning 1 in real → This experiment is same as cpa-cg and A receives valid CT etc. from B ⇒ $P_1 = Pr[Exp^{cpacg}_{EG}(A) \Rightarrow 1]$

$$= \frac{1}{2} + \frac{1}{2} Adv^{INDCPA}_{EG}(A_1)$$

P_2 = Prob of returning 1 in random → A was given a CT which is random since g^z is a random group element i.e. A was given 2 random gp ele.

⇒ $P_2 \leq 1/2$ ⇒ $Adv^{DDH}_{g,g}(B) \geq \frac{1}{2} Adv^{INDCPA}_{EG}(A)$

Resources: Time to run A + Group multiplication i.e. Quadratic time

* ElGamal is not INDCCA: Adversary $A^{Ex(LR(\cdot,\cdot,b)),D_K(\cdot)}$ with $pk = X$
  First query oracle for any $M_0, M_1 \in G$ $M_0 \neq M_1$ create:

is hard, then RSA is hard. Do not know if reverse is true. i.e. don't know if RSA can be inverted without factoring but factoring has to be hard for RSA to be hard.

Naive factoring algo: for $i = 2$ to $\sqrt{N}$, if $N \bmod i = 0$ $p \leftarrow i$ $q \leftarrow N/i$ return p, q

– correct but takes $O(\sqrt{N}) = O(e^{0.5 \ln N})$ – prohibitive

- Factoring algorithms with running times:

Naive	$O(e^{0.5 \ln N})$
Quadratic Sieve	$O(e^{c(\ln N)^{1/2}(\ln \ln N)^{1/2}})$
Number Field Sieve	$O(e^{1.92(\ln N)^{1/3}(\ln \ln N)^{2/3}})$

- Though factoring and DL are very different problems, similarity b/w the best known algo for DL and NF Sieve ⇒ still slow

  By 2009, were able to factor RSA-768 (with time) using NFS.

- Recommended Size: For 80 bit of security, use RSA modulus 1024b

  - Factoring 2^{80} time → seems out of reach at present to factor / invert

  Obviously larger numbers mean RSA efficiency is lower.

- Plain RSA Encryption:

  KeyGen K: $(N, p, q, e, d) \xleftarrow{\$} K_{rsa}$; $pk \leftarrow (N, e)$, $sk \leftarrow (N, d)$; return pk, sk

  Encryption $E_{pk}(M)$: $C \leftarrow M^e \bmod N$; return C

  Decryption: $D_{sk}(C)$: $M \leftarrow C^d \bmod N$; return M

  $M \in \mathbb{Z}_N^* = Msg\$$

  From RSA assumption & math → hard to recover PT

  But Plain RSA is NOT IND-CPA! ∵ It is deterministic !!!!

  More attacks:

1) If $N = pq$ and $q < p < 2q$ (p, q prime) and $d < \frac{1}{3} N^{1/4}$ then given N, e can efficiently compute d. → Secret key should be large enough - can be ensured by RSA keygen algo

2) Hastad's broadcast attack for RSA with low public exponent:

   Say sender sends message ($= m$) to 3 different recipients with 3 different public keys and with low public exponent, say 3.

   Then $c_1 = m^3 \bmod N_1$, $c_2 = m^3 \bmod N_2$, $c_3 = m^3 \bmod N_3$.

   Attacker can compute m from c_1, c_2, c_3 using Chinese Remainder Theorem

$C = m^3 \bmod N_1N_2N_3$. Since $m^3 < N_1N_2N_3$ $\quad m \leftarrow \sqrt[3]{C}$

* RSA is one way for 1 recipient but not for 3 recipients!!

- Fixing these issues - Assumed that weakness was that it is deterministic so when RSA was first standardized, randomness was introduced.

PKCS #1: Randomized encryption: Random Padding - Prepended along with fixed padding $\quad 00||02||Pad||00||M \rightarrow$ same message

- when decrypting, can check that fixed padding is what it's supposed to be and reject invalid CT

• IND CPA not strong enough: RSA as used in PKCS#1 was broken in practise as part of previous SSL protocol

1998 - Daniel Bleichenbacher:

- User picks session key and encrypt using PKCS#1 → Server sends Ack if valid ciphertext
- Attacker takes $CT = C$, modifies it and initiate new session with server initially will fail because padding of C' won't match expected value
  Attacker keeps trying - sometimes OK, mostly not → After million tries
  ⇒ can completely recover session key!

• Details of the Attack: Attacker knew: (N,e), C

- Attacker could send C and get whether valid or not

→ Obviously padded RSA - part of PKCS was not INDCCA.

RSA-OAEP - Optimal Asymmetric Encryption Padding: Designed by BR - 1994

Encryption: Last stage same as applying RSA function

Preprocessing Message: Use 2 hash functions G, H → $M||0^n||\$$

* Message viewed as bitstring, not a number.

Part 1 = $G(\$) \oplus M||0$ $\quad$ Part 2 = $H(\text{Part 1}) \oplus \$$ $\quad$ $C = RSA_{N,e}(\text{Part 1}||\text{Part 2})$

* View this as a number $\in \mathbb{Z}_N^*$ (though it is actually bitstring)

- G is not a compressing function!

Decryption: Reverse process → Note: OAEP transform - can always be inverted without any keys

- Decryption checks if padding all zeroes, then o/p M, else $\perp$

.... can check correctness.

- used in practical protocols: SSL/TLS, SSH, Part of various standards.

- Security: IND-CCA secure assuming RSA is one way and G, H are random oracles i.e. INDCCA secure in random oracle model if RSA problem is hard

* RSA OEAP not proven INDCCA in general.

- Random Oracle Model - Assumes all parties including attacker have access to a truly random function which they use to compute hash
  - Problem is not assuming random looping o/ps (which hashes do have) but the public oracle assumption (In reality, can compute locally)
  - Belief that security in RO model will hold in standard model But many examples of uninstantiable schemes: proven secure in RO model but insecure for ANY instantiation of random oracles with a real function
    → But most of these are artificial schemes, hence belief of security
- Note: Though Cramer Shoup proven secure with only RSA assumption vs RSA-OAEP only secure under RO assumption, RSA-OAEP used more widely because of efficiency (C.S - several mod exp, RSA-OAEP-1)

MODULE - 15 - HYBRID ENCRYPTION

- How to use RSA to send actual data? RSA only allows us to encrypt somewhat short messages - E.g: RSA1024 - Message ≈ 700b + random + padding
  - Option 1: Encrypt block by block similar to ECB (Not Secure)
    → Do not do this in pk setting → Not efficient at all!
- Solution: Hybrid Encryption -- Combination of private & public K.E
  1) Choose random symmetric key K using AE algo
  2) Encrypt message under K using SE algo
- Key for SE should be in Message Space of AE
- Syntax For $AE(K^a, E^a, D^a)$ and $SE(K^s, E^s, D^s)$, then $\overline{AE} = (K^a, \overline{E}, \overline{D})$:

$\overline{E}_{pk}(M)$: $K \xleftarrow{\$} K^s$, $C^s \leftarrow E^s_K(M)$; If $C^s = \perp$ return $\perp$, $C^a = E^a_{pk}(K)$, $C \leftarrow (C^a, C^s)$; Return C

$\overline{D}_{sk}(C)$: $(C^a, C^s) \leftarrow C$; $K \leftarrow D^a_{sk}(C^a)$; If $K = \perp$, return $\perp$; $M \leftarrow D^s_K(C^s)$; Return M

- Correct if base encryption schemes are correct.
  - Syntactically an asymmetric encryption scheme (NO shared

key apriori, E_k takes a pk input and D_k takes secret key input)

- Security: if components are IND-CPA/IND-CCA, then asso. scheme is also IND-CPA/IND-CCA respectively
  - We don't actually need IND-CPA/IND-CCA for the SE → New key for each new message → one time security is sufficient. But because practical SEs are efficient and secure, can use those & even repeat key for some messages.

MODULE - 16 - ENCRYPTION IN THE MULTI USER SETTING

- Definitions of security so far (IND-CPA/IND-CCA): Considered a single receiver with a single public key → Different from practice!
  → Real life is multiuser setting → Are these schemes secure in this setting as well
  E.g.) Hastad's attack on plain RSA: though one way in single user setting, not so in multi user. (i.e. can recover PT in multiuser :)
2) RSA-OAEP: Nothing leaked in single user setting ....
3) General question: Are schemes provably secure in single user setting still secure in the practical setting
   → for good definitions of security, single user security ⇒ multi user security
   INDCPA/INDCCA etc.
  - To show this need a new defn of security → Attacker observes several senders/receivers → Messages may be related to each other.
- IND-CPA Security in Multi User Setting:
  Difference between standard defn: 1) Generate a number of (pk, sk) from Keygen initially and give attacker all public keys
  2) Instead of one oracle, give attacker n oracles for same challenge bit b
  - Attacker can query oracles in any order
  - " can query adaptively + can depend on previously seen CTs etc.

  Advantage of attacker = Pr [A⇒0 in n-indcpa0] - Pr[A⇒0 in n-ind-cpa1]
  ... Secure if ... polynomial resources.. negligible in security parameter
- IND CCA for multiuser: Give attacker n decryption oracles. +

- Attacker not allowed to query decryption oracle on c returned by corresponding encryption oracle

- General case - Can be proved that:
  IF scheme is secure in single user setting, then secure in practical multiuser setting
  For $\mathcal{AE}$ asymmetric encryption scheme with adversary $\mathcal{A}$, $\exists$ adversary $\mathcal{B}$ with similar running time who does only one query to LR en. oracle.

$$Adv^{nindcpa}_{\mathcal{AE}}(\mathcal{A}) \leq n \cdot q_e \cdot Adv^{indcpa}_{\mathcal{AE}}(\mathcal{B})$$

  n = number of users, q_e = number of queries allowed in multiuser

  - Same stmt applies for INDCCA

- Asymptotically: secure in single then also secure in multiuser.

- Significance of the $n \cdot q_e$ → security is preserved but degrades as we add more users

- Need for concrete security improvements:
  Consider PKE Scheme with INDCPA advantage of adversary $< 2^{-60}$
  But say no. of users = 2×10^8 (200 million - not unrealistic) $q_e = 2^{30}$ under each pk
  → $n \cdot indcpa\ adv \leq 0.2$ → Obviously not good!

- Is it possible to have a better reduction than this: $\exists$ encryption scheme (maybe artificial) for which security drop is exactly $n \cdot q_e$. → cannot be secure in general, but can have better security stmt for specific schemes.

- Improved reduction for Specific Schemes:
  From general reduction: $Adv^{nindcpa}_{EG}(\mathcal{A}) \leq n \cdot q_e\, Adv^{indcpa}_{EG}(\mathcal{B})$

  ElGamal: Can show that for any A in multiuser, there is a B in single user that makes only 1 query → As secure in multiuser as single user

$$Adv^{nindcpa}_{EG}(\mathcal{A}) \leq Adv^{indcpa}_{EG}(\mathcal{B})$$

RSA-OAEP: No known improved security definition*

Cramer Shoup: Reduction possible but only one of the terms - still better than general

* Should be aware of potential security loss when scheme used in practice
  → addnal motivation to use larger modulo for RSA in practice

Encryption Scheme Variants

- Identity based Encryption (IBE): sender needs to look up pk of receiver usually. Here sender uses arbitrary string (like email or URL) -

Receiver gets secret key from central authority → generate sk for identity of receiver

→ Obvious Disadvantage: Central authority must be trusted + has to know secret key of every party

→ Attractive because arbitrary strings can be public keys

- Described long ago by Shamir but no specific construction known till 2001 when Boneh and Franklin designed one.

- Attribute Based Encryption: Secret key and CT associated with attributes (age, title, citizenship, etc.) - Decryption only possible if attr(receiver key) matches attr(CT) → Only parties with matching attr can decrypt
  → Useful for access control

- Homomorphic Encryption: Allows computing on encrypted data. i.e $E_k(x)$ can be modified to be $E_k(f(x))$!!!
  - Different classes of H. Schemes depending on which operations are supported
    → Additively HE, Multiplicatively HE
    → Recently discovered - fully homomorphic i.e any arbitrary fnf
      - Not practical, theoretical right now.

- Searchable Encryption

MODULE - 17 - DIGITAL SIGNATURES

- Goal: Authentication and Integrity in public key setting i.e. Receiver assured that message from legit. sender and not modified.
  - Solution for sym key setting: MACs For asym key setting: digital signatures
- Syntax: DS: (K, Sign, VF) associated with Msgsp(pk)
  - Message space can be associated with pk
  - K: Randomized algo, takes security parameter as i/p, o/p = (pk, sk)
    : Run by sender i.e. sk kept by sender
  - Sign: Run by sender, i/p: msg, sk, o/p: Signature - Need not be randomized, sometimes it is
    [Assume that receiver gets both sign and message here]
  - VF: Run by receiver, i/p: msg, sign, pk of sender o/p: bit indicating valid or not
- ° Correctness: For every M ∈ Msgsp and every (pk, sk) from K, if σ is output from Sign:

$$VF(pk, M, \sigma) = 1$$

- * - Signing algo can be randomized or stateful (or neither)
  - Msgsp may depend on pk, but often is $\{0,1\}^*$ for every pk
  - Obviously key usage is reverse - anyone can verify but not send (In encryption, anyone can encrypt, but not decrypt)
- Difference with MACs: Non-repudiation -

In MAC: verifier needs to share secret with sender ⇒ They can impersonate sender
Not possible in digital signature scheme ← Because verifier has no secret

- Security - Intuition similar to MACs - Attacker cannot produce legit forgeries even after seeing valid signatures.

UF-CMA for DSA: Fix DSA = (K, Sign, VF) - Adversary is given: 1) Public key
2) Signing Oracle (similar to MACs) - Attacker can observe valid signs for messages of its choice
- No verification oracle!!! → Not needed because pk is public!

UF CMA Advantage: $Pr[VF(pk, M, \sigma) = 1]$ for M ∈ Msgsp not queried to Signing Oracle
- Attacker o/ps M, σ → forgery
- Attacker should have polynomial running time* in security parameter (* or resources) and advantage is negligible in security parameter.

Plain RSA Signature Scheme - Referred to as plain or basic RSA signature

Key Generation: Same as RSA → Generates parameters: (N, e) (N, p, q, d)

- Messages are from $\mathbb{Z}_N^*$
- Sign (M): Applies RSA^{-1} i.e. $x \leftarrow M^d \bmod N$ Return $\perp$ if $M \notin \mathbb{Z}_N^*$
- VF (M, x): If $M \leftarrow x^e \bmod N$ return 1 else 0 Return 0 if $M \notin \mathbb{Z}_N^*$ or $x \notin \mathbb{Z}_N^*$

· Security: Deterministic but this is not a problem for Sign. schemes.

- Plain RSA is not UF-CMA:

1) Consider $M = 1$, $x = 1$ → Forger simply returns (1,1)

2) Can get M from σ since e is public → Forger simply does $x \xleftarrow{\$} \mathbb{Z}_N^*$

$M \leftarrow x^e \bmod N$; Return (M, x)

→ But these may not be considered practical attacks

3) Using multiplicative property of RSA i.e. $(M_0^d \bmod N) \cdot (M_1^d \bmod N) = (M_0 \cdot M_1)^d \bmod N$.

- Attacker chooses 1 message to forge
- Attacker chooses 1 random message to query signing oracle:

$M_1 \xleftarrow{\$} \mathbb{Z}_N^* - \{1, M\}$; $M_2 \leftarrow M \cdot M_1^{-1} \bmod N$

$\sigma_1 = Sign(M_1)$; $\sigma_2 = Sign(M_2)$

$\sigma = \sigma_1 \sigma_2 \bmod N$

return (M, σ)

→ All these attacks are efficient and advantage = 1

DH Signatures → Conversion from p.k. encryption i.e. use decryption as signing

Verification: Same as encryption.

$S(sk, M) = D(sk, M)$

$V(pk, M, \sigma) = 1$ iff $E(pk, \sigma) = M$.

Works only if encryption scheme is deterministic

→ Obviously still doesn't work security wise

- Basically using trapdoor permutations.

Hash then Invert Paradigm Idea: Hash the message first

Goal: RSA based signature scheme with a more flexible message space, is provably secure and resists the attacks above.*

Full domain Hash (FDH) RSA Signature Scheme:

- Uses a hash function $H: \{0,1\}^* \to \mathbb{Z}_N^*$. $DS = (K_{rsa}, Sign, VF)$
- KeyGen: Same as plain RSA.
- Signing and VF - similar. Main difference: Apply hash to message before signing

Sign (M): $y \leftarrow H(M)$; $\sigma \leftarrow y^d \bmod N$; return σ

VF (M): $y \leftarrow H(M)$; $y' \leftarrow \sigma^e \bmod N$; IF $y = y'$ return 1, else return 0

* Why: Because unlike pke which we use for sym key, signing used for msg itself

- **Security:**
  - If hash is collision resistant and destroys algebraic structure of message, previous attacks don't apply. [1: algebraic 2 CR 3: mult · no longer applies]
  - To prove security, need to assume the hash function is a random oracle
    → Not realistic assumption but commonly used.
  - Concrete Security Statement:

* Under the RSA assumption, the FDH RSA sign. scheme is UF-CMA in the random oracle model

For K_{RSA} an RSA generator if $\mathcal{F}$ is an FDH RSA adversary making at most q_H queries to the random oracle and q_S queries to the signing oracle $\exists$ adversary I with comparable resources s.t.:

$$Adv^{uf\text{-}cma}_{FDH\,RSA}(\mathcal{F}) \leq (q_S + q_H + 1) \cdot Adv^{ow\text{-}cda}_{K_{RSA}}(I)$$

Intuition:

- Build I who uses $\mathcal{F}$ as a method -- I is given $(N, e, y = x^e \bmod N$ for $x \xleftarrow{\$} \mathbb{Z}_N^*)$
  I is obviously not given d.
- $\mathcal{F}$ is given (N, e) for the signature (we will give same N, e) + Signing oracle + Random hash oracle (returns random $H(M)$) → returns a forgery on a new message
- $\mathcal{F}$ outputs (M^*, σ) at the end. For this, $\mathcal{F}$ must obviously query H to get $H(M^*)$ (But I can actually see M^* in that case) and I can return y as the $H(M^*)$.
  Then when $\mathcal{F}$ computes forgery, $\sigma = H(M^*)^d \bmod N = y^d \bmod N = x$. I might simply output $\sigma = x$ as its own output.
- Note that if $\mathcal{F}$ directly queries the signing oracle, I won't know what to return But note that M^* has to be distinct from any message queried to the signing oracle!!! So I never has to give that!
- But what about other queries to signing and hash oracle? E.g. I has to return valid signature or $\mathcal{F}$ may verify and complain invalid signature [$\mathcal{VF}$ is public!] → Problem: even if H is just a random function, need secret key to sign!

Trick: If we know signature, can compute message for the signature!!!
Output random (new) number for signature, then compute corr $y = H(M)$ to output from hash oracle! i.e. return $H(M) = s^e \bmod N$!
Can even keep these answers ready in advance..

Note that I only succeeds if $\mathcal{F}$ returns a successful forgery. In fact, I is not even successful if $\mathcal{F}$ is always successful!?!

It only works if the forgery matches $H(M)$ I think!?!

- Choosing size of the modulus:

  To have 80 bits of security, we need adv at most 2^{-80}. For RSA alone* we need a 1024 b modulus. But because of the (q_S+q_H+1), need a larger modulus i.e. 3700 bits for N - Not used in practise because not efficient

Probabilistic Signature Scheme (PSS) - Variant of FDH - Uses randomness.

Sign(M): $r \xleftarrow{\$} \{0,1\}^s$ $y \leftarrow H(r \| M)$ $x \leftarrow y^d \bmod N$ Return (r, x)

VF(M, σ): $\sigma = (r, x)$ where $|r| = s$ $y \leftarrow H(r \| M)$ if $x^e \bmod N = y$ return 1 else 0

- Why use this at all: Security does not degrade, security guarantee is better
  ⇒ can use fewer bits (i.e more efficient) of RSA modulus for same security parameter
- Security Statement:

$$Adv^{uf\text{-}cma}_{PSS}(\mathcal{F}) \leq Adv^{owkea}_{RSA}(I) + \frac{(q_H-1)\,q_S}{2^s}$$

  No longer have multiplicative factor!
- Choosing size of modulus:
  - No attacks showing FDH is less secure than RSA (or) FDH with 1024 b modulus has <80 b security
  - For provable security, need larger modulus or PSS
- PSS part of many standards including latest PKCS

RSA PKCS#1 : Practical Scheme

- Assume RSA-1024 - Practical hash is 256b or so. [$H: \{0,1\}^* \to \{0,1\}^n$]
  - Soln: Prepend H(M) with publicly known padding → Fixed padding || HashID || H(M)
- Obviously if H is collision resistant, PKCS-HASH also is.

* More importantly, hardness of computing RSA inverse on a random point not same as hardness of ... In $S = \{PKCS\text{-}HASH(M) : M \in \{0,1\}^*\}$
  → Points are coming from a smaller subset (much smaller!)
  - No known practical attack, but definite room for concern.

- Better Solution:

  H(M) → first n bytes of SHA-3(1||M) || SHA-3(2||M) ... SHA-3(l||M)

- Not used in practice because obviously less efficient.

- **ElGamal with hashing:**

$G = \mathbb{Z}_p^* = \langle g \rangle$ for prime p $H: \{0,1\}^* \to \mathbb{Z}_{p-1}$ a hash function

$pk = X = g^x \in \mathbb{Z}_p^*$ $sk = x \xleftarrow{\$} \mathbb{Z}_{p-1}$

$S_x(M)$: $m \leftarrow H(M)$ $k \xleftarrow{\$} \mathbb{Z}_{p-1}^*$ $r \leftarrow g^k \bmod p$ $s \leftarrow (m - xr) \cdot k^{-1} \bmod (p-1)$

* Return (r, s)

$V_X(M, (r,s))$: $m \leftarrow H(M)$ if $r \notin G$ or $s \notin \mathbb{Z}_{p-1}$ return 0

if $X^r \cdot r^s \equiv g^m \bmod p$ return 1 else return 0.

Here: $X^r \cdot r^s = g^{xr} \cdot g^{k(m-xr)k^{-1} \bmod (p-1)} \bmod p = g^{xr + (m-xr) \bmod (p-1)} \bmod p$

$= g^m \bmod p$!

- **DSA:** (Also DL based)

Fix p, q: $q \mid p-1$; $G = \mathbb{Z}_p^* = \langle h \rangle$ and $g = h^{(p-1)/q}$: g has order q in G

$\Rightarrow g^q \bmod (p-1) = 1$

$H: \{0,1\}^* \to \mathbb{Z}_q$ a hash function ; $pk = X = g^x \in \mathbb{Z}_p^*$ $sk = x \xleftarrow{\$} \mathbb{Z}_q$

$S_x(M)$: $m \leftarrow H(m)$ $k \xleftarrow{\$} \mathbb{Z}_q^*$ $r \leftarrow (g^k \bmod p) \bmod q$ $s \leftarrow (m + xr) \cdot k^{-1} \bmod q$

Return (r, s)

$V_x(M, (r,s))$: $m \leftarrow H(m)$ $w \leftarrow s^{-1} \bmod q$ $u_1 \leftarrow mw \bmod q$ $u_2 \leftarrow rw \bmod q$

$v \leftarrow (g^{u_1} X^{u_2} \bmod p) \bmod q$

if $v = r$ return 1, else return 0.

- **Correctness:** $(g^{u_1} X^{u_2} \bmod p) \bmod q = g^{mw \bmod q} \cdot g^{x \cdot rw \bmod q} \bmod p \bmod q$

$= g^{[w \cdot (m + x \cdot r)]} \bmod p \bmod q = g^{[k(m+xr)^{-1}(k^{-1})^{-1} \cdot m + xr]} \bmod p \bmod q$

$= (g^k \bmod p) \bmod q = r$.

- DSA works only over group of integers but a version of DSA = ECDSA works on elliptic curve groups - Popular scheme. (used in FIPS 186)
- Conveniently, the modular exponentiation above can be done offline.
- No proof of ElGamal UFCMA under standard assumptions, even if H is RO. Proofs exist for variants of ElGamal.
- Security of DSA/ ECDSA proven in 2016 under DL and RO assumptions

Efficiency comparison for same level of security

Scheme	Signing Cost	Verif Cost	Sign size
EG	1024b exp	1024b exp	2048b
	160b "	160b "	320b

DSA obviously more efficient - EC also makes it more efficient [6-4 times - DSA]

Schnorr Signatures Also DL based

$G = \langle g \rangle$ of prime order p ; $H : \{0,1\}^* \to \mathbb{Z}_p$ hash function

$pk = X = g^x \in G$; $sk = x \xleftarrow{R} \mathbb{Z}_p$

$Sig(M)$: $r \xleftarrow{R} \mathbb{Z}_p$ $R \leftarrow g^r$ $c \leftarrow H(R \| M)$ $a \leftarrow xc + r \bmod p$ Return (R, a)

$V_X(M)$: If $R \notin G$ return 0; $c \leftarrow H(R \| M)$ If $g^a = R X^c$ return 1, else 0.

Correctness:

$R \cdot X^c = g^r \cdot g^{x \cdot H(R \| M)} = g^{r + xc} = g^a$

Efficiency: As efficient as ECDSA when implemented in 160b ECG

Security: Proven secure in RO model under DL assumption - security reduction is loose i.e. guarantees not strong under these parameters.

Signature Scheme Variants

Multisignature: Several signers create sign on single message Faster and shorter than collection of all individual signatures

Application: Blockchain, Digital currencies

Aggregate Signature: similar to above, but sign different messages - still faster than old schemes

Threshold Signature: splitting secret key and secret signing among several users - group of n users hold single public key (or can be 1 user with n computers, etc) - Each user holds a share of the private key → At least t users need to cooperate to produce a valid signature of a message [Here both key and signing are split]

Group Signature: Group of users hold single pk. Each user can sign on behalf of group and remain anonymous except to group manager [add/revoke users] - Manager can revoke anonymity of the user.

Applications: Autonomous vehicles - Emergency v. can send signals to other vehicles - authenticity with anonymity (like don't want others to impersonate but also shouldn't be able to track sender

Ring Signature: Similar to group signatures without a group manager - Noone can revoke anonymity + can join/leave without a manager

Blind Signature: Any user can obtain a sign. on a message of its choice from a signer, without signer knowing what they signed i.e. the message

Applications: 1) To strengthen passwords 2) David Chaum (who proposed it)

- digital cash issued by a bank: coins - random strings signed by bank → bank's signature public so everyone can verify it's valid, but when coins come back to bank, bank wouldn't be able to trace where users shopped etc.

- Multisignature scheme Example: (DL Based)

Assume an efficient algo $\mathcal{V}_{DDH}(g, g^x, g^y, g^z) = 1$ iff $g^z = g^{xy}$ i.e. $z = xy \bmod |G|$
i.e. can solve DDH problem with prob 1

Regular scheme: $sk = x \in \mathbb{Z}_{|G|}$, $pk = g^x$; $Sign_x(M) = H(M)^x$

$\mathcal{V}_{pk}(M, \sigma) = \mathcal{V}_{DDH}(g, pk, H(M), \sigma)$ → if algo o/p 1, valid sign

$H(M) = g^y$ for some y

Multisign: Say 3 users $(x_1, g^{x_1}), (x_2, g^{x_2}), (x_3, g^{x_3})$

In regular scheme, each user computes $\sigma_1 = H(M)^{x_1}$, $\sigma_2 = H(M)^{x_2}$, $\sigma_3 = H(M)^{x_3}$

Multisignature $\sigma = \prod_i \sigma_i = H(M)^{\sum x_i}$

$$\mathcal{V}(M, \sigma_1, \sigma_2, \sigma_3) = \mathcal{V}_{DDH}(g, \prod_i pk_i, H(M), \prod_i \sigma_i)$$

Here $\prod_i pk_i = g^{\sum x_i}$ $\quad \prod_i \sigma_i = H(M)^{\sum x_i}$

In a standard signature scheme, would need n group elements for σ, instead of 1
Also can do 1 verification (expo.) instead of n. instead we do n mult, but that's faster

→ There are groups where this is possible - ECG with bilinear variance

- Blind Signature Example: (RSA Based)

Signer has secret key S → $sk = d$ $(pk = (N, e))$.

User u has message on which need sign from signer → $R \xleftarrow{\$} \mathbb{Z}_N^*$

u sends $H(M) \cdot R^e \bmod N$ to the signer $= M_u$

S - use plain RSA and return. $M_u^d \bmod N = H(M)^d \cdot R \bmod N = \sigma_s$

u then multiplies σ_s with R^{-1} → $H(M)^d \bmod N$ - standard form RSA sign scheme

- Can be shown to have 2 properties:

1) Blindness: Signer has no info about message inside - Signer only sees $H(M) \cdot R^e$ for a random R, so looks like random num to S independent of M
Also known as Anonymity

2) Can also show that this signature scheme is Unforgeable

- SIGNCRYPTION! - Public key primitive to achieve privacy, integrity and authenticity of transmitted data.

- Similar to Authenticated Encryption in private key setting
- Cannot simply use an asymmetric ES with additional properties because of PK. i.e. Receiver has (pk, sk) for privacy but for auth/int - sender has (pk, sk) ⇒ Both sender and receiver must have key pair
- Must be considered in 2 user (or multi user setting)

Security

- Can define IND CCA (privacy) and INT-CTXT (authenticity) ||| to AE
- have to consider 2 user or multi user (including possibility of identity fraud)
- Again, how to compose AE (asym) + DS scheme to get secure signcryption

Security results

- If AE is IND-CPA and sign. scheme is SUF-CMA, then Encrypt then Sign is IND CCA and INT CTXT in 2 user model
  - SUFCMA - stronger than UFCMA, ≡ UFCMA for det. schemes, satisfied by all practical schemes
- For Multiuser Security: same notion holds but
  - Add pk of sender to message to encrypt
  - Add pk of receiver to message to sign
- Encrypt then Sign: Encrypt, then sign the CT
- SUF CMA: Det like FDH, Rand: PSS, DSA, Schnorr - proven secure under SUF-CMA

MODULE - 18 - SECRET KEY SHARING

- Public key infrastructure:

For asymmetric crypto, assume public keys are public, authentic and tied to user's identities, corresponding secret key is secret and not compromised
⇒ Assured by PKI

- To make PKI work, need a trusted 3rd party - i.e. a CA or Certification Authority
- Public key Certification Process:

Assume pk of CA is known to everybody and is authentic (How can we assume this? Have to start 'somewhere!) - User U tries to register its key pk_U.

1) U presents to CA User ID_U, pk_U,
2) CA verifies ID_U (depends on auth. of comm. channel) ⇒ CA checks that user knows sk for pk ⇒ Prevent rogue key attacks: malicious user

registers pk based on pk of real user

3) CA picks random R challenge - message to sign (or random C to decrypt)

4) User completes challenge - sends signature $\sigma = Sign(SK_u, R)$

5) CA verifies response using pk → IP valid:

6) CA issues digital certificate on pk → Signature computed using CA's sk on message containing (ID_u, pk_u, exp.date...)

*) User can verify certificate using pk_{CA} then if valid, keep cert.

- Where to get pk_{CA} from:
  - Embedded in s/w like browser.
  - PKI ensures public keys and cert. are public → important for encryption → senders look up pk of receivers
  - For sign; public key and cert could be added to sign. but can also be av. to PKI.
- IP secret key gets compromised:

  PKI maintains Certificate Revocation List (CRL) - contains bad certificates
  - On security breach, user supposed to let CA know - get new pk certified and get old one added to CRL
- Revocation Issues:

  Say sk is compromised, there may be sometime btw compromise and when cert was revoked → users may be treating this cert as valid in that period → Signatures created in this time not guaranteed to be authentic (or data not private)

Online Certificate Status Protocol (OCSP) - Alternative to CRLs

Enables online checks of whether or not cert has been revoked - allows user to simply query with CA instead of consulting a list

→ CA has to be online & has to reply → kind of defeats the purpose of pk crypto

- Revocation in Practice:
  - 8-20% estimated to be revoked ⇒ CRLs are very large

→ Revocation big problem holding up widespread deployment of PKI and pk crypto

- Can have many CAs, hierarchical CAs - upper level CAs certifying lower level CAs
  - Easy for lower level CA's to check identities
  - Only need the root CA's public key for verification.

X.509 - standard for digital certificates developed by Int. Telecom Union

$M \leftarrow D_x(Y, W \cdot g)$

Here, $Y = g^t$ $W = g^{xy} M_b \cdot g$ if $M = M_0 \cdot g$ then return 0, else return 1

$Adv^{INDCCA}_{EG}(A) = 1$

Resources = $t_A = O(|k|^2)$ $q_e = 1$ $q_d = 1$ $\mu_e = 2k$ $\mu_d = 2k$

⇒ For any group, ElGamal is not INDCCA secure

Cramer-Shoup Encryption (1998) Uses 2 generators; 5 exponents and public key has more group elements.

Encryption/Decryption* involve more group exponentiations

⇒ Definitely less efficient than ElGamal

* Also include a hash function.

- Scheme is INDCCA secure if H is a CR hash function and DDH is hard.

→ Not practically used (more efficient INDCCA secure schemes available)

-

→ DDH is hard ⇒ CDH is hard ⇒ DL is hard

But DDH is easy ⇏ CDH is easy ⇏ DL is easy.

Also DL is easy ⇒ CDH is easy ⇒ DDH is easy.

But DL is hard ⇏ CDH is hard ⇏ DDH is hard!

→ for DL- x is picked from $\mathbb{Z}_m$ [Resources = running time. No oracle query]

- Attacker knows g, G, m. $z = xy \bmod m$

- Real DH Tuple vs Random DH Tuple (X, Y, Z)

- DDH is easy in $\mathbb{Z}_p^*$. Currently no better mtd for CDH other than using DL

- ElGamal: Alice receiver. Bob sends $Y = g^y$, $K = X^y = (Alice_{pk})^{bob's\ sk}$. Alice: $Y^x = Y^{sk}$

  Msg sp = G. → To find sk from pk: DL. To find M from C, CDH.

  → Not INDCPA in $\mathbb{Z}_p^*$ → INDCPA where DDH is hard.

- If DDH is hard → EG is INDCPA in that group

  If EG is not INDCPA → DDH is not hard in that group

-

- Relied on downgrading to 512b + weak DH groups - 7) Timing and Side channels - power analysis (recover key), acoustic cryptanalysis, electromagnetic attacks, cache side channels

[Last but one lecture] MPC - Millionaires problem, Lattice based problem - for quantum

- Certificate Use Example:

Browser and server establish a secure channel i.e. set a session key for auth. encryption → First, browser checks auth of server's pk → can view cert & CA

- Pretty Good Privacy (PGP): PK Crypto without CA - Encryption for emails.

- Instead of CAs, have a web of trust - Get A's pk from B, then trust A dep on how much you trust B. → Requires user involvement

Secret key sharing - Obviously, sec. of both sym & asym relies on this

- How to keep secret key secret: Split secret key and store shares in n diff. computers so that: any t shares allow you to reconstruct K

[Why is this good: Now attacker has to compromise n computers instead of one]

→ If t-1 computers compromised, attacker should still not be able to find K

(Obviously when t are compromised, key will be lost)

(t,n) = t out of n secret sharing scheme

- Example: (2,2) secret sharing - 128b key K

- If we simply split in half, correctness works but not security (can get 1/2 of sk)

- Goal is any single share shouldn't give you any info about the sk

→ Better soln.: Keep K_1 = random n bit string, Then $K_2 = K \oplus K_1$

Obviously can get K from $K_1 \oplus K_2$ → K_1 and K_2 appear random - so good.

→ How to improve for n,n: Pick (n-1) random shares.

- Unfortunately does not work for (t,n) schemes

- How to build (t,n) secret sharing scheme:

Say, we are trying to share α which lies b/w 0 and 90 → Choose any points on line that makes $\angle\alpha$ with x-axis.

→ Any 1 point insufficient to get α, but 2 points sufficient.

For t=3, use parabola ... To increase t, increase dim of equation.

- Shamir's secret sharing scheme:

p = large prime, To (t,n) secret share $z \in \mathbb{Z}_p$:

- Choose (t-1) random elements from $\mathbb{Z}_p$: $a_1, a_2 \cdots a_{t-1}$
- Set $a_0 = z$
- Construct $f(x) = a_0 + a_1 x + a_2 x^2 \cdots a_{t-1} x^{t-1}$ i.e. poly of degree t-1
- Store $y(i) = f(i)$ for arbitrary i (for convenience, choose i = 1 to n)

- To recover secret, use Lagrange interpolation to find z from t pairs of (i, y_i) for $i \in S$

$$z = a_0 = f(0) = \sum_{i \in S} y_i \prod_{j \in S, j \neq i} \frac{-j}{i-j}$$

- The product term can be precomputed independent of y.
- Unconditionally secure !!!
- Weaknesses: 1) If some parties cheat during secret recon, secret cannot be recovered and cheating cannot be detected by others

Verifiable ss protocol ←

2) Might also want to split secret computn (like decryl signing) → when the step is actually done, if a single computer does computn., and constructs key from all the shares, attacker can compromise it at that point!

Soln: Threshold Cryptography or Threshold Signing - Each party computes share of decryption or sign - legit party combines results, but no shares has all info.

- Diffie Hellman Key Exchange: $G, g, m = |G|$ - public
  - $K = g^{xy}$ can be used as symmetric key for future communication.
  - Attacker gets g^x, g^y and should compute g^{xy} → CDH - presumed hard
  - " computing any info about key (INDCPA) ≡ DDH

→ DH is secure against passive (eavesdropping) attack.

* Even though DDH is hard in $\mathbb{Z}_p$, DH is commonly used in $\mathbb{Z}_p^*$ with some modification i.e. $H(g^{xy})$ is used as the key K → Security from CDH under RO

- Security under MITM attack:

Eve pretends to be Alice and does the exact same key exchange with Bob i.e. Bob sends messages thinking receiver is Alice but it is actually Eve!

→ DH is insecure against active attack

Soln: Not easy! No way for A and B to distinguish each other from adversary.

→ Need some apriori info advantage over the adversary → long term keys

- Long Term Keys: Can be of several types:
  1) Public Key setting: Both parties have known public keys
  2) Symmetric " : Somehow share key K
  3) 3 party setting: each share key with a trusted 3rd party server S
- Session keys and Sessions: Typically comm. involves multiple sessions - 2 parts:
  1) Key establishment protocol based on L.T. keys to share fresh, authentic session key

→ Easier

2) Encrypt and/or auth. data under sess. key for duration of the session - Secure Channel

- Session Key Establishment: Hard to get right
  - Server & client auth i.e. client's $\hat{pk}_{server}$ matches key from server + Key secrecy of session key
  - Attacker may be active
  - In PK Setting: (SSL, SSH, TLS, ..) - Very common that only server has pk - here assume both do.

  Protocol KE 1: Session key K chosen by B. 1) A sends A, R_A (random nonce)

  2) B picks K and sends $(R_B, C, B, Sign(A, B, R_A, R_B, C))$ 3) A can verify sign under pk_B then sends $A, Sign_A(A, B, R_A, R_B)$

  → Can be shown to be secure if $\mathcal{E}$ and Sign are secure.
  - Forward Secrecy of this protocol: say B sends $C_B \leftarrow \mathcal{E}_K(M)$

  If attacker has observed all the comm. prior to a security breach in which, say sk_A becomes known, attacker can get K and all the data encrypted under K

  KE2: Adding fwd secrecy: First A sends A, g^a → B sends $B, g^b, Sign(A, B, g^a, g^b)$

  A sends $A, Sign(A, B, g^a, g^b)$ → Use Session Key = $H(A, B, g^a, g^b, g^{ab})$

  → Since pk, sk only used for signing, not for encryption, sk leak does not allow retrieving old messages (can verify sign. now - revoking prevents this)

i.e. Knowledge of LT key does not allow knowledge of previously exchanged session key

  - Std protocols use DH like this.

- Passwords - Human memorizable key (Usability vs security) → Easy to brute force
  - Can create dict D → usually know some f(pw) for some public fn f. → ↑
  - Usually hashes are stored → Attacker brute forces for $p \in D$
  - Salt does not protect from attacks against a particular user - Att. may also know salt
  - Dictionary Attack
  - Obviously convenient & entrenched - Pwds still in dict - Attackers also get better at making dict
  - 81% of security breaches because of stolen and/or weak pwds.
  - FIDO (Fast ID Online) → working to move away from pwds - base auth on secure devices or biometrics and secure PINs.

- Implementation Issues: 1) Impl. shortcuts (SSL Heartbleed - Buf OF) 2) Security defn not strong enough (Blei..) 3) Wrong primitives 4) Weak randomness - 2007 - NIST - deliberate weakness → trapdoor 5) Reuse randomness 6) Weak bldg blocks not paying attention to concrete security bounds (Long range attack on TLS