CSCE-689 : Network Security

Module 1, lecture 1

Cryptography overview

* (I) Alice & Bob
* (II) Adversary
  - Eve — passive
  - Mallory — Man in the middle, active (tamper communication)
* (III) trusted (TTP) client (third-party, protect comm. b/w A & B)
* Confidentiality using cryptography

Models

Private key

* Sender & receiver share a common (private) key
* ~~(Encrypt)~~ (Decrypt) ≠ (Encrypt) & Dec using the private key
* conventional / shared-key / single-key / symmetric-key

Public key

* Every user has a public key & a private key
* Enc → public-key
  Dec → private-key
  [public-key → everyone has, private-key → only A has (communicate)
* two-key / assymmetric-key

* Alice will use Bob's public key to encrypt msg & send. Bob will use his private key to decript the msg.
* public-key of recipient & private-key of recipient.
* Mostly focus on private key in this module.

* Plaintext → msgs that 2 communicating parties want to exchange w/ each other.
* key → shared / (public-private)
* Encrypt (or cipher) → $f(x)$, obfuscation $f(x)$ [Plain-text + key → $E(f(x))$ → ciphertext]
* ciphertext
* Decrypt → Decryption function (key, decrypt → plaintext)
* Cipher → algorithm used for encrypting / decrypting msgs
* Cryptosystem → mathematical principle used to build cipher
* Cryptanalysis (codebreaking) → breaking the encryption function
* Cryptology: Cryptography + Cryptanalysis
  (writing codes) (breaking codes)

* DES → Encryption algorithm

Open V/s Closed-design

closed-design
- keep the cipher secret
- proprietary design
- Bad practice (?)

(encryption & decryption algorithms are kept secret from the adversaries)

Security → (whether or not the adversary can learn the algorithm)

open-design (Kerckhoff's principle)
- keep everything public, (except the key)
- Good practice

[relies on secrecy of keys & not on algorithms]

* hide the algo, (reveal the algo) → more confidence bcoz algo. known to everyone (can look at it, critique it), open-source software than closed

In open-design

* (I) If keys get leaked down, you can upgrade the keys, refresh, pick new keys, same algo, smooth. (II) talk to multiple people only multiple keys required
* In closed design come up with new algo, not scalable (Scabble)
  → Diff algo if you work with different entities. (not scalable)

Lecture 2

Private key Encryption

main-functions

1) KeyGen = KeyGen(I) (I is a security parameter)
2) Enc : C = Enc(K, M) (K: key, M: plaintext)
3) Dec : M = Dec(K, C)

(O/p is cipher-text, gobbled-msg) also has to be randomized (even if you feed same key & same msg of a different cipher)

* key-generation → I/p is a security-parameter, key length of the key
  randomized-function. output is a key. (output of function) Different everytime.
* M = Dec(K, C)
  (K: key, C: cipher; plaintext: output of key-generation function)
* Dec(K, C) → deterministic function (same key + cipher ⇒ always same plaintext back)
* correctness property → encryption scheme is working properly.

  Enc(K, M) ⇒ C,
  → K, C ⇒ get M same back

* Security property → Goal of attacker (Basic) learn the plaintext
  → capability of attacker

Goals of the attacker

- Learn the plaintext corresponding to a given ciphertext
  — One-way security (I) [plaintext —Key→ ciphertext; ciphertext —✗ Key→ plaintext]
- Extract the key → Key recovery security (II)
- Learn some information about the plaintexts corresponding to a given ciphertext → Semantic security (III)

(encryption scheme does not give out the key)

→ (I) & (II) must, (III) → ideal

Capabilities of the Attacker

1) No information (besides the algo.)
2) Ciphertext only (eavesdropping)
3) Known plaintexts
   ↳ Adversary knows a set of plaintext-ciphertext pairs (or more) [Based on ~~inter~~ inferences (Based on events)]
4) Chosen (and adaptively chosen) plaintext (CPA attack)
   → Adversary choose a number of plaintexts & obtains the corresponding ciphertexts (temporal access to encryption machine, not actual the key)
5) Chosen (and adaptively chosen) ciphertext attack (CCA attack) → Adversary choose a number of ciphertexts & obtains the corresponding plaintexts.
   (decryption machine temporal access)

CPA

CCA

(Capability ↑)

1 < 2 < 3 < 4 < 5

* A cryptosystem secure against 5 is automatically secure against 4, 3, 2 & 1.
* Brute force attacks → key recovery

PREREQUISITE
- atleast 80 bits (more than that) → (atleast 128 bits now)
- really long time to perform brute-force attack

Lecture 3 : Classical ciphers

Historically

Substitution ciphers

plaintext letter —substitute→ other letter

* eg → Caesar's cipher (shifting certain no. of locations)
* polyalphabetic substi. cipher
  - ↳ michener (same letter gets mapped to a diff. letter in cipher text)

→ prone to language characteristic analysis, breaking based on freqs very easy (e letter freq is most)

transposition ciphers

plaintext → letters you rearrange them, letter, transpose them! permute them

---

One-time pad / Binary Vernam cipher

* $C = P \oplus K$, Encrypt
* (xored) / ~~xoring~~ → Decrypt
* cannot know K/P from C (attacker)

* Knowledge of ciphertext does not add any advantage (in deciphering actual P.T) could have done the same guessing with plaintext probability. ($\frac{1}{4}$)

* Complete secrecy !!
  - requirement
  - (1) key as long as msg
  - (2) never use key for encrypting new msg

* length of key / never use key ⟹ not practical
  2GB file, 2GB key, new 2GB file, new 2GB key

Module 1, lecture 4

DES Functioning

* Data encryption standard → modern cipher

Block-ciphers

* partitions P.T into blocks & encrypts each block independently.
* focus on block ciphers
* wait for a block (not on the fly)

Stream cipher

* generates a keystream & encrypts by combining keystream with P.T (eg bitwise XOR)
* on the fly

DES

* encrypts with ~~out~~ series of substitution & transpositions (multiples of both)

* Feistel structure
* DES → AES (Advanced encryption standard)
  ↳ security (smaller length of key), atleast 128 (DES → 56 bit long)
* 16 rounds (several transpositions & substitutions)

permutation → transposition

K_1 – K_{16} (all keys different) potentially

* LHS RS (32-bit)

Initial ⟷ Reverse (at end)

1st → 8th bit ⟷ 8th → 1st bit

* $L_i = R_{i-1}$

  $R_i = L_{i-1} \oplus F(K_i, R_{i-1})$

  series of substitution/transposition

* 8 substitution block [8 bit → 4 bit]
* nearly randomized o/p, no correlation with i/p → Goal of DES

→ Decryption (process)

repeat same computation just reverse key schedule

$(K_1 : K_{16}) \Rightarrow (K_{16} : K_1)$

* avalanche property

DES security : Avalanche effect

* 1.5% change in i/p ⟹ 45% change in o/p because 16 rounds of computation
* Good security (Pseudo-random function)
* looks nearly random if you don't have key !!

Module 1, Lecture 5

DES Security :

* size of key → Insecure (brute-force attack) (2^{56} encryp/decry) → pretty much doable.
* advantage: behaves as pseudo-random but short-key. disadvantage
* brute force → get right-key.
* (1) Super-encryption [Double-encryption]
* key-length → encrypt twice.

$$C = E_{K_2}(E_{K_1}(P)) \quad (K_1 \& K_2 \text{ are two keys})$$

$$P = D_{K_1}(D_{K_2}(C))$$

* encryption will multiple-keys → super encryption
* 2^{56} operations for brute-force attack.
* security-level $2^{56} \times 2^{56} = 2^{112}$ (both K_1 & K_2 required)
* known plaintext, know P & C, guessing K_1 & K_2.
* good idea? → No
* same level of security as single-DES (Just need more memory)

Double-DES :

* Due to meet-in-the-middle attack (due to Diffie-Hellman)

$$C = E_{K_2}(E_{K_1}(P))$$

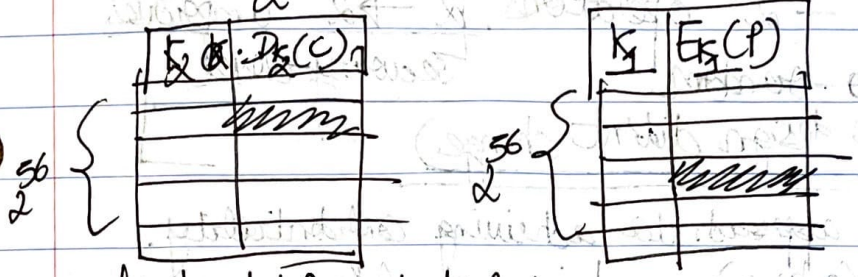
$$X = E_{K_1}(P) = D_{K_2}(C) \quad \text{.. intermediatory eqn}$$

* known P.T attack. known (P, C), find K_1 & K_2 !!
* 2^{56} possibilities for K_1 (Table T)
  K_2 (2^{56} ⁄⁄) (Table again)

LHS	RHS
2^{56}	2^6

matching - values

$$D_{K_2}(C) = E_{K_1}(P)$$



* find match & get K_1 & K_2
* only need 2^{56} operations to build both the tables.
  $2 \times 2^{56} = 2^{57}$ (close to 2^{56}) to find right-key.
* almost same level of security as single DES.

Triple DES ?

1. Triple DES (2 keys) requires 2^{112} search. reasonably secure.
2. Triple DES (3 keys) requires 2^{112} as well (3 Encrypt + 3 Decrypt)

better?

Same level of security for ① & ②

1. Standardized (If $K_1 = K_2$, it is same as single-DES (Backward-compatible).

(1) → 2^{112}
(2) → 2^{112}

(I) Same level of security

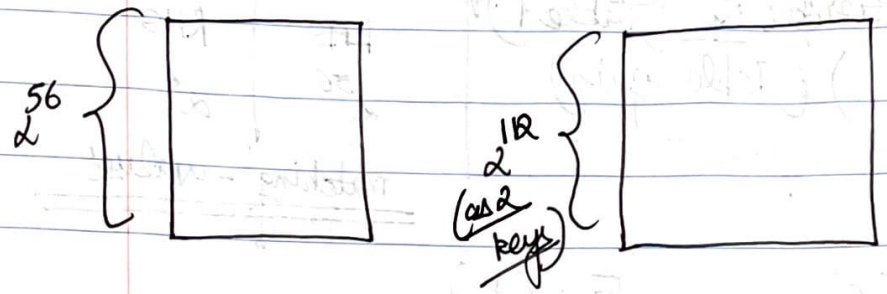
(II) 2 keys ∴ better key management

① is better

meet in the middle (3-DES)

$$C = E_{K_1}\left(D_{K_2}\left(E_{K_1}(P)\right)\right)$$

$$D_{K_1}(C) = D_{K_2}\left(E_{K_1}(P)\right)$$

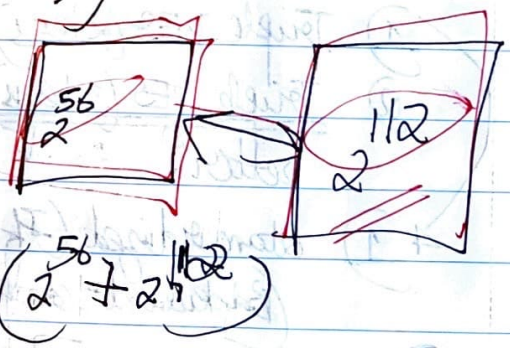


* $2^{112} + 2^{56} \approx 2^{112}$ operations. [$2^{56} \rightarrow 2^{112}$ improves security level]
  + pseudo-random (as DES design didn't change)
* practical approach to achieving confidentiality.
* 3 DES (2 keys) used in actual systems
* close to 2^{128} (2^{112})

$$C = E_{K_3}\left(E_{K_2}\left(E_{K_1}(P)\right)\right)$$

$$D_{K_3}(C) = E_{K_2}\left(E_{K_1}(P)\right)$$

$$D_{K_2}\left(D_{K_3}(C)\right) = E_{K_1}(P)$$



$= 2^{112} \cdot 2^{56} + 2^{112} \cdot 2$

Module 2 : Cryptography II

23/09/23

1.1 * Block cipher modes of encryption (DES → block cipher)

1.2 * other ciphers (briefly)

1.3 * Public key crypto overview

1.4 * Math Background

1.5 * Public key encryption (RSA)

1.6 * RSA security

1.7 * Digital signatures (authentication & integrity using public-key cryptography)

Lecture 1

Block cipher encryption - modes

- Book
  - Electronic Code Book (ECB)
- Cipher Block-chain (CBC)
  - * most popular one & secure

* others → 1) Cipher feed back (CFB), 2) Output feedback (OFB)

Analysis :

1) security, 2) computational-efficiency (parallelizing encry/decry), 3) Transmission-errors

[If some of the blocks were not transferred correctly still you are able to recover some of the blocks]

ECB-Mode

Not secure !!

* Deterministic → If same key is used then identical plaintext blocks map to identical ciphertext. will not hide P.T from C.T. Dont hide mapping from P.T → C.T
* efficient (parallel encryp/decr)
* C_1 (error), P_1 will not be recovered (corrected but all other blocks will be recovered.

* parallelized, only affect current block (error) but deterministic security error.

Cipher-Block Chain (CBC) mode

* randomize it.
* Initialization-vector (IV), random 64 bit-string.
* DES o/p is pseudo-random, so use that as IV, but need clear new fresh IV/block.
* $C_0 \leftarrow IV$

  $C_i = E_K(P_i \oplus C_{i-1})$

  $C_0, C_1, C_2, \ldots C_n$

$$P_i = C_{i-1} \oplus D_K(C_i)$$

ADVANTAGE

(I) * randomizes, two same P.T blocks have diff. C.T (as IV is new)

(II) * losses in terms of efficiency. Cannot parallelize these as IV of i^{th} block depends of C.T of $(i-1)^{th}$ block.

DISADVANTAGE

(III) * decryption can be parallelized.

* Transmission errors transfer to next block

  $i^{th} \longrightarrow i^{th}$ & $(i+1)^{th}$

  $C_1 \rightarrow$ not sent prop, P_1 ✗ P_2 also

* CBC → secure against CPA
* not secure against CCA → C.T can be messaged to correlate with the P.T
* ECB → not secure to known P.T attack.

* how to achieve CCA security?
→ authentication
→ MAC (msg authentication code) on the C.T, if attacker manipulates C.T, she will also have to manipulate MAC which she/he cannot do due to properties of MAC system. (if changes C.T change would be detected using MAC)
→ other party decrypts only if MAC is valid.
→ CBC+ MAC → CCA security (highest-level security)

Lecture 2 : other ciphers

1) Advanced encryption standard (AES)
DES → 56, triple DES → 112
— Rijndael (AES)
— AES → key longer, key size can vary, both of substitution/transposition.
— AES + CBC + MAC (Best)
* maximize efficiency
2) IDEA (used in PGP) → email ✓
3) Blowfish (password hashing in OpenBSD)
4) RC4 (used in WEP), RC5 (wireless security protocol) ✓
5) SAFER (used in Bluetooth), pairing operation
* most devices can now support AES because of their computational efficiency.

Lecture 3: Public key Crypto Overview

* Private key model is symmetric bcoz it takes the same key K to encrypt / decrypt messages.
* encrypt → Public-key, Decrypt → Private-key (assymmetric)

Private-key

* Good: Quite efficient
* key sizes (112, 128, 192) relatively short, computation → easy & efficient
* Bad: Multiple people, multiple keys. key management is a problem. (Header problem)
* million keys (million clients - server)

Public-key

* less efficient → nature of large sizes of key (slower) much larger (1024, 2048 bits long)
* Good: key management & distribution (Post on Public domain/website) computation cheaper (E/D)

talk to (encrypt / decrypt) (public / private)

public private Public-key encryption

* e, d
* If d is not known, Decription hard.

Security Notions

* similar, difference? (hide P.T from C.T)

private key should not leak-out semantic (from C.T → anything about P.T)

Public (I) attacker has public key access
(II) Adversary can create encryption on its own.

Security against CPA

rest are same

Lecture 4: Math Background

G (group) (G is a set &, $\cdot : G \times G \to G$) :

$\cdot \to$ operation

1) Closure : $a, b \in G$, $a.b \in G$
2) Associativity : $a, b, c \in G$, $a.(b.c) = (a.b).c$
3) Identity : identity element e, $a.e = e.a = a$, for any $a \in G$
4) Inverse : there exist an element a^{-1} for every a in G, $a.a^{-1} = a^{-1}.a = e$

Groups: Examples

1) set of all integers with addition $(Z, +)$ (✓) [$e = 0$, $(a + (-a) = 0$ ($-a$ inverse, 0 e)]
2) $(Z, *)$ → not a group [$e = 1$, but $a \times \frac{1}{a} = 1$ $\frac{1}{a}$ not exist]
3) $(R, *)$ → ✓ [each element has inverse $(\frac{1}{a})$ which is real] (R set of all real no's) $3 \times \frac{1}{3} = 1$ ($\frac{1}{3}$ X (integer but not real))
4) Set of all integers modulo m with modulo addition $(Z_m$, "modular addition") (remainder when divide)

$(0, 1, 2, \dots, m-1)$ $(a^{-1} = -a)$

[does not belong Z_m, but same as $(m-a) + a = m$ $\frac{m}{m} = 0$]

$Z_m \Rightarrow [0, 1, 2, \dots, m-1]$

$Z_9 \Rightarrow [0, 1, 2, \dots, 8]$ $3 \times (\frac{1}{3}) = 1$

$Z_m = [$ $(\div / m = a)$

$Z_2 = [0, 1]$

$Z_3 = [0, 1, 2]$

$Z_9 = [0, 1, 2, 3, 4, 5, 6, 7, 8]$

$[0 + 1 = 1 \div 9 = 1]$

$0 + 0 = 0$
$1 + 0 = 1$
$2 + 0 = 2$

$(8 + 7 = 15 \div 9 = 6)$

$8 + (-8) = 0$ (-8 e)
$8 + 1 = 0$ $(9 - 8)$
$7 + 2 = 0$

* modular addition → $[\text{mod}_m(a+b)]$

* Identity = 0
* Inverse $a \rightarrow -a (\notin Z_m)$

$-a = (m-a)$, $\text{mod}_m[a+(m-a)] = 0$

~~Multiplicative~~ Multiplicative Inverse in Z_m:

$x(\text{mod}\, m) \equiv 1 * x(\text{mod}\, m)$

* $x^* x^{-1} \equiv 1 \text{ mod } m$

Imp * not all elements have inverses

$3X$ something $\neq 1 \text{ mod } 9$

* $4 \times 7 = 1 \text{ mod } 9 = 1$

$4 \times 7 = 28 = 1 \text{ mod } 9$

$4 \times 7 = 28$; $\frac{28}{9} = 1 = 1 \text{ mod } 9$

$4^{-1} = 7$

Imp * Inverse exists for elements which are relatively prime to $m (= 9)$

$\gcd(x, m) = 1$ Eg 4 in Z_9

↳ greatest common divisor

$\gcd(3, 9) = 3$

* efficient algos to compute inverses → to find multiplicative inverse even for very large nos (used in public key model)
  - Extended Euclidian Algorithm.

* Modular exponentiation

* $x^c \bmod n$

↳ multiply x, $(c-1)$ times & computing mod everytime

Inefficient if c is large

* Efficient algo → square & multiple algo

Euler's totient function $[\phi(n)]$:

* Given +ve integer n, $\phi(n)$ is no. of +ve numbers less than n that are relatively prime to n $[0, 1, .., (n-1)]$ $(\gcd = 1)$

* fact: If p is prime then

1) $-\{1, 2, 3, ..., p-1\}$ are relatively prime to p

$\phi(p) = (p-1)$

2) If p & q are prime, & $n = p \cdot q$

$\phi(n) = (p-1) \cdot (q-1)$

3) Each number that is not divisible by p or by q is relatively prime to $p.q$.

Euler's theorem and Fermat's (little) theorem:

1) If a is relatively prime to n, then

$$a^{\phi(n)} \equiv 1 \bmod n$$

2) If a is relatively prime to p, then

$$a^{p-1} \equiv 1 \bmod p$$

(prime no. p) $\phi(p) = p-1$

Eg $9^{100} \bmod 17$ (9 & 17 are relatively prime)
($\phi(17) = 16$, as 17 is prime.)

Imp $\phi = 17$, $(\phi - 1) = 16$. $100 = 6 \times 16 + 4$

$$9^{100} = 9^{(6 \cdot 16 + 4)} = (9^{16})^6 (9)^4$$

$$(9^{16})^6 (9)^4 \pmod{17} \equiv (1)^6 (9)^4 \pmod{17}$$
$$= (81)^2 \pmod{17} = 16$$

Q1* ~~Z_{10}~~ [3X9

gcd(3,10) = 1 (✓)
gcd(4,10) = 2 (X)
gcd(5,10) = 5 (X)
gcd(7,10) = 1 (✓)

Q2 $\phi(10) = [1, -, 3, -, -, -, 7, -, 9]$
<10 +ve

Q3 $3^6 \bmod 7$ 6 & 7 are relatively prime
$\phi(7) = 6$ gcd(6,7) = 1

$(3^6) = (1 \bmod 7) = 1 \bmod 7 = 1$

Q4 $7^4 \bmod 10$ 7 & 10 R.P
gcd(7,10) = 1

$\phi(10) = 4$

(7) $7^4 = (1 \bmod 10) \bmod 10$

Lecture 5: The RSA Cryptosystem (Encryption)

(multiplicative group) **RSA Math setting:**

- Z_n^* → a group of numbers b/w 0 to (n−1) that are relatively prime to n.
  - ↳ as arithmetic done in modulo n
- $n = (p \times q)$, p & q are prime
- size of $Z_n^* = \phi(n) = (p-1)(q-1)$
- Computation is done modulo n. (?)

(I) "Textbook" RSA: KeyGen

(1) KeyGen, (2) Encryp, (3) Decry

* $ed = 1 \bmod \phi(n)$ $\quad e = d^{-1} \bmod \phi(n)$

(e, n) → public-key

d → private, decryption-key

(e, n, Z_n^*, $\phi(n)$ → deleted)

(II) Encryption

- Bob wants to send a msg x (an element of Z_n^*) to Alice.
- encryption-key (e, n) in directory
- $y = E(x) = x^e \bmod n$
- Bob sends y to Alice.

(III) Decryption

- $y = E(x) = x^e \bmod n$
- $D(y) = y^d \bmod n = x$ [D(y) = x, claim]

* $e \cdot d = 1$, ∴ works-out well.

E and D are inverses:

$$D(y) = y^d \bmod n$$
$$\equiv (x^e \bmod n)^d \bmod n$$
$$\equiv (x^e)^d \bmod n \qquad \left[\frac{ed}{\phi(n)} = \text{or}(1)\right]$$
$$\equiv x^{ed} \bmod n$$
$$\equiv x^{t\cdot\phi(n)+1} \bmod n \quad * \left[ed = 1 \bmod \phi(n)\right]$$
$$\equiv (x^{\phi(n)})^t \cdot x \bmod n$$
$$\equiv 1^t \cdot x \bmod n \equiv x \bmod n \quad [\text{from euler's}]$$

[If x is relatively prime to n, $x^{\phi(n)} \equiv 1 \bmod n$]

Eg:

$p=7$, $q=11$, $n=77$ $[Z_{77}^*]$

[size of Z_{77}^* $\phi(n) = (7-1)(11-1) = 60$]

* [choose $e=13$; $d = 13^{-1} \bmod 60 = 37$] (known algo)
  - [as 13 is relatively prime to 60 ($\phi(n)$)]
  - [$13 \times \square = 1$ / 60]
* msg $= 2$
  - $E(2) = 2^{13} \bmod 77 = 30$
  - $D(30) = 30^{37} \bmod 77 = 2$ [square & multiplication algo]

Lecture 6: RSA - Security (Factoring)

* Eve can look-up (e, n) in the public-directory.
* Compute $d = e^{-1} \bmod \phi(n)$, then
  $D(y) = y^d \bmod n = x$!!
* Attacker does not know $\phi(n)$ as $\phi(n) = (p-1)(q-1)$ & attacker does not know (not know) (p, q).
* If (p, q) known, then can learn $\phi(n)$, d & $D(y)$.
  ↳ problem-statement [p, q from n]
* Attacker knows n = product of 2 primes, he needs to find factors of n.
  → Brute-force method takes $O(\sqrt{n})$ divisions. [order $(1, 2, \ldots, \sqrt{n})$, divide n fully] [from 1 to $\sqrt{n}$]
  → Prohibitive for large n, such as 160 bits → ($\sqrt{2^{160}} = 2^{80}$ operations)
  → Better factorization algos exist, but are still slow for large n. [faster than grade-school]
  → $n = p \times q$ (If one finds p/q, he can find the other). Also, p, q are the only 2 factors of n. [(p known; $q = n/p$)]
  → Lower bound for factorization is an open-problem
    2048 (round-off, both p & q 2048)
    • n = 1024 bits $\cong$ security provided by 80-bit long key in private-key crypto (atleast 2^{1024})
    (sophisticated algos ↳ take years to crack RSA)
* No other attack on RSA function known.
  - Except some side-channel attacks, based on timing, power-analysis etc. But, these exploit certain physical characteristics of, not a theoretical weakness in the cryptosystem.

* Even with very large n, the E&D can be done pretty efficiently!

* **During key generation:** $n \geq 1024$ (hard to break RSA as factoring hard)
  - select large primes
  - Primes are dense so choose randomly.
  - Probabilistic primality testing method. works in log time.
* **Compute multiplicative inverse** $[d = e^{-1} \bmod (\phi(n))]$
  - Efficient algo (extended euclidean algo)
* **During E&D**
  - modular multiplication (use squared & multiply) (exponentiation)

RSA in practise

- Textbook RSA is insecure
  - Since it is deterministic. [same msg twice, always get same CT]
- In practice, use a "randomized" version of RSA called RSA-OAEP [randomized P.T & then performs textbook RSA] → now, difficult to break.

(cannot hide P.T from C.T)

$$C_1 = m_1^e \bmod n$$

$$C_2 = m_2^e \bmod n$$

$$C_{eq} = [(m_1 m_2)^e \bmod n]$$

$$= [(m_1^e \cdot m_2^e) \bmod n]$$

$$\frac{m_1^e}{n} \rightarrow o_1$$

$$\frac{m_2^e}{n} \rightarrow o_2$$

$$(o_1 \times o_2)$$

$$\left[\frac{(m_1 m_2)^e}{n}\right] = (o_1 \times o_2)$$

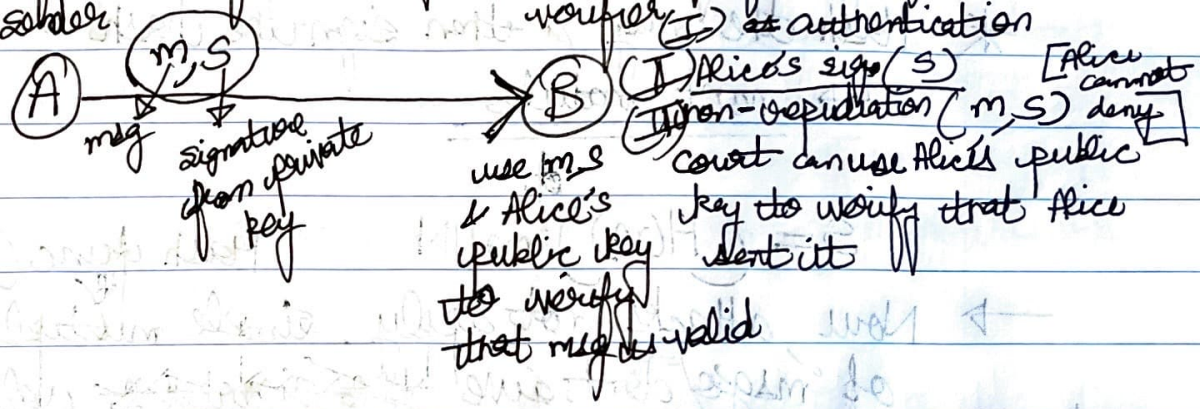
Lecture 7: Digital Signatures

~~[illegible]~~

* analogous to physical sig.
* Goals
  - Authentication (from the correct sender) — not guaranteed
  - Integrity (msg is not tampered in transit)
  - Non-repudiation (Someone not deny what they did) (Bob sent msg, Alice does not deny sending the msg)
* Not-confidential

Public Key Signatures

- Signer has public, private key pair.
- Signer signs using its private-key $[S = E(k, m)]$ (m, S) other party (public key)
- Verifier verifies using public-key of the signer.



Security Notion/Model for signature

- Existential forgery under (adaptively) chosen message attack (CMA)
  - Adversary (adaptively) choose messages m_i of its choice
  - obtains the signature S_i on each m_i.
  - outputs any msg $m (\neq m_i)$ and a signature S on m.

Goal of attacker → forgery

- Give power to attacker [query signature on any msg & then come up with a m, s pair to do forgery]

RSA signatures

- Key generation : same as encryption (e, d)
- $\text{Sign}(m) : s = m^d \bmod N$
- $\text{Verify}(m, s) : (s^e == m \bmod N)$

→ Goal of attacker & primitive diff. (but RSA)

→ textbook RSA insecure → ~~deterministic~~

→ randomized version of RSA (implemented in PKCS#1)

→ easily do forgery

eavesdrop

(m_1, s_1)

(m_2, s_2)

$s_1 \times s_2 = (m_1 \times m_2)^d \bmod N$ $(m_1 \times m_2) \Rightarrow (s_1 \times s_2)$

forgery

→ easy to go about it using hash-functions

* Hash the msg & then sign the hash

↳ next module

$$s = (H(m))^d \bmod N$$

Hash func (randomize)

→ Now attacks not apply. simple multiplication of msg's don't give ~~the signature~~ valid forgery

drawback → full-domain Hash ~~function~~ version / randomized RSA (randomized) in PKCS#1 standard.