1) Caesar Cipher → Add a constant difference to alphabet set.
ex: "abc" → "def" (Difference = 3)

K' Kerchoffs Principle:

Security depends on the secrecy of the key and not the obscurity of the encryption algorithm.

- Passwords are stored as shadow files.

a) → We want to follow Kerchoffs principle since reverse engineering of algo should not be possible.

b) → Secure memory is expensive, storing entire algo as a secret is hard.

c) → Multiple users and updation → If exposed, just need to change key.

- Nowadays, however, following this principle is mandatory.

d) Birth of ethical hacking
↳ Ethical hackers have access to algo so they could help solve issues.

e) Standards → To maintain systems used by a lot of users.

---

2) Shift Cipher: (26 ways)

- Have a random shift key which is known to both the sender & receiver (securely somehow)
- Break using frequencies p & encoded q. $\sum_{i=0}^{25} p_i q_{i+k} = \sum_{i=0}^{25} p_i^2$ (k = key)

Note: Principle of large key space (To stop brute force attacks)

---

3) Mono-alphabetic substitution cipher:

* • Each character has a substitute in lookup table
• Key = Permutation of the alphabet set. (26! ways)!
• Break cipher using: Frequencies of alphabets occurring in original & cipher text.

Made us realize ← brute force is not the only attack.

Note: Principle to ensure frequencies are not preserved.

---

4) Poly alphabetic substitution ciphers:

* • A key "k" is repeated & added to original text to get cipher.

ex: crypto
cat cat
---
er.....

→ Key = "cat" 3,0,19

Vegenere Ciphers: (Another name for poly alphabetic substitution ciphers)

* ⇒ Break cipher: If length of pass phrase known, we can break it.

* Brute force over the key lengths if it is unknown.
  - ⇒ Consider cipher text with jumps which would give you a shift cipher which you can break.
  - ex: Cipher text: $c_0 c_1 c_2 c_3 c_4 c_5 c_6 \ldots$
    Key len = 3
    Consider $c_0 c_3 c_6 \ldots$
    This is a shift cipher.
    ↓
    Break using frequencies
  - (Once all shifts are known, compare frequencies over entire text, maximum match occurs if length prediction of key is correct).

---

1. Security definition – What do you want (when to stop)
2. Computational hardness definition – Precise assumptions involved.
3. Proof of security – The tradeoff involved

---

2 Shannon's perfect secrecy:

- Secure communication scheme:

  <key gen, encrypt, decrypt, message space>

1) ⇒ n (length of key) → [Gen] → Key K

- $\mathcal{K}$ = {k | Gen outputs k} } Key space
  $\mathcal{K}$ needs to be finite.

2) m ∈ $\mathcal{M}$ (Message m ∈ $\mathcal{M}$) → [Enc] → c , c ∈ $\mathcal{C}$ (Cipher text space)
   k ∈ $\mathcal{K}$

3) c ∈ $\mathcal{C}$ → [Dec] → m ∈ $\mathcal{M}$
   k ∈ $\mathcal{K}$

---

* Correctness principle:

- $m_{input} = m_{decrypt}$

- If decompression algo uses same key as the encryption algo then $m_{ip} = m_{op}$. (ex: Shift cipher follows this).

⟹ Cipher scheme (Gen, Enc, Dec, $\mathbb{M}$) is

* secure if $\forall$ prob distributions over Message space $\mathbb{M}$, $\forall m \in \mathbb{M}$ & $\forall c \in \mathbb{C}$, no matter which message of cipher, what we know about m before cipher is what we know after cipher then the scheme is perfect.

ie. $P[M=m] = P[M=m \mid C=c]$

for all c such that $P(c=C)>0$

$\left(\because P(A|B) = \frac{P(A \wedge B)}{P(B)} \rightarrow \text{Non zero}\right)$

Shannon's secure bandwidth theorem:

⟹ Irrespective of how fast insecure channel is, max bandwidth of sending secure messages is the secure channel bandwidth so send it along secure channel only instead of using both.

Corollary: If secure channel b/w is 0, then you have no cipher that can send secure messages.

Proof: A cipher is perfectly secret if,

** → Show $P[c=C|m=m_0] = P[c=C$

- We have, $P[c=C|m=m_0]$ (LHS)

$P(c=C|m=m_0) \cdot \frac{P(m=m_0)}{P(c=C)}$ → Multiply on both sides

$= \frac{P(c=C) \times P(m=m_0)}{P(c=C)}$

$= P(m=m_0)$

$\left(\because \frac{P(B)\cdot P(A|B)}{P(A)} = P(B|A)\right)$

Now, RHS $\Rightarrow P[c=C]$,

$\Rightarrow P[c=C] \cdot \frac{P(m=m_0)}{P(c=C)}$

$= P[m=m_0]$

$\therefore$ LHS = RHS.

$\Rightarrow$ A cipher is perfectly safe iff $\forall m_0, m_1 \in \mathbb{M}$

$P[c=C|m=m_0] = P[c=C|m=m_1]$

- If the cipher is perfectly safe

$\forall m, C \quad P(c=C|m=m) = P(c=C)$

$P(c=C|m=m_0)$

$= P(c=C)$

$= P(c=C|m=m_1)$

$\Rightarrow P(c=C) = \sum_{m\in\mathbb{M}} P(c=C|m=m)\cdot P(m=m)$

Now, $P(c=C|m=m_0) = P(c=C|m=m_1) = p$

$\forall m_0, m_1$

$\therefore P(c=C) = \sum_{m\in\mathbb{M}} P(m=m)\cdot p$

$= p \sum_{m\in\mathbb{M}} = P(c=C|m=m)$

---

* Cipher (One Time Pad) — Vernam cipher *

- Gen: Key drawn randomly from 2^n keys of bitstrings of length n.

$$\therefore P(K=k) = \frac{1}{2^n} \quad (k = \text{key})$$

$$\Rightarrow \text{Bitstrings of length } n = \{0,1\}^n$$

- Enc: Bitwise XOR of key & m. (Assuming message $\in \{0,1\}^n$)

$$C = Enc_k(m) = k \oplus m$$

- Dec: $m = Dec_k(c) = k \oplus c$

Theorem: Vernam cipher is perfectly secret.

Proof:

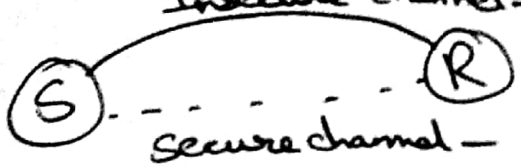
$$P(c=c \mid m=m_0) = P(c = k \oplus m_0 \mid m=m_0)$$
$$= P(c = k \oplus m_0)$$
$$= P(k = c \oplus m_0)$$
$$= \frac{1}{2^n}$$

$\therefore$ For any 2 messages $P(c=c \mid m=m_0) = P(c=c \mid m=m_1) = 1/2^n$.

(Since keys are chosen uniformly at random)

- ~~However, reusing pad, proof does not work~~

• However, length of key = length of message



⇒ Key needs to be sent securely which limits total bandwidth to bandwidth of secure channel, so we could just send data through secure channel instead.

• Applications of one time pad
  ↳ Useful when secure channel is not always available, send key & store once for further communication.

⇒ Limitations of perfect secrecy

*Theorem: For any perfectly secret encoding scheme $|M| \leq |K|$ must hold.

ie. Size of key space should be ≥ message space.

Proof: Suppose $|M| > |K|$,

$Enc_K(m) = C$, we would certainly have two messages m_1 & m_2 such that $Enc_k(m_1) = Enc_k(m_2) = c$ (pigeon hole principle)

∴ But this does not ensure correctness and hence its not possible

∴ There exist $m \in M$ such that $Pr[C=c \mid M=m] = 0 \ \forall c \in C$

∵ We need $Pr[C=c \mid M=m]$ to be equal $\forall m = Pr[C=c]$

---

(3) • Fast secure channels are necessary & sufficient for fast secure communication.

⇒ Relaxations on Perfect secrecy

* → P.S Forever (Infinite time) → Some large period of time

→ Adv. cannot side his/her luck (Zero error prob) → A very small permissible error

∴ Both are necessary relaxations for practical feasibility.

• Forever P.S must be relaxed for password schemes etc for practical purposes. This makes finite key spaces and hence should relax for a small permissible error.

+

⇒ One way functions: $x \to f(x)$ is easy but $f(x) \to x$ is hard.

⇓

• These two together are sufficient for perfect secrecy

*

---

* P vs NP:

Theorem: One way functions exist ⇒ P ≠ NP

Proof: • NP = ∃ certificate c (solution) s.t in polynomial time

(eg: Clique etc)

w, c, Does $w \in L$?

(Verification possible)

(Non deterministic polynomial time)

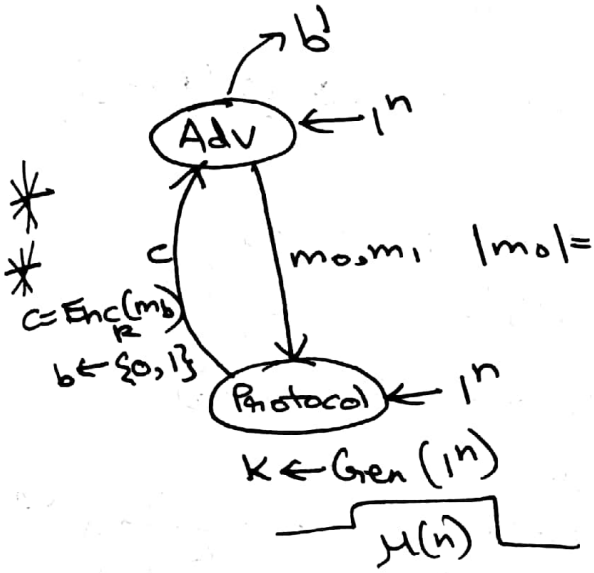
→ This can only be NP at best since a solution x can be verified in P. So verification possible.

∴ If one way functions exist, forward is P & backward is NP but reverse is hard & forward is easy. Therefore $P \neq NP$.
(for one way functions)

Note: Two fields

→ Slow secure + Fast insecure ≡ Fast secure
(Symmetric key cryptography)

→ No secure + Slow insecure ≡ Slow secure
(Public key cryptography)



Probabilistic polynomial time turing machine A

- ∀ PPTM A,

$$P[(b'=b)] \leq \frac{1}{2} + \mu(n)$$

↓ Negligible

(Cipher text only attack)

- A function is said to be sufficiently negligible if ∀ polynomials $p(\cdot)$ ∃ n_0 such that ∀ $n \geq n_0$, $\mu(n) < \frac{1}{p(n)}$

- This "game" is equivalent to shannon's theorem if μ(n) is 0 etc. But this is a relaxed version

---

esc: Consider $\mu(n) = 1/2^n$, show that this is a negligible function.

$\because \frac{1}{2^n} < \frac{1}{p(n)}$ ∀ p(n) for some $n \geq n_0$ //
(Cutoff point)

• We need $\mu(n)$ to be negligible since any polynomial amount of trials m would only provide an advantage $P(b=b') \leq \frac{1}{2} + \mu(n)$ (still negligible)

∴ We need $\mu(n)$ to be negligible.

⟶ This would allow us to just increase key length as adversary gets faster (remaining polynomial)

• The $\mu(n)$ also ensures we are not too lucky.

∴ This definition is secure against cipher text only attacks.

---

(4) Pseudo Random Generator: (PRG)

* • It is a deterministic program,

** $G: \{0,1\}^n \longrightarrow \{0,1\}^{l(n)}$

↳ $l(n) > n$ (Expansion)

↳ Pseudo randomness

• $\forall$ PPTM D, $s \xleftarrow{R} \{0,1\}^{l(n)}$ (Random truly)

$s \xleftarrow{R} \{0,1\}^n$

• $\left| P[D(s)=1] - P[D(G(s))=1] \right| \leq \text{negl}(n)$ must hold for pseudorandom

∴ We can use pseudo random generators since they cannot be distinguished in polynomial time.

↳ With a polynomial set of samples for D, it can't distinguish s & G(s) if $\{0,1\}^{l(n)}$ & $\{0,1\}^n$ are non polynomial. Hence even though G(s) maps to only a small space of $\{0,1\}^{l(n)}$ we can still its distinguish

$\Rightarrow M = \{0,1\}^{l(n)}$, Gen: $k \xleftarrow{R} \{0,1\}^n$

$K = \{0,1\}^n$

- $Enc_k(m) \Rightarrow c = m \oplus G(k)$
- $Dec_k(c) \Rightarrow m = c \oplus G(k)$

- Consider an adversary A that sends 2 messages m_0, m_1 and gets back C corresponding to one of m_0 or m_1. Assume A can figure out which message C corresponds to.

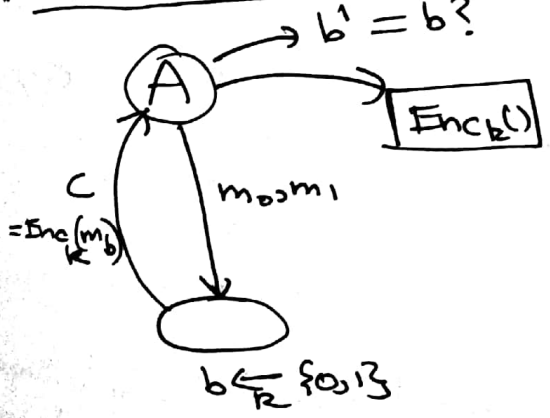
$\Rightarrow$ Consider a discriminator D, all it does is send encrypted $w \oplus m_b$ to A and ask it to guess. If its correct then w is pseudorandom else it is truly random. (assumption)

$\therefore P[D(G(s))=1] - P[D(G(s))=1] = 0$ since A is always correct

$\Rightarrow$ $\therefore$ If PRG exists then our method of encryption is secure as per our definition. (& vice versa?)

---

* Chosen Plaintext Attack: (CPA)



- CPA secure scheme is safe against this as well.

- No deterministic encryption algorithm is CPA secure. Since A could just get c_0, c_1 from oracle and figure out b'. Decryption algorithm should be deterministic to be correct.

- $c = \langle r, m \oplus Enc_k(r) \rangle$ (r = Random)

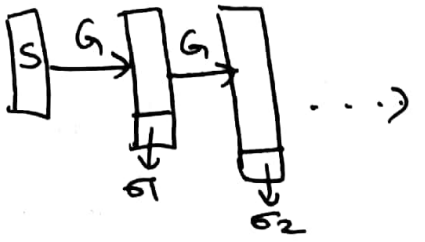
Discrete Logarithm Problem (DLP)

* • $\{1, 2, \ldots, p-1\}$ where p is a prime
  - Assume multiplication $\% p$ for group as prime
  - $\mathbb{Z}_p^*$
  - $\{g^1, g^2, \ldots, g^{p-1}\} \in$ group.
  - $\Rightarrow g$ = Generator. There always exist such generators.

- $y = g^x \bmod p$, Find $x = \log_g y$
  - Given p, g, y
  - (This is a hard problem as $p \uparrow$)

Note:

* • Given that $G : \{0,1\}^n \rightarrow \{0,1\}^{n+1}$ (Pseudo random generator.) then it is capable of generating $\{0,1\}^{l(n)}$.
  - ↳ Just keep passing outputs back to G till we get $\{0,1\}^{l(n)}$ starting from $\{0,1\}^n$.
  - ↳ This is also pseudo random since if we say final output is pseudo random, we cannot distinguish it



$\sigma_1 \sigma_2 \ldots \sigma_{l(n)}$
$r_1 \sigma_2 \ldots -$
$r_1 r_2 \ldots -$
$\vdots$
$r_1 r_2 \ldots r_{l(n)}$

Need to be indistinguishable if total is pseudo random

We know any 2 pairs of adjacent are indistinguishable since G is pseudorandom. → All indistinguishable pairs cannot give distinguishable result

∴ Our method is pseudorandom

PSD

A) ⇒ If discrete log is hard, then we

* have a pseudo random generator as such.

Starting seed is an element of $Z_p^* = s_0$

$$\sigma_i = MSB(\underbrace{g^{s_{i-1}} \bmod p}_{s_i})$$

This is indistinguishable from random.

Discrete log: Given $p, g, y = g^x \bmod p \rightarrow$ what is x? (Assumed Hard)

- However this method is really slow since it only generates 1 bit at a time.

---X---

(5) Note: Trap door one way functions

↳ $x \rightarrow f(x)$ easy

$f(x) \rightarrow x$ is hard unless we have more info, then it is easy

---

*Solving DLP (Not possible) → (But in theory)

Given $< g^x \bmod p, p, g >$, what is $MSB(x)$?

*

⇒ Getting $LSB(x)$ is easy.

---

* Fermat's Little theorem:

↳ $\forall a \in \{1, 2, \ldots, p-1\}$

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

Proof: Consider $a, 2a, 3a, \ldots, (p-1)a$

↳ Zero never appears since $0 \bmod p$ is not part of set

↳ $ia - ja$ for any i, j cannot be same since $ia - ja = (i-j) \cdot a$

∴ If same $(i-j) \cdot a = 0$ & $0 \bmod p$ is not a part of group. (∵ All are distinct)

→ On multiplying all,

$$1(p-1)! \equiv a^{p-1} \cdot (p-1)!$$

$$\therefore a^{p-1} \bmod p = 1 \quad //$$

Now, suppose we consider → Since p is prime, p-1 is divisible by 2.

$$* \quad \left[g^x \bmod p\right]^{\frac{p-1}{2}} \bmod p = \begin{cases} 1 \text{ if } x \text{ is even} \\ g^{\frac{p-1}{2}} \text{ if } x \text{ is odd} \end{cases}$$

↑ Can be calculated easily since we have $g^x \bmod p$

↳ Therefore we can find out if last bit (LSB) is 0 or 1 by doing this.

- If we want to know more bits, we can extend the approach further as such.

↓

- We can extend this to get more bits as,

* if $4 \mid [redacted]_{p-1}$ then

Order 4 algorithm

$$\left[g^x \bmod p\right]^{\frac{p-1}{4}} \bmod p = \begin{cases} 1 \text{ if } [redacted] \equiv 0 \pmod 4 \\ g^{\frac{p-1}{4}} \text{ if } [redacted] \equiv 1 \,(\%4) \\ g^{\frac{p-1}{2}} \text{ if } [redacted] \equiv 2 \,(\%4) \\ g^{\frac{3(p-1)}{4}} \text{ if } [redacted] \equiv 3 \,(\%4) \end{cases}$$

- Similarly if p-1 is divisible by 3 or 4 or 5 etc, we can find a similar method to get answers if $x \equiv i$ (% 3 or 4 or 5 etc)

⇒ If p-1 can be written as a product of powers of primes easily, then we can apply this algo to each individually and then use chinese remainder theorem to get the actual values.

* MSB(x) is hard though. ($\sqrt{y} \Rightarrow g^{x/2} \bmod p$) → Root divides x/2 → Used in DLP solution

Proof: Let $y = g^x \bmod p$ → Try to find z

$\rightarrow z^2 \equiv y \pmod p$ (p = Prime)

↓

↳ Now $p \equiv 0 \pmod 4$ } Not possible
$p \equiv 2 \pmod 4$ } Not possible

If, $p \equiv 3 \pmod 4$

↳ SQRT of y = ?

• Is $y^{\frac{p+1}{4}} \stackrel{?}{=} z$

↓

Consider $\left(y^{\frac{p+1}{4}}\right)^2 = y^{\frac{p+1}{2}}$ → Given y is a quadratic residue

$= \left(g^{2k}\right)^{\frac{p+1}{2}} = g^{k(p+1)}$

$= g^{k(p-1)+2k} = \left(g^{k(p-1)}\right)^1 \cdot g^{2k}$ (From format)

$= y$

• $y = g^{2k} \% p$ (Since it is a quadratic residue)

$\therefore \pm y^{\frac{p+1}{4}}$ is a root

$\therefore z = \pm y^{\frac{p+1}{4}} \bmod p$ }

* DLP Solution:

* Now, $y = g^x \bmod p$, To find x, if we have algo for LSB,

LSB(x) → 0 → $\sqrt{y}$ → 0 → $\sqrt{\sqrt{y}}$ ...
LSB(x) → 1 → $\sqrt{y/g}$ → 0 → ...
$\sqrt{y/g}$ → 1 → ...

$\because y/g \Rightarrow g^{x-1}$

• However, $\sqrt{y}$ gives 2 numbers, ∴ branches exponentially

Note: Quadratic residues:

*

$\{x^2 \% p \quad \forall x \in \{0, 1, \ldots, \}\}$

• These numbers that have square roots are called quadratic residues of p

- Our square root algo gives us 2 solutions, we don't know which of them is the answer.

→ ∴ The recursion expands out way too quickly

⟹ Now, if we have an algo for MSB(x)

↳ MSB(x_1), MSB(x_2) (If x_1, x_2 are our solution to square root)

↳ Use that root where MSB is 0, ignore the root where MSB is 1.

- This would allow us to get our complete solution to DLP.

Note: $-1 = g^{\frac{p-1}{2}} \bmod p$,

$\therefore \pm g^{x/2} \bmod p \Rightarrow g^{x/2} \bmod p$ or $g^{\frac{x}{2}+\frac{p-1}{2}} \bmod p$

∴ MSB is a hard core predicate of DLP.

---

* One way function:

* $f: \{0,1\}^n \to \{0,1\}^n$

- $\exists$ PPTM M such that,

$\forall x, P[M(x) = f(x)] \geq 1 - \text{negl}(n)$

- $\forall$ PPTM A,

$\forall x, P[A(f(x)) \in f^{-1}(f(x))] < \text{negl}(n)$

---

* Hard core predicate:

(of a one way function f)

- Predicate = Any function that outputs a boolean.

$h: \{0,1\}^n \to \{0,1\}$ is a h.c.p of f if,

$x \to h(x)$ (Easily)

$\forall$ PPTM A,

$$P\left[A(f(x)) = h(x)\right] < negl(n) + \frac{1}{2}$$

$x \xleftarrow{R} \{0,1\}^n$

Eg: If Discrete log is a one way function, MSB is a hard core predicate.

---

* Pseudo Random Generator:

$$G_1(s) = h(s_1) \| h(s_2) \| h(s_3) \| \ldots\ldots$$

$s_i = f(s_{i-1})$, $s_0 = s$ } seed.

- Here f = One way function & h is a hard core predicate of f.
- Using any such (f, h) we can form PRGs as above by concatenation.

Note: There are generic hard core predicates that work with any one-way fns.

∴ PRGs just require one way functions to exist.

Note: If $p - 1 = 2^s \cdot s$ (s = Odd) (In DLP)

- Then s^{th} bit is a hard core predicate
  → (From LSB side) → No need to go all the way to MSB.

(6) Pseudo random functions: (PRF)

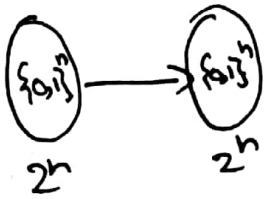
**

$$F : \{0,1\}^n \times \{0,1\}^n \longrightarrow \{0,1\}^n$$

(K: $\{0,1\}^n$, x: $\{0,1\}^n$, $F_k(x)$: $\{0,1\}^n$)

- First time you query with an x, you should get a random output but further queries with x should give same output as earlier.

↳ Truly random functions:



} Key basically selects a function from all possible functions.

→ This has $(2^n)^{2^n}$ possible functions. So using a key to index needs a key of length $n.2^n$ bits → Hard and hence we can't use truly random functions

---

- Hence our key of n bits only allows us to index 2^n functions from $(2^n)^{2^n}$ which is minute.

∴ Ratio $2^n/(2^n)^{2^n}$ is easy to differentiate and hence people can differentiate blw our pseudo random function and truly random functions.

Note: Pseudo random one-one functions are called Pseudo random permutations or block ciphers (DES, AES, IDEA)

* Oracle machines:

- A PPTM which can make input/output calls to oracle machines can allow tasks to be done.

exc: P^{NP} } Oracle machine is NP, our machine is P.

Def:

- F_k is a PRF if

* i) Given x, $F_k(x)$ is easy to compute
  ii) $\forall$ PPTM A, $\left| P[A^{F_k(\cdot)} = 1] - P[A^{R_k(\cdot)} = 1] \right| \leq negl(n)$

  ↓ Oracle pseudo random func.

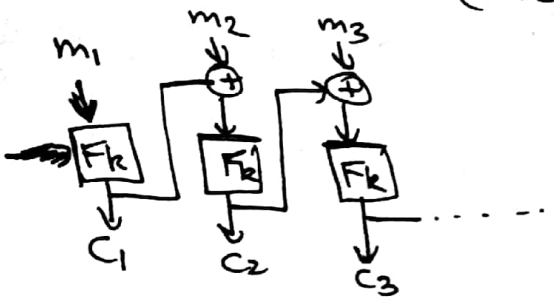
  → Should not be able to differentiate b/w pseudo random and truly random.

---

* Modes of operation of a block cipher:

- Standard way is split message into blocks of q bits and each block is encoded as $\langle s, F_k(s) \oplus m \rangle$ } This however leads to length doubling of the message.

i). CBC (Cipher block chaining) (Secure)



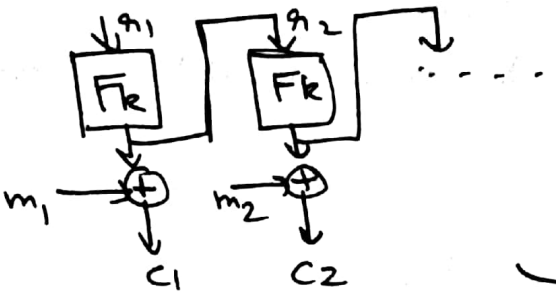
- Decryption requires entire output cipher, so loss in network packet transmission etc, methods fail.

(For decryption ↑)

- Requires F_k to be invertible. In practice, invertible PRFs are known as block ciphers.

* ii) Output feedback mode: (Secure)

- $r_i = F_k(r_{i-1})$



- This ensures decryption just requires r_1 & rest can be generated on the fly as required.

↳ (Pseudo random function to pseudo random generator)

iii) Randomized counter mode: (Secure)

- $r_i = r_0 + i$ (Lightweight & easy, yet secure)

---

* PRF from PRG:

- Consider a length doubling PRG $\{G\}$

$$G : \{0,1\}^n \rightarrow \{0,1\}^{2n}$$

- Let $G_0(x)$ = 1st n bits of $G(x)$
  $G_1(x)$ = Last n bits of $G(x)$

$$(G(x) = G_0(x) \| G_1(x))$$

$\Rightarrow$ Let $r = r_1 r_2 \ldots r_n$ } n bit string

- Define $F_k(r)$ as follows, (efficient to compute)

$$G_{r_n}\left[\cdots G_{r_3}\left[G_{r_2}\left[G_{r_1}[k]\right]\right]\cdots\right] = F_k(r)$$

$\Rightarrow$ This is pseudo random since, by induction

Induction:

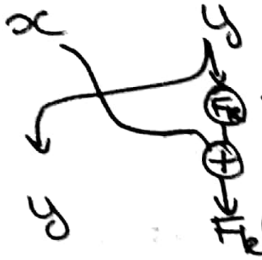
- At first level $G_0(k)$ or $G_1(k)$ is pseudo random function since G is a pseudo random generator

- Further levels have to be pseudo random functions since if level i is pseudo random function, so is level $i+1$.

---

* Feistel Structure: (Over a PRF)
(Finally gives a possible solution)

* Although we need invertible $F_k(x)$'s, we can get them through using methods like,



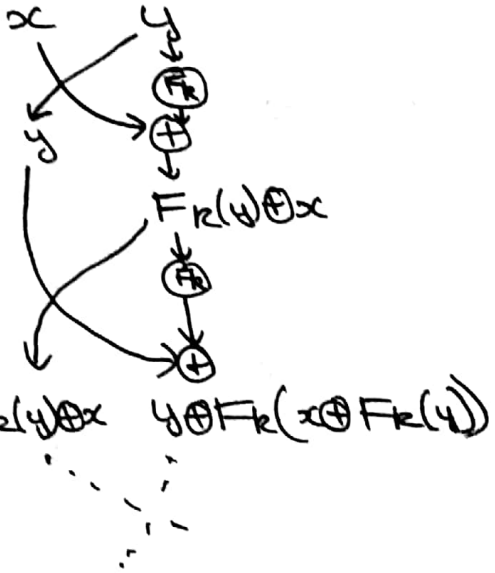
→ Need not be invertible.

} Given outputs, we can get back inputs.

(This is only semi-pseudo random, so we repeat)

* ⇒ Feistel Structure:

(Repeat for multiple rounds)



- This is believed to be secure after 4 rounds.

---

* DES: (Data Encryption Standard)

↳ Same as Feistel structure but goes for 16 rounds with a weak $F_k()$ which is not truly pseudo random.

(This is pretty secure in practice)

↳ This requires a key for each round and propose a method to derive K_i from k_1 for all rounds i.
(An attempt at a strong pseudorandom permutation)

Note: Strong pseudorandom permutations,
∀ PPTM, they can't distinguish given oracle access to $F_k()$ & $F_k^{-1}()$.
(Stronger block ciphers → AES etc)

(7) Secure communication

↳ PPTM & negl error allowed
↳ Security against cipher text only attack
↳ Scheme based on PRGs

- Block ciphers (CBC, OTB, Counter Modes)

* Data Integrity problem:

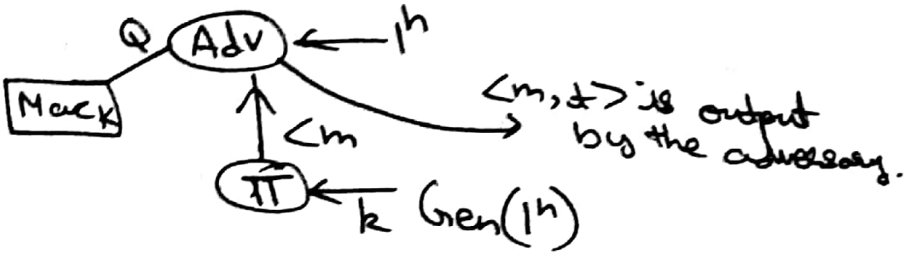
- Receiver should be able to detect if message is intact or modified.

Note:
(• CCA (Chosen ciphertext attack)
↳ Adversary has access to decryption servers.)

⇒ We use message authentication
* codes (MAC_k)

(Gen, Mac, Vrfy)
$t \leftarrow Mac_k(m)$
Yes/No $\leftarrow Vrfy_k(m, t)$

- Mac is probabilistic, but Verify is deterministic.
- A MAC scheme is secure if: for any message except m should be given with their tags if asked and the adversary should not be able to forge tag t of m.



Then adversary is successful { a) If $m \notin Q$ and $VES = Vrfy(m,t) \Rightarrow$ Break
↳ MAC = 1 condition.

Replay attack: May or maynot be a valid security breach depending on the application.
(Can be ~~broke~~ guarded by using sequence numbers at the application layer) code words.

- MAC scheme is secure if $P[\text{Adv success}] \leq negl(n)$

* Solutions to Data Integrity:

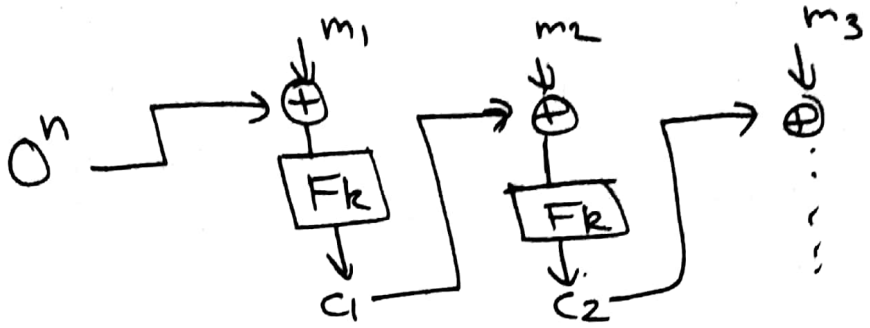
→ CBC-MAC (Mid 80s - 90s)

→ H-MAC (Used now)

- $t \leftarrow F_k(m)$ } F_k gives the tag for a message.
  → If $F_k(m)$ was random, then for any new message m regardless of previous seen messages, the output would be random $\frac{1}{2^n}$ probability (for an n bit key)
  - $t_i \leftarrow F_k(t_{i-1} \oplus m_i)$ ~~the tag fails~~.

*CBC-MAC: (Fixed Length MAC)

*



- In crypto 0^n had to be random or else it would fail, here it cannot be random otherwise it would not work.

$m_1 \rightarrow c_1$

$m_1\, m_2 \rightarrow c_2$

~~(illegible)~~

- $tag_0 = 0^n$, $tag_1 = F_k(m_1 \oplus tag_0)$

  $tag_n = F_k(m_n \oplus tag_{n-1})$

- If $m_2 = m_1 \oplus c_1$, then

  $c_1 \oplus m_2 = m_1$

  & $F_k(c_1 \oplus m_2) = F_k(m_1) = c_1$

  c_2

  $\therefore c_2 = c_1$ we know even though we never queried it.

This is an issue since we get same tag for different messages.

- CBC-MAC has issues for variable length inputs and is not completely secure.

⇒ To handle variable length messages,

① Prepend length → to $|m|$ → Include length of message first → $l \, m_1 \ldots m_q$ (q blocks)

② Use key to encrypt 'l',

$K: k_1 \leftarrow F_k(l)$

↓ Fixed length CBC } Use this key to encrypt.

③ $k_1, k_2, t = CBCMAC_{k_1}(m)$

(tag) $\leftarrow F_{k_2}(t_0)$

~~(Encrypt last tag over & over till desired length)~~

(Encrypt last tag directly to denote end)

(with key k_2)

---

⑧ Mid | Review:

- Historical ciphers
  - → Shift cipher — Brute force attack
  - → Mono alphabetic substitution cipher — Frequency attack
  - → Vignere cipher — Brute force & frequency.
- Kerchoff's principle
- Shannon's perfect secrecy
- One time pad is perfectly secret.
- Limitations of any perfectly secret scheme

  $|M| \leq |K|$
- Two relaxations.
  - → PPTM adversary
  - → Negl probability of error

    *(Negligible functions) above

- PRG (exists iff one-way functions exist)

- Secure encryption schemes
- CPA (Chosen plaintext attack)
  - ↳ CPA secure encryption scheme (oracle access to encryption oracle)
- Probabilistic encryption
- PRF
- CPA secure enc $\langle r, F_k(r) \oplus m \rangle$
- Modes of operation
  - ↳ Cipher block chaining (CBC)
  - ↳ Output feedback mode (OFB)
  - ↳ Randomized counter mode
- PRF $\leftrightarrow$ PRG
  - ↳ Feistel structure
- DES (3-DES is used thesedays)
  - ↳ $DES_{k_1}(DES^{-1}_{k_2}(DES_{k_1}(m)))$
- CCA (Chosen ciphertext attack)
  - ↳ Adversary knows certain functions on ciphertext would correspond to certain operations on the plain text.
  - ↳ Adversary attempts to recover the key used for decryption from pairs of ciphertexts & their decryptions

  (Semantic security)
- MAC (Message authentication codes)
  - ↳ CBC_MAC

(Mid2) - CPA secure + MAC $\Rightarrow$ CCA secure scheme.

* ↳ Encrypt m using CPA secure to get c
* ↳ Apply MAC on c (To render oracle access to decryption useless)

$\langle \langle r, F_k(r) \oplus m \rangle, tag \rangle$ } This is CCA secure.

(Compress, encrypt & authenticate)

⇒ What is information?

In reality its a combination of these

- Randomness (Shanon) — Randomness determines how data can be simulated.
- Space (Kolagarov) — Size determines amount of info
- Time (Levin) — Time determines amount of info.

↓

what is the best way to define information?

simplest way to simulate

---

(9) Public Key Cryptography:

↳ Diffie Hellman Key Exchange

↳ RSA algo & PKCS v1.5

↳ ElGamal scheme. (Provably CPA secure)

- Use two keys, encryption key is pubic, decryption key is private.

* DH SKE: (Diffe Hellman Secure Key Exchange)

* ① A chooses $a \xleftarrow{R} \{1, \ldots, \}$
$g^a \bmod p$ } Sent to B by A.

② B chooses $b \xleftarrow{R} \{1, \ldots, \}$
$g^b \bmod p$ } Sent to A by B

③ Key $= k = g^{ab} \bmod p = (g^a \bmod p)^b \bmod p$ → Both of them now have a secure key

- COM (Computational)
  ↳ given $g^a \bmod p$ & $g^b \bmod p$, getting $g^{ab} \bmod p$ is hard.

---

*RSA: (An example trapdoor one way function)

Gen $p, q \rightarrow$ two large primes

$N = p \cdot q$ } Hardness of system depends on factorizing N.

If, $e \mid \gcd(e, \phi(p-1)(q-1)) = 1$

$\Rightarrow \exists\, d \mid ed \equiv 1 \pmod{\phi(p-1)(q-1)}$

$\Rightarrow ed \equiv K(p-1)(q-1) + 1$

Pubic Key $\rightarrow pk \langle N, e \rangle$

Secret Key $\rightarrow sk \langle p, q, d \rangle$

$c = Enc_e(m) = m^e \bmod N$

$Dec_{sk}(c) = c^d \bmod N$

From Fermat's little theorem extension

No. of coprimes of N $= (p-1)(q-1) = \phi(N)$

Now,

$c^d \bmod N = [m^e \bmod N]^d \bmod N = m^{ed} \bmod N$

$= m^{ed \bmod (p-1)(q-1)} \bmod N$

$\Rightarrow$ Now, $a^{p-1} \equiv 1 \pmod p$ → Fermats little theorem

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

$(\phi$ = Eulers totient function)

$a^x = a^y \bmod N$ where $x = y + \phi(N) \cdot k$

$\phi(n) = pq - p - q + 1 = (p-1)(q-1)$

---

exc:

$\Rightarrow$ If e – small, $(e=3)$

$c_1 = m^3 \bmod N_1$

$c_2 = m^3 \bmod N_2$

$c_3 = m^3 \bmod N_3$

} $0 \leq m^3 < N_1 \cdot N_2 \cdot N_3$ (there exists)

* • Using chinese remainder theorem (CRT)

* ↳ If $n_1, n_2, \ldots n_k$ are mutually coprime

$x = a_1 \pmod{n_1}$

$x = a_2 \pmod{n_2}$

$x = a_k \pmod{n_k}$

$x < n_1 \cdot n_2 \cdots n_k$

• We can find x given $a_1, \ldots$ & $n_1, \ldots, n_k$. (& vice versa)

Proof: If $a_1 = 0, a_2 = 0, \ldots, a_k = 0$

* ↳ Here $x = 0$ is the only one that satisfies

- If $a_i = [1, 0, \ldots, 0]$

  ↳ $x = \left(\prod_{i=2}^{k} n_i\right)\left(\left[\prod_{i=2}^{k} n_i\right]^{-1} \bmod n_1\right)$

- If $a_i = [0, 1, 0 \ldots, 0]$

  ↳ $x = \left(\prod_{i \neq 2}^{k} n_i\right)\left(\left[\prod_{i \neq 2}^{k} n_i\right]^{-1} \bmod n_2\right)$

and so on...

⇒ Now $[a_1, a_2, \ldots, a_k]$

$= \left(a_1[1, 0, \ldots, 0] + a_2[0, 1, \ldots, 0] + \ldots + a_k[0, 0, \ldots, 1]\right)$

$\bmod \left(\prod_{i=1}^{k} n_i\right)$

Final solution,

$$\Rightarrow x = \left\{\sum_{i=1}^{k} a_i \cdot \left(\frac{\prod_{j=1}^{k} n_j}{n_i}\right)\left[\left[\frac{\prod_{j=1}^{k} n_j}{n_i}\right]^{-1} \bmod n_i\right]\right\} \bmod (N)$$

$$N = \prod_{i=1}^{k} n_i$$

---

* PKCS v1.5:

$$c = \left[00000000 \| 00000010 \| r \| 00000000 \| m\right]^{e} \bmod N$$

(r: random atleast 8 byte string which are not all zeroes; m: Message)

∴ Additional length is > 11 bytes.

- This is probabilistic encryption since we have r.

(r can't be all zeros since for decryption we need to know where r ends)

- For RSA, LSB of x is a hard core predicate. MSB of x is easy. Hence we pad with 16 bits so that it can't be cracked. (Efficient algos can crack at most 16 bits MSB side)
  - ↳ Some algos can crack a few more bits but since we have randomness, the attacker gains nothing.

Note: RSA - OAEP } Proved to be theoretically secure

---

* El Gamal: (Public Key Exchange)

(Partially homomorphic encryption scheme)

- $\langle G, g, p, h = g^x \rangle$ } Public
  - p → Order of group
- x } Secret Key

⇒ Enc: choose $y \xleftarrow{R} G$ } → Random element from group

$$Enc(m) = \langle g^y, h^y \cdot m \rangle$$

⇒ Dec: $\langle g^y, h^y \cdot m \rangle$,

$$\frac{h^y \cdot m}{(g^y)^x} = m //$$

(From g^x & g^y, we can't get g^{xy} from diffie hellman)

- This is probabalistic (inbuilt into scheme)

∴ This is CPA secure.

Note: Homomorphic encryption scheme:

$$\Pi(Enc(x)) = Enc(\Pi(x))$$

- Allows for cloud computing keeping data secret.

---

(10) Hashing (Hashes were indexed functions H^*) → Some mapping function. (Fk → key k is search, indexes are not search)

↳ Collision Resistant Hashing schemes

- Collision: $H(x) = H(y)$ s.t. $x \neq y$.

⇒ Hashing is compressing. With compression it is impossible to have no collisions.

- Since hashes are of a fixed length, in theory ∞ inputs map to same hash. But we relax it as PPTM adversary A should not be able to find a collision.

---

* Merkle Damgaard Transform:

Input: $h: \{0,1\}^{2n} \to \{0,1\}^n$ } If this is collision resistant

↓

Output: $H: \{0,1\}^* \to \{0,1\}^n$ } This is also collision resistant

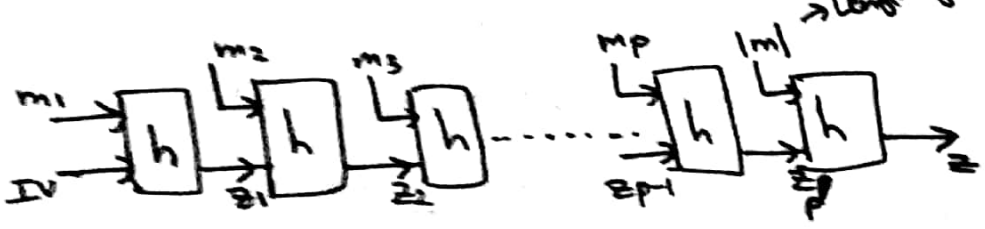
---

Note:
⇒ SHA is Merkle Damgard performed multiple times to get a practically resistant function (hashing).

Proof:

- $H(m)$: is defined as follows,

Let, $m = \underbrace{m_1}_{n \text{ bits}} || m_2 || \ldots || m_p$ } Pad with zeroes if not a multiple of n.



$H(m) = z$

Suppose we can find a collision for H.

↳ $H(x) = H(y)$ where $x \neq y \rightarrow \exists i : x_i \neq y_i$

- Now at each of iterations of h, if output (z_i) is same & inputs (z_{i-1}) are not same then h is not ∄ collision resistant. Since IV is same, we will find a collision for h.
- But we know h is collision resistant,

∴ It is a contradiction & H is collision resistant

⇒ Now, if we have a h, then we can get H.

✱ $h : G \times G \rightarrow G$ ∃ Task to build this.

- Consider $\mathbb{Z}_p^*$: g → Generator
  ↓
  Mult mod p

DLP:

✱ • Let f be the element for which we need $\log_g f$ (or) ↓ Since discrete log is hard, this is secure

✱ Let $f \xleftarrow{R} G$, → Indexed by f.

Let $h^f(x, y) = g^x \cdot f^y$

$g^x * f^y \bmod p$ (in $\mathbb{Z}_p^*$)

- A collision means,

$h^f(x_1, y_1) = h^f(x_2, y_2)$

where $(x_1, y_1) \neq (x_2, y_2)$

$\Rightarrow g^{x_1} \cdot f^{y_1} = g^{x_2} \cdot f^{y_2}$

$g^{x_1 - x_2} = f^{y_2 - y_1}$ } In mod p

- If $y_2 = y_1$, then $g^{x_1 - x_2} = 1 \Rightarrow x_1 = x_2$

∴ Both are same, but given $(x_1, y_1) \neq (x_2, y_2)$

∴ $x_1 \neq x_2$ & $y_1 \neq y_2$

Now since $y_2 - y_1 \neq 0$

$\Rightarrow (x_1 - x_2) \equiv (y_2 - y_1) \log_g f$ (In mod $(p-1)$) } Does not belong to $\mathbb{Z}_p^*$

$\Rightarrow \log_g f \equiv \frac{(x_1 - x_2)}{(y_2 - y_1)}$ } Hard to solve.

↑ Discrete log problem.

→ mod $(p-1)$

- Inverse of $(y_2 - y_1)$ might not exist, if its not invertible. — Note below

∴ In order to find a collision, we need to essentially find a discrete log which is hard. ∴ Finding a collision is hard.

- Note: $(y_2 - y_1)$ is not invertible if $\gcd((y_2 - y_1), p-1) \neq 1$

$\Rightarrow (x_1 - x_2) \rightarrow$ [redacted] is divisible by gcd.

∴ If we remove gcd out from $\frac{(x_1 - x_2)}{(y_2 - y_1)}$ } In its simplest form, denominator gcd with $p-1$ is 1.

∴ It is invertible.

---

* Birthday Attack:

- Prob of 2 people having same birthday = p
- Num n so that prob of 2 people having same birthday is? → Around $\sqrt{365}$. $\simeq 23$.

Consider, a generic question. → Can be used to solve birthday attack.

- There are n items, pick any q items one by one with replacement. Prob[collision] = ?

Theorem: → we show $\frac{q(q-1)}{4n} \leq \text{Prob[collision]} \leq \frac{q^2}{2n}$

Proof: Prob(collision) = 1 − Prob[No collision]

$= 1 - \{P[\text{No col\_1}] \cdot P[\text{No col\_2} \mid \text{No col}] \cdots\}$

$$= 1 - \{P[\text{No-col-1}] \cdot P[\text{No-col 2} | \text{No-col 1}] \cdot P[\text{No-col 3} | \text{No-col 2}] \cdots \cdot P[\text{No-col q} | \text{No-col q-1}]\}$$

$$= 1 - \{1 \cdot (1 - \tfrac{1}{n}) \cdot (1 - \tfrac{2}{n}) \cdots \cdots (1 - \tfrac{q-1}{n})\}$$

$$P[\text{col}] = 1 - \left\{\prod_{j=1}^{q-1} (1 - \tfrac{j}{n})\right\}$$

* Note: $0 \le x \le 1$, $1 - x \le e^{-x} \le 1 - \frac{x}{2}$

$$P[\text{col}] \ge 1 - \prod_{j=1}^{q-1} e^{-j/n} = 1 - e^{\frac{-1}{n}\sum_{i=1}^{q-1} j}$$

$$\therefore P[\text{col}] \ge 1 - e^{\frac{-(q-1)q}{2n}}$$

But we know $1 - x \le e^{-x} \le 1 - \frac{x}{2}$

$$\therefore P[\text{col}] \ge \frac{(q-1)(q)}{4n}$$

- Now for the other bound, (union bound?)

$$P[\text{col}] \le \frac{qC_2}{n} = \frac{q(q-1)}{2n}$$

($\frac{1}{n}$ = prob of collision & qC_2 possible ways)

$$\Rightarrow P[\text{col}] \le \frac{q^2}{2n}$$

Note: ∴ The birthday attack shows that collision resistance is obtained only for square root. ie: If there are 2^x possible outputs (hashes), a collision can be detected by $2^{x/2}$ itself & not 2^x. So whatever bound you set, use a double length hash for safety.

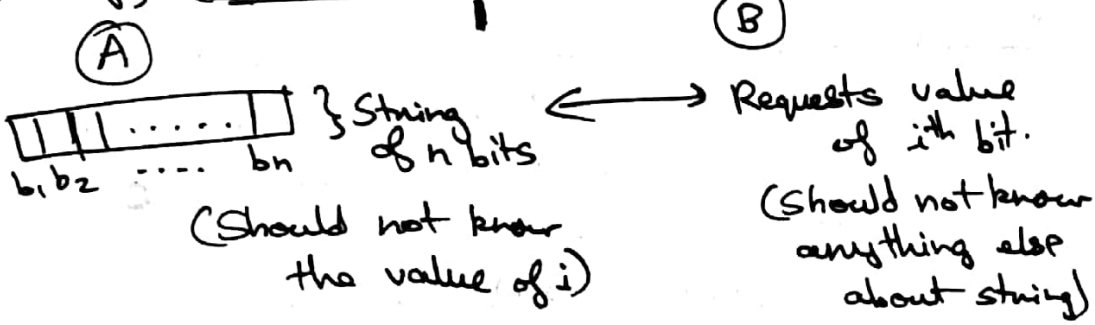
11

Q) How to query a database

a) Without revealing the query

b) And the database reveals nothing about the database except the result.

* Basically, (Oblivious transfer)



- We need a trapdoor one way function in order to solve this. (ex: RSA, ElGamal)

* System: (Solution)

* ① B chooses n bits at random

$$\beta_1, \beta_2, \ldots, \beta_n$$

Using A's public key:

② B applies f_x only on β_i to get,

$$Z = \langle \beta_1, \beta_2, \ldots, f(\beta_i), \ldots, \beta_n \rangle$$

③ B sends Z to A.

Only A can decrypt:

④ A decrypts (finds f^{-1} of Z) → Since we use a trapdoor one way function

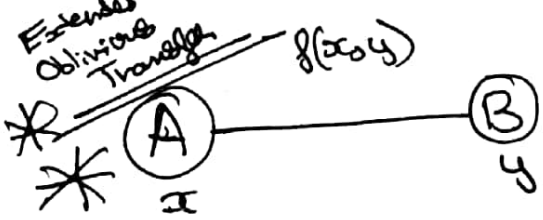
$$Y = f^{-1}(\beta_1), f^{-1}(\beta_2), \ldots, \beta_i, f^{-1}(\beta_{i+1}), \ldots, f^{-1}(\beta_n)$$

⑤ A computes $Y \oplus X$

$$C = \langle b_1 \oplus f^{-1}(\beta_1), \ldots, b_i \oplus \beta_i, \ldots, b_n \oplus f^{-1}(\beta_n) \rangle$$

⑥ A sends C to B

⑦ B obtains $b_i = C[i] \oplus \beta_i$



- A generalized version of oblivious transfer

→ f is known, A should not know y and B should not know x but we want to know f(x, y).

Note: This problem can be generalized to two PCs with memory Mem(A) & Mem(B) and they went to simulate machine S without revealing their memories & not having complete control of S.

- This can further be generalized to n machines working together.

Solution: (to extended oblivious transfer)

- Useful for secure data mining etc. (Share data without revealing what it is)

1) Let $z = x \oplus y$ & $z = f(x,y)$ → Secure simulated machine

↓ Ans

Storage: 

- If x is stored in S, then x_A in A & x_B in B at the same location (Isomorphic rows)

- Similarly y is stored in S.

(Basically A has x_A, y_A & B has x_B, y_B which they don't want to reveal)

$x = x_A \oplus x_B$

$y = y_A \oplus y_B$,

$z = z_A \oplus z_B$

Code for A:

$z_A \leftarrow (x_A \oplus y_A)$

Code for B:

$z_B \leftarrow (x_B \oplus y_B)$

- Without revealing x to B & y to A, we are able to simulate S.

∴ We store data on S as shown, we can compute XOR as shown. To create a universal basis, we need the AND operation.

---

ii) $\Rightarrow z = x \wedge y$

* • A should have z_A & B should have z_B such that $(z_A \oplus z_B) = (x_A \oplus x_B) \wedge (y_A \oplus y_B)$

Since data storage is isomorphic with XOR.

Code for A:

- $z_A \xleftarrow{R} \{0,1\}$ } Randomly choose z_A (If not random, then B could be able to get data on S)
- Create array of size 4 | b_0 | b_1 | b_2 | b_3 |

↳ Arr[i] = value of z_B if x_B, y_B are as follows

x_B	y_B	z_B
0	0	$(x_A \wedge y_A) \oplus z_A = b_0$
0	1	$(x_A \wedge \bar{y}_A) \oplus z_A = b_1$
1	0	$(\bar{x}_A \wedge y_A) \oplus z_A = b_2$
1	1	$(\bar{x}_A \wedge \bar{y}_A) \oplus z_A = b_3$

Can't prove functions 1 way

Code for B:

Uses trapdoor 1 way function (the only thing keeping these methods from being used in market)

← • Run oblivious transfer with array of size 4 (Database of 4 elements) (Inputs x_B, y_B → Result z_B)

∴ To store z_B, it just queries for all x_B, y_B present

∴ Finally z_A & z_B are stored securely & $z = z_A \oplus z_B$ //

iii) Load & Store instructions can be done as,

A: $x \oplus r_1$ ⟵ B: $y \oplus r_1$

A: r_1 ⟶ B: r_1

A stores r_1. B stores $y \oplus r_1$. Together they can get back x or y from storage.

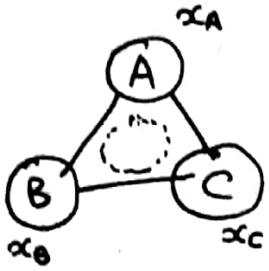
} Machines don't record what x or y are but they send R if needed to store, and then doing a XOR again will get x or y.

Note: If there are n people & t are actively faulty (give junk answers) → If $n > 3t$ then we can even cluster based computing.

---

(12) (Distributed)

* • Secure Multiparty computation:
  ↳ Adversary isn't omnipresent. (But isn't polynomially bounded either)



• Need to compute $f(x_A, x_B, x_C)$ keeping x_A, x_B & x_C private.

• If adversary is eavesdropping at one of A, B or C but we don't know → 1/3 secure multiparty computation.

---

* ⇒ Key Management: (A similar problem to secure multiparty computation)

• Keys need to be kept secure
  ↳ Can this be stored in a network with n shares such that any t shares reveals nothing about the key & $\geq t+1$ shares together reveal the secret?

*Shannon's Secret sharing scheme:

* • Secret $S \in F$ } Finite field,

Lets say $S \in Z_p$ } For simplification (Operations are + %p & * %p)

⇒ Choose a polynomial of degree t

$$P(x) = \sum_{i=0}^{t} a_i x^i$$ } a_0 = Secret } $S = P(0)$ & rest a_i's are chosen randomly from Z_p.

• Share $s_i = P(i)$

$i = 1$ to n } n shares

⇒ Upto t shares, no info is revealed & $t+1$ or more shares reveals everything.

---

ex: If $t=2$, $P(x) = S + a_1 x + a_2 x^2$

$P(1) = S + a_1 + a_2$

$P(2) = S + 2a_1 + 4a_2$

⇒ If we have $P(3)$ then we have 3 equations & 3 variables, can be solved.

---

* Basically a matrix, (Vandermonde matrices)

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 2 & 2^2 & \cdots \\ 1 & 3 & 3^2 & \cdots \\ \vdots & & & \end{bmatrix} \begin{bmatrix} S \\ a_1 \\ a_2 \\ \vdots \\ a_t \end{bmatrix} = \begin{bmatrix} P(1)=s_1 \\ P(2)=s_2 \\ \vdots \\ P(t+1)=s_{t+1} \end{bmatrix}$$

↑ This is a full rank invertible matrix.

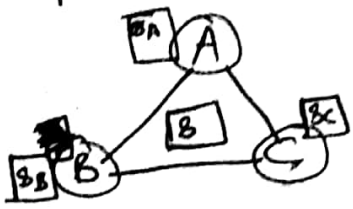
• Originally $P[S=s] = 1/p$ } Before revealing the shares,

we can show that ↗ with shares revealed.

$$P[S=s \mid S_1=s_1, S_2=s_2, \ldots S_t=s_t] = \frac{1}{p}$$

Secure Multiparty computation

Now applying this to secure multiparty computation, we have.



$n=3, t=2$
∴ All 3 together can reveal S else not.

(Polynomial of degree 1)

⇒ If we want to have $n=3$, $t=1$, we can have A, B, C store 3 collinear points and secret = y-intercept. We would need atleast 2 shares to reveal secret.

$s_A = s+s$
$s_B = 2s+s$
$s_C = 3s+s$

$P(x) = sx + s$

⇒ We show that for $t=1$ case, we don't need any one way functions.

Proof: We assume our instruction set consists of ADD & MUL operations.

* *)

$z \leftarrow x+y$

↳ $x_A, y_A, x_B, y_B, x_C, y_C$

⇓

z_A, z_B, z_C

⇓

z

- Earlier we used XOR with 2 parties

Let,

$z_A \leftarrow x_A + y_A$
$z_B \leftarrow x_B + y_B$
$z_C \leftarrow x_C + y_C$

} we show this is correct and z would be constant intercept of z_A, z_B, z_C.

⇒

$P_x(1) = x_A$	$P_x(2) = x_B$	$P_x(3) = x_C$
$P_y(1) = y_A$	$P_y(2) = y_B$	$P_y(3) = y_C$
$x_A + y_A = P_x(1) + P_y(1) = P_{x+y}(1)$	$x_B + y_B = P_{x+y}(2)$	$x_C + y_C = P_{x+y}(3)$

∴ These points lie on a 1degree polynomial P_{x+y} where $z_A > z_B$ & z_C lie on P_{x+y}, ∴ The constant term for this is $z = P_{x+y}(0)$

$= P_x(0) + P_y(0)$

$= x + y$ //

∴ The addition instruction is adding shares, & this is because [redacted] polynomials are additively homomorphic.

2) For MUL operation,

$z \leftarrow x * y$

$z_A \leftarrow x_A * y_A$
$z_B \leftarrow x_B * y_B$
$z_C \leftarrow x_C * y_C$

} This won't work since the degree of the polynomial increases. (Cascaded multiplication becomes an issue)

⇒ Also final polynomial z is a reducible polynomial so it doesn't qualify shannon's secret sharing since its not a randomly chosen polynomial

⇒ We do this,

$z'_A \leftarrow x_A * y_A$ → z'_{AA}, z'_{AB}, z'_{AC}
$z'_B \leftarrow x_B * y_B$ → z'_{BA}, z'_{BB}, z'_{BC}
$z'_C \leftarrow x_C * y_C$ → z'_{CA}, z'_{CB}, z'_{CC}

} z'_{xy} are sent to share holder y.

2 degree polynomial

⇒ $\lambda_A z'_A + \lambda_B z'_B + \lambda_C z'_C = z$

Lagrange coefficients { λ_A, λ_B & λ_C are the inverse of vandermond matrix's first row

(Using lagrange interpolation)

⇒ We also know,

$$z_A' = \lambda_A' z_{AA}' + \lambda_B' z_{AB}' + \lambda_C' z_{AC}'$$

This is possible since z_A' is a 2 degree polynomial

Similarly z_B' & z_C'

$$\therefore z = \lambda_A \left[\lambda_A' z_{AA}' + \lambda_B' z_{AB}' + \lambda_C' z_{AC}'\right] + \lambda_B \left[\lambda_A' z_{BA}' + \lambda_B' z_{BB}' + \lambda_C' z_{BC}'\right] + \lambda_C \left[\lambda_A' z_{CA}' + \lambda_B' z_{CB}' + \lambda_C' z_{CC}'\right]$$

$$= x * y$$

⇒ Here z_{AA}', z_{BA}' & z_{CA}' are known to A and similarly for B & C.

$$\Rightarrow z = \underbrace{\lambda_A \lambda_A' z_{AA}' + \lambda_B \lambda_A' z_{BA}' + \lambda_C \lambda_A' z_{CA}'}_{\text{Known to A.}}$$

$$\text{Known to B} \left\{ + \lambda_A \lambda_B' z_{BA}' + \lambda_B \lambda_B' z_{BB}' + \lambda_C \lambda_B' z_{BC}' \right.$$

$$\text{Known to C} \left\{ + \lambda_A \lambda_C' z_{CA}' + \lambda_B \lambda_C' z_{CB}' + \lambda_C \lambda_C' z_{CC}' \right.$$

$$+ z_A + z_B + z_C$$

} Let the above z be written as such.

$$\Rightarrow z = \mu_A z_A + \mu_B z_B + \mu_C z_C$$

} We choose μ_A, μ_B, μ_C such that $z_A = P(1)$, $z_B = P(2)$ & $z_C = P(3)$.

$$\therefore z_A = \frac{\lambda_A \lambda_A' z_{AA}' + \lambda_B \lambda_A' z_{BA}' + \lambda_C \lambda_A' z_{CA}'}{\mu_A}$$

and similarly others.

⇒ But we know $M_A = \lambda_A$, $M_B = \lambda_B$, $M_C = \lambda_C$?

(Rabin's scheme)

∴ Essentially, multiply locally & then reshare as $z_{AA}, z_{AB}, \ldots$ etc to ensure the properties are maintained.

⇒ With these ADD & MUL operations, we can create a general shared secret computation device.

Note: This will work for any n & $t < n/2$, this method can be extended.

---

(13) Generalized Secret Sharing

- Beginning towards a major invested problem in cryptography.

* ⇒ Access structure: All the subsets that can together obtain key. So these & supersets can access.

(In Shamir: The access structure is all subsets of size $t+1$)

- The access structure could also be viewed as $f : \{0,1\}^n \to \{0,1\}$ where $\{0,1\}^n$ denote any subset of shares s_i.
- f is monotonic in the sense that, If $f(S) = 1$ then $f(S') = 1$ where S' = superset of S.

- Without monotonicity, total possible functions f are $O(2^{2^n})$
  - ↳ Doubly exponential
    - ↳ ∴ This requires an exponential length of secret sharing scheme.

      Communication + Computation = Scheme
    - So we don't know if both are exponential here or just one etc.

Open Problems

* 1) Does there exist an access structure [redacted] s.t that every secret sharing scheme for it has shares of super polynomial length.

2) Do all efficiently computable access structures have efficient sharing schemes?

⇒ Generalized fault tolerance, given f,
* our system should be fault tolerant
* to all those subsets where $f(S)=0$.
* But this would require us to provide f as a part of our input. So what is the size of our input?
  - ↳ The input will have the basis of access structure. If we say simple fault tolerance of t nodes, basis has ${}^{n}C_{t+1}$ elements.
    - ↳ Now since input is not polynomial we can argue that our non polynomial algorithms run in polynomial time of input and hence is efficient

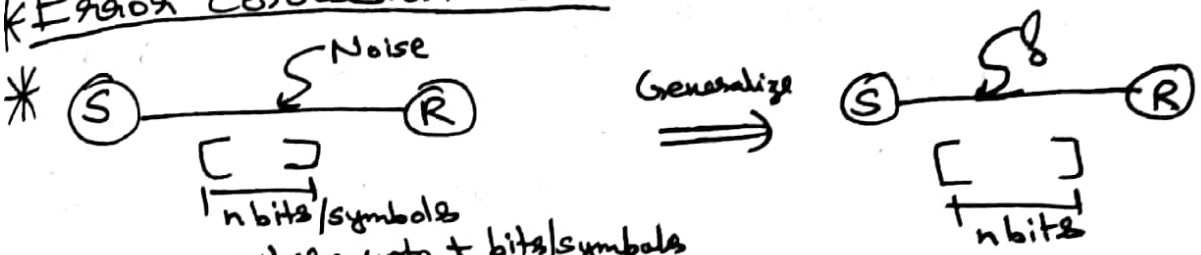
→ We could think of encoding f and sending as inputs but they have their share of problems as well.

↳ If we have an encoding that is small but expanding it out to f is quite time taking etc then we need to decide which of the two we use to quantify as the input size.

Open Problems

3) Assuming adversary is a PPTM, negl. error is allowed and one way functions exist — Do all efficiently computable access structures have efficient sharing schemes.

Error correction codes:



- Any subset can be in error if $f(S) = 0$

- We can specify error limits in multiple ways.

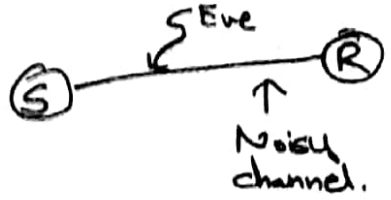
These are not the same.
- ex: First $n/2$ bits at max 20% toggle & next $n/2$ bits at max 20% toggle allowed.
- ex: 22.5% max toggle allowed.

- The channels for which $f \notin P$ is unrealistic since a channel is not capable of solving NP problems in P time.

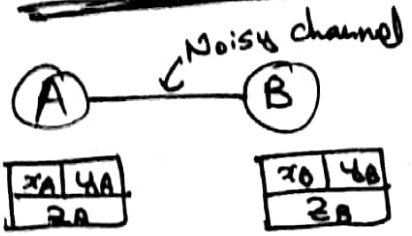
↳ So now, do we have efficient error correction codes (ECC) for all realistic channels?

→ In order to make f probabilistic, we also have $\{0,1\}^n$ as input to f along with $\{0,1\}^n$. This makes it harder to talk about input size.

(14) Cryptography from channel noise:



→ Secure AND



$z \leftarrow x \wedge y$

$x = x_A \oplus x_B$, $y = y_A \oplus y_B$ } Secure storage

$(z_A \oplus z_B) = (x_A \oplus x_B) \wedge (y_A \oplus y_B)$

- Secure XOR was proved earlier but secure AND with noiseless channels required one way functions.
- Assume noise is 1 out of 4 noise (one bit is toggled in every 4 bits)

① A sends 4 random bits say, $\langle r_0, r_1, r_2, r_3 \rangle$ to B.

② B receives these four bits $\langle s_0, s_1, s_2, s_3 \rangle$ where 3 of them match but 1 is toggled.

③ Consider $\langle r_0 \oplus s_0, r_1 \oplus s_1, r_2 \oplus s_2, r_3 \oplus s_3 \rangle$, three are zero but one of them is one.

④ Let $M = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}_{4\times 3}$ (Public matrix) (Truth table for AND)

- A computes $\langle r_0, r_1, r_2, r_3 \rangle \cdot M = \langle a_A, b_A, c_A \rangle$

• B computes $\langle a_B, b_B, c_B \rangle \cdot M$

$= \langle a_B, b_B, c_B \rangle$

→ We will show that $c = a \wedge b$

where $c = c_A \oplus c_B$, $a = a_A \oplus a_B$, $b = b_A \oplus b_B$

Proof:

• $\langle a, b, c \rangle = \langle a_A \oplus a_B, b_A \oplus b_B, c_A \oplus c_B \rangle$

$= \langle r_{a_A} \oplus r_{a_B}, r_{b_A} \oplus r_{b_B}, r_{c_A} \oplus r_{c_B} \rangle \cdot M$

(We know that these are 3 pairs)

∴ $\langle a, b, c \rangle$ = One random row from M.

∴ $\langle a, b, c \rangle$ satisfy $c = a \wedge b$ which means that we were able to do a secure AND for random noise input.

(We need to convert this to an AND on our provided inputs)

(From here, we assume it's a noiseless channel) → use error correction codes to handle noise

⑤ A sends $(x_A \oplus a_A)$ to B and B sends $(y_A \oplus b_A)$ to A. $(y_B \oplus b_B)$ $(x_B \oplus a_B)$ (This does not reveal any info since a_A blinds B from x_A & similarly b_B blinds A from y_B)

⑥ A computes

$(x_A \oplus a_A \oplus x_B \oplus a_B) = X$

$(y_A \oplus b_A \oplus y_B \oplus b_B) = Y$

B computes

$(x_B \oplus a_B \oplus x_A \oplus x_A) = X$

$(y_B \oplus b_B \oplus y_A \oplus b_A) = Y$

∴ Both A & B know X, Y

→ Now, $x \wedge y = (x_A \oplus x_B) \wedge (y_A \oplus y_B)$

We show,

$$x \wedge y = (x \oplus a) \wedge (y \oplus b) \oplus x \wedge (y \oplus b) \oplus y \wedge (x \oplus a) \oplus (a \wedge b) \longrightarrow (2)$$

(On expansion, $xy + xb + ay + ab + xy + xb + yx + ya + ab$

(∵ In boolean arithmetic, this is valid and only xy remains) $= xy$)

$$\therefore x \wedge y = (x_A \oplus x_B) \wedge (y_A \oplus y_B)$$

(From (2)) $$= (X \wedge Y) \oplus (x_A \oplus x_B) \wedge (Y) \oplus (y_A \oplus y_B) \wedge (X) \oplus (c_A \oplus c_B)$$

(1) A defines z_A as,

$$z_A = (X \wedge Y) \oplus (x_A \wedge Y) \oplus (y_A \wedge X) \oplus c_A$$

(All of these terms A knows)

B defines z_B as,

$$z_B = (x_B \wedge Y) \oplus (y_B \wedge X) \oplus c_B$$

(All of these terms B knows)

∴ We have created a secure AND using a noisy channel.

• Now, based on this we can build one way functions etc.

⇒ However, we started with an assumption that 1-out-of-4 noise exists, we need to try overcome this.

• Before starting our method, we do a * check to see if only one bit is toggled or not.

↳ We use the secure XOR algorithm first to see how many bits are toggled since bits toggled can be 0, 1 or 2 (Since minority noise channel is used)

↳ If this XOR is 1 (only 1 bit is toggled), ∴ we can perform secure AND. If XOR is 0 that means 0 or 2 bits are toggled and hence we try again.

⇒ This can also be extended to practical purposes where even 3 or 4 bits are toggled.

Note: 3 party computation can detect an eavesdropper but with 4 parties, a byzantine party can be detected & corrected.

(Using 4 points on a straight line instead of 3 points on a straight line)

• We know how to fix errors if error < 1/3, then the only assumption needed on the channel is P(1 out of 4) > P(3 out of 4).

⇒ This security need not require channel noise specifically. Any uncertainty would work. ex: We can use race conditions in the

---

Conjecture: Buggy software is 'more secure' than bug free ones.

(15) Byzantine Agreement in Distributed Systems:

* — If one way functions exist then possible.

⇒ Computational adversity
- all modern cryptosystems

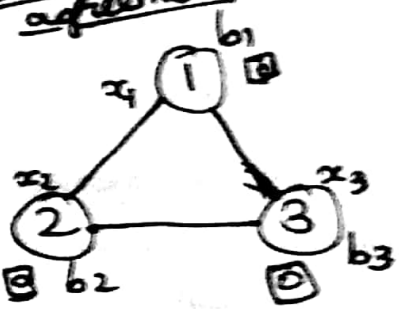
Practical adversity
- Noise based [redacted] Crypto

Cryptographic/Hacking adversary
- ex: Quantum crypto

Natural adversity

? Philosophical adversity

Byzantine agreement:



x_i's ⇒ Inputs
b_i's ⇒ Outputs

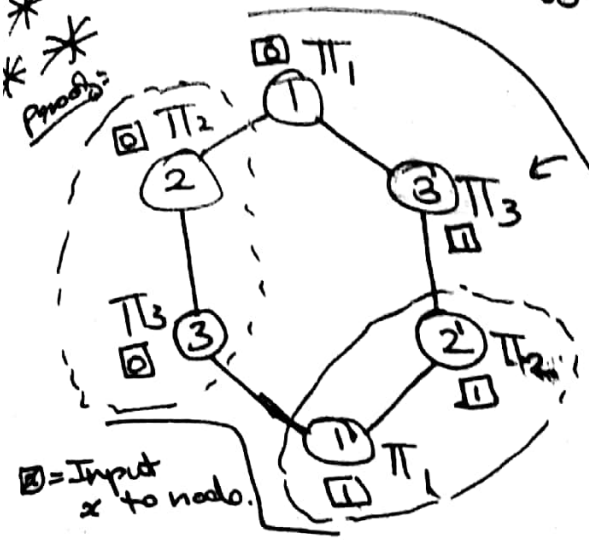
Goals:
- Agreement: All non faulty nodes have the same/equal output
- Validity: The output of all non faulty nodes is equal to the input of one nonfaulty node.

⇒ If one node is byzantine faulty the agreement system will still work, but with two byzantine nodes won't work.

Theorem:

⇒ Let Π be a byzantine agreement protocol (BA protocol)

Proof:



→ In a 3 node system

- Π_1, Π_2, Π_3 correspond to the codes for the three nodes of the system.

- We define Π' as a new hypothetical program. (Each of the 6 nodes run their associated program)

- This is syntactically correct as well since every Π_i talks to Π_j where $j \neq i$. → Similar to original Π protocol.

(Hexagon Proof)

⇒ We now show that if Π is byzantine, it will follow goals (from previous page) and due to this Π' does not provide outputs but this is a contradiction and hence there can't be BA protocols.

- If we consider the dotted pairs of nodes 2,3. (2) gets an output of β whereas (3) gets an output of α.

- Now in the original byzantine setup of 3 nodes, since node (1) is faulty it sends β to (2) & α to (3). ∴ It is not possible to distinguish between the original 3 node system & this 6 node system.

(In second system)

- Now (2), (3) know that (1) is faulty so since its byzantine, (2), (3) output (0) (Their input)

- Looking at 1', 2', if ③ is faulty (byzantine) then similar to earlier we cant differentiate which network they belong to.

  ∴ 1', 2' must output 1.

- Looking at 3, 1', if ② is faulty then similar to earlier we cant distinguish. Though outputs of 3, 1' should be the same but they already decided on different outputs

  ∴ It is a contradiction, so the second system does not exist (if it has to be byzantine)

∴ No such protocol Π exists. //
(BA)

---

(16) Mid 2 topics:

1. Hashing schemes
   - Collision resistance
   - Birthday attacks
   - Merkle Damgard transform
   - DLP based design and collision resistant hash functions
2. Public key cryptography
   - Key establishment
     - ↳ Diffie Hellman
   - El Gamal scheme (CPA-secure)
   - RSA PKC-scheme
     - ↳ Attacks on textbook RSA } using chinese remainder theorem
   - Padded RSA
   - PKCSv1.5
3. Oblivious transfer
   - From any 1-way permutation

- General secure two party computation (of any function)
  ↳ Secure XOR and AND.

④ Secret sharing
- Shamir's Algorithm / scheme
  (points on a t degree polynomial)
- General multi party schemes (secure computation)

⑤ General access structures ($f: \{0,1\}^n \to \{0,1\}$)
- Secret sharing schemes for hashing access structures
- Computational secret sharing

(Assigning weights i.e. num of shares per person can help us accomplish this)

⑥ Noise based secure protocols
- General 2-party secure computing
- Oblivious transfer.

⑦ Byzantine agreement
- Impossibility of 1-out-of-3 case.

Sample questions (Practice question sheet)
↳ Show elgamal scheme is CPA secure

---

⑧ Digital signatures

---

* Digital Signatures:

- Appending a normal signature to the message is not useful.

∴ We want signatures to be a function of the message itself. $\langle m, \sigma_k(m) \rangle$

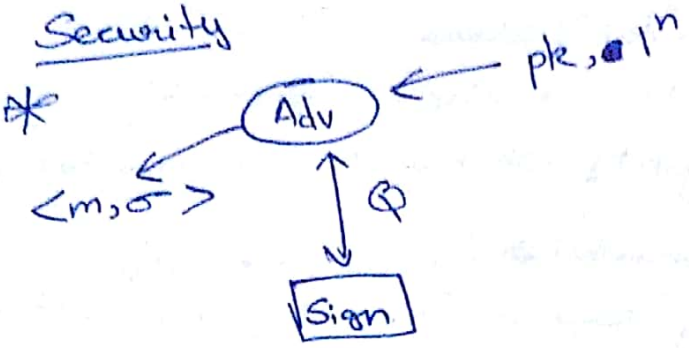
Scheme: $\langle Gen, Sign, Vrfy \rangle$

$Gen(m) \longrightarrow \langle sk, pk \rangle$ (secret key & public key)

$Sign_{SK}(m) = \sigma$

$Vrfy_{pk}(m, \sigma) = Yes/No$

Security



$P[Vrfy(m,\sigma) = Yes] \leq negl(n)$ if $m \notin Q$

⇒ Consider RSA,

↳ RSA-Sign : $\sigma = m^{d} \bmod N$ (d → secret key)

↳ RSA-Verify : $\langle m, \sigma \rangle$, $\sigma^{e} \bmod N = m$ (e → public key)

---

* Textbook attack on RSA for signatures

i) • Adv: Chooses random ρ, computes $\rho^{e} \bmod N$ } [redacted]

(e = Public key)

$\Rightarrow m = \rho^{e} \bmod N$, $\sigma = \rho$ } $Vrfy(m, \sigma)$

$\sigma^{e} \bmod N = \rho^{e} \bmod N = m$

∴ Verification passes.

⇒ [redacted] If $\langle m_1, \sigma_1 \rangle$, $\langle m_2, \sigma_2 \rangle$ pass Vrfy

ii) then $\langle m_1 m_2, \sigma_1 \sigma_2 \rangle$ also passes Vrfy.

$m_1^{d} \bmod N = \sigma_1$, $m_2^{d} \bmod N = \sigma_2$

$(m_1 m_2)^{d} \bmod N = \sigma_1 \sigma_2$

• Due to this as well, textbook RSA for signatures is bad as well.

* "Hash and Sign" paradigm: (To secure RSA signatures)

⇒ Hashed RSA: Sign: $\sigma = [H(m)]^d \bmod N$

Verify: m, σ, $H(m) = \sigma^e \bmod N$

- Here the previous attack won't work,
  - ↳ $\langle m_1, \sigma_1 \rangle, \langle m_2, \sigma_2 \rangle$ work

$$\sigma_1 = (H(m_1))^d \bmod N$$

$$\sigma_2 = (H(m_2))^d \bmod N$$

$$\sigma_1 \sigma_2 = (H(m_1) \cdot H(m_2))^d \bmod N$$

- Since hash algorithms are generally not homomorphic, $H(m_1 m_2) \neq H(m_1) H(m_2)$.

∴ $\langle m_1 m_2, \sigma_1 \sigma_2 \rangle$ does not pass verify.

---

* Publicly verifiable: ("Transferable" Trust is) (Certification Authorities)

- An adversary can sent out fake public keys.
- Certification authorities are used in browsers where the public keys of CAs are already present in the browser.
- Now user requests digital certificate instead of just the public key. Digital certificate is the public key of some xyz signed with CA's private key. Hence it can be trusted & adversary can't forge the CA and hence fails.

PKI: List of all CAs and their public keys. (that we trust)

Public key infrastructure.

——END of Mid 2——

17