Shannon security

- unlimited computational power,
- perfect secrecy can only be achieved when the size of the shared secret is the same as the size of the message
- seeing the ciphertext reveals no information whatsoever about the message

Modern crypto

- PPT adversaries
- PPT encryption and decryption algorithms
- security parameter k
- **impossibility** of breaking an encryption system **turns into infeasibility** of breaking an encryption system
    - encryption is secure against PPT adversaries but not against computationally-unbounded adversaries

WEP

- How it works?
    - **plaintext** $p = \langle m, CRC(m) \rangle$
    - for each transmitted frame, a random **initialization vector** (IV) v is generated
    - **keystream** $K_s = RC4(v, k)$
    - **ciphertext** $c = p \oplus K_s$
    - **frame** $f = \langle v, c \rangle$
- Vulnerabilities
    - IV collisions (24 bit IV space is too small), after 6000 frames a collision is bound to occur
    - secret shared key k is rarely changed
    - man in the middle attack, can modify frames because CRC32 is not a cryptographic hash function
        - $CRC(p_1 \oplus p_2) = CRC(p_1) \oplus CRC(p_2)$
    - reaction attack because of ACKs sent by underlying 802.11 layer
- How to fix WEP?
    - Bigger IV space to avoid collisions
    - RC4 is pretty bad, use something better like a cryptographic hash
    - Use a MAC instead of CRC32
    - 802.11

Computational indistinguishability

- D_0 and D_1 are $(\boldsymbol{t}, \boldsymbol{\varepsilon})$**-computationally indistinguishable** if $\forall$ algorithms A ($A(x) = 0, if\ x \in D_0$ and $A(x) = 1, if\ x \in D_1$) running in time $\leq t$, it holds that
    - $Adv\ A = |\Pr[A(x) = 0 | x \leftarrow D_0] - \Pr[A(x) = 0 | x \leftarrow D_1]| \leq \varepsilon$
        - Pr[guess correctly]-Pr[guess incorrectly]
    - Advantage is a measure of how successfully you can attack a cryptographic algorithm

DPI

- If $D_0 \overset{t}{\underset{\varepsilon}{\sim}} D_1$ and f is a function running in time t' then $f(D_0) \overset{t - t'}{\underset{\varepsilon}{\sim}} f(D_1)$

- If $D_0 \overset{t}{\underset{\varepsilon}{\sim}} D_1 \overset{t}{\underset{\varepsilon'}{\sim}} D_2$ then $D_0 \overset{t}{\underset{\varepsilon+\varepsilon'}{\sim}} D_2$.

PRNGs

- Criteria
  - $G(k)$ needs to be pseudo-random (look like a randomly generated string)
  - G needs to be hard or impossible to invert
  - Given n bits of $G(k)$, an attacker should not be able to correctly guess the $(n+1)$th bit
  - G needs to be an injective function: $x_1 \neq x_2 \Rightarrow f(x_1) \neq f(x_2)$
  - G needs to be efficient, and not take too long to execute
  - G needs to be hard to recognize
- A pseudo-random generator $G: \{0,1\}^l \rightarrow \{0,1\}^L$ is $(\boldsymbol{t}, \boldsymbol{\varepsilon})$**-secure** if $G(U_l) \overset{t}{\underset{\varepsilon}{\sim}} U_L$
- **Extending a PRNG** from $G: \{0,1\}^l \rightarrow \{0,1\}^{l+1}$ to $G': \{0,1\}^l \rightarrow \{0,1\}^{l+2}$, yields a $(t - t', 2\varepsilon)$-PRNG
  - $G'(s) = G\big(G(s) - last\ bit\ of G(s)\big) + last\ bit\ of\ G(s)$
- **Concatenating PRNGS:** If $G_1: \{0,1\}^{l_1} \rightarrow \{0,1\}^{L_1}$ is (t_1, ε_1)-secure PRNG running in time t_1' and $G_2: \{0,1\}^{l_2} \rightarrow \{0,1\}^{L_2}$ is (t_2, ε_2)-secure PRNG running in time t_2' then $G_1 \parallel G_2$ is $(t_3, \varepsilon_1 + \varepsilon_2)$-secure PRNG, with $t_3 = \min(t_1 - t_2', t_2)$.
- $G(x) = (AES(x,0), AES(x,1), AES(x,2) \ldots)$, x will be the AES key, is a secure PRNG

PRFs

- Guarantee of a PRF is that all its outputs appear random given that the PRF was chosen randomly
- $F_k: \{0,1\}^l \rightarrow \{0,1\}^L$ How many such functions are possible? $(2^L)^{2^l}$
- **DPI for PRFs:** If f runs in time t' and F is a (t, q, ε)-PRF then $f(F_k)$ is a $(t - O(q), q, \varepsilon)$-indistinguishable from $f(R)$.
- **Transitivity:** If $F_k \overset{t,q}{\underset{\varepsilon}{\sim}} G_k \overset{t,q}{\underset{\varepsilon'}{\sim}} H_k$ then $F_k \overset{t,q}{\underset{\varepsilon+\varepsilon'}{\sim}} H_k$
- **Building PRGs from PRFs:** If F_k is a (t, q, ε)-PRF then $G(k) = (F_k(0), F_k(1), \ldots F_k(q))$ is a (t, ε)-secure PRG.

PRPs

- $P_k: \{0,1\}^n \rightarrow \{0,1\}^n, k \leftarrow \{0,1\}^l$
- A subset of PRFs in some sense. P_k is injective and surjective ⇒ P_k is bijective ⇔ P_k is invertible
- From an attacker's point of view $PRPs$ and $PRFs$ look almost identical. The only way to distinguish between then is by looking if the function is one-to-one.
  - $Funcs_{\{0,1\}^n}^{\{0,1\}^n} \overset{\infty, q}{\underset{\frac{q^2}{2^{n+1}}}{\sim}} Perms_{\{0,1\}^n}$
- A bad example of a PRP would be $P_k(x) = A_k x$, where $A_k \leftarrow A$, a linear transformation essentially where $A = \{n \times n\ invertible\ matrices\ over\ GF(2)\}$,
  - It's bad because you can feed it the right input (vectors with only one 1) and figure out A_k

RoR-security

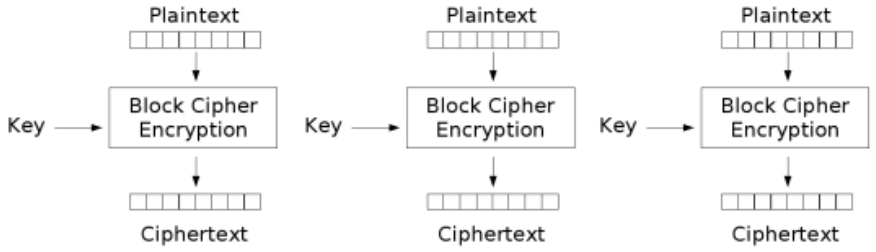
- The goal in RoR security is for Eve to not be able to tell whether she's in the real or in the random world
- **Definition**: An encryption scheme $E_k: \{0,1\}^* \rightarrow \{0,1\}^*$ is $(\boldsymbol{t}, \boldsymbol{q}, \boldsymbol{\varepsilon})$**-real-or-random secure** if ∀ algorithms A running in time t and making at most q bits of queries, then the advantage of A is:
    - $Adv\ A = \left|\Pr[A^{E_k} = 0] - \Pr\left[A^{E_{k^\circ}\$} = 0\right]\right| \leq \varepsilon$

Modes of operation for block ciphers

A mode of operation takes a PRF or PRP and turns it into an encryption scheme.

ECB (Electronic codebook mode)

An attacker could distinguish patterns in the ciphertext when ECB mode is used.



Electronic Codebook (ECB) mode encryption

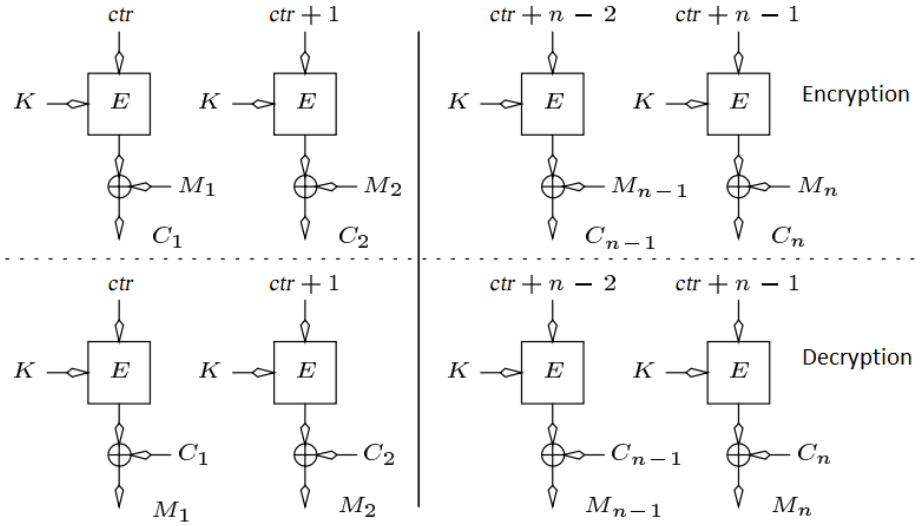
CTR (Counter mode)

Uses a **PRF**. Essentially, it **turns a block cipher into a stream cipher.**

The **IV will be part of the ciphertext**, since the receiver needs it to start decrypting the ciphertext.

In order to keep this secure, you have to **keep track of the IV properly** and **do not reuse IVs**.

What happens if **one of the packets gets lost** (i.e. when the encryption scheme is used along a network connection)? Even if some packets get lost, the recipient can resync with the sender by just looking at the IV in the package.

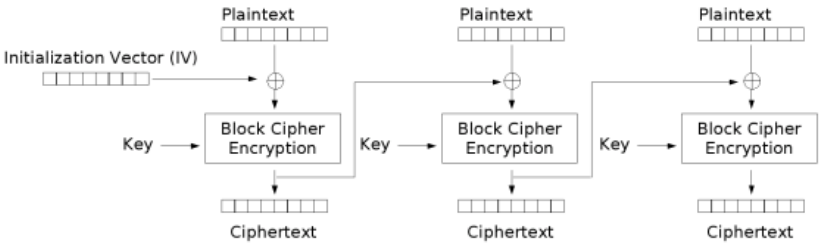


If F_k is a (t, q, ε)-secure PRF then CTR^{F_k} (**counter mode built on F_k**) is a $\left(t - O(q), \min\left(q, \left|IV_{space}\right|\right), \varepsilon\right)$-RoR-secure encryption scheme.

CBC (Cipher block chaining mode)

$$C_i = F_k(P_i \otimes C_{i-1}), C_0 = IV$$

$$P_i = F_k^{-1}(C_i) \otimes C_{i-1}, C_0 = IV$$



Cipher Block Chaining (CBC) mode encryption

If C_i **gets corrupted**, it's going to screw up P_i *and* P_{i+1} but P_{i+2} will decrypt correctly. **Encryption is not parallel** but **decryption is parallel**. F_K **needs to be invertible**. **How do you choose the IV?** Pick it randomly for each message, a.k.a. $CBC^{\$}$, or pick it randomly for the first message and use C_n as IV for the next message and repeat.

Message authentication codes (MACs)

- MACs give you **integrity.** Encryption schemes give you **secrecy**.
- A **MAC is a tuple** $(Gen, Mac, Vrfy)$, where:
  - Gen – generate a key of length n
  - $Mac_k(m)$ – generates a tag (or a MAC) on the message m using the key k generated by Gen as input
  - $Vrfy_k(m, t)$ – used to verify the integrity of the message m that was tagged with the tag t and key k
    - returns 1 when the message is authentic
    - returns $\perp$ when the message has been forged
- **Adversary** can query $Mac_k(m), \forall m \in M_q$ and get a list of tags. The goal is for the adversary not to be able to build a message tag pair (m, t) with $m \notin M_q$ that will be correctly validated by $Vrfy$.

Building MACs

- MACs built on PRFs ($Mac_k(m) = F_k(m)$) are EU-MACs
- Constructing **MACs for fixed length** and **variable length messages**
  - Bad variable length message MACs
    - $t = Mac_k(m_1 \otimes m_2 \otimes \ldots \otimes m_x)$
    - $t_i = Mac_k(m_i), t = \langle t_1, t_2, \ldots, t_x \rangle$
    - $t_i = Mac_k(seq + m_i)$
  - If you have **sequence numbers**, and the **size of the message** (maybe as the first tag t_1), you should be able to build a secure MAC.

Integrity of ciphertext (INT-CCA)

- Mallory wins if after making q queries $m_1 \ldots m_q$, which get mapped to $c_1 \ldots c_q$, he finally manages to make a V_k query $c \notin \{c_1 \ldots c_q\}$ such that $V_k(c) \neq \perp$

- **Definition:** M_k, V_k is a (t, q, q', ε)-secure message integrity code if ∀ algorithms A running in time ≤ t and making $\leq q$ M_k queries and $\leq q'$ V_k queries then:
    - $Adv\ A = \Pr[A^{M_k, V_k}\ wins] \leq \varepsilon$
- Winning, in this case, means successfully forging a message…

IND-CPA

- You have an **oracle**, an adversary who is allowed to keep sending messages to an **encryption oracle** at any point in time: $m_1, m_2 \ldots m_N$ and the oracle will send back $E(m_1), E(m_2), \ldots, E(m_N)$.
- The adversary will send m_1', m_2' (they could be one of the $m_1, m_2 \ldots m_N$) and the oracle encrypts one of them (picks b randomly, 0 or 1) and sends the encryption to the adversary, who's supposed to tell whether m_1' or m_2' was encrypted.
- Therefore, **no CPA-secure scheme can be deterministic**, since the adversary could already have the encryption of m_1' or m_2'.

IND-CCA

- Same as IND-CPA, but the **adversary also has access to a decryption oracle**.
- He would always win since he would query the decryption oracle with the $E(m_b')$ challenge he received from the encryption oracle. Therefore, **the attacker cannot query the decryption oracle with the challenges** that he receives from the encryption oracle.
- This is the **strongest model** of security for a cryptosystem.
- **CPA-secure system that is not CCA-secure:**

$$Enc_k(m) = \langle r, F_k(r) \oplus m \rangle, where\ |m| = n, r \leftarrow U_n\ and\ F_k\ is\ a\ PRF$$

- **Building CCA-secure schemes from CPA-secure ones**
    - CCA attacks rely on the attacker modifying the encryption of the challenge message (just a tiny bit) such that after feeding it to the decryption oracle, the oracle's answer will enable the attacker to differentiate between which message was encrypted
    - By adding an EU-MAC to the ciphertext to ensure that it hasn't been tampered with, the decryption oracle is rendered useless and the scheme becomes CCA secure
        - $c \leftarrow E_{k_1}(m)$ and $t \leftarrow Mac_{k_2}(c)$

Diffie-Hellman key exchange

- Alice and Bob agree on a large number p and a base g
- Alice picks a random number a and computes $x = g^a \bmod p$.
    - Alice sends x to Bob. (Note that a remains secret to Alice)
- Bob picks b random and computes $y = g^b \bmod p$.
    - Bob sends y to Alice. (Note that b remains secret to Bob)
- Alice computes $k = y^a \bmod p = g^{ab} \bmod p$
- Bob computes $k' = x^b \bmod p = g^{ab} \bmod p$
- Alice and Bob now both have a secret key $k = k'$

Diffie-Hellman is vulnerable to the **omen problem**: it is not clear that to compute $g^{ab} \bmod p$ you actually need to know a or b.

Diffie-Hellman is vulnerable to a **man in the middle attack:**

- She can send $g^e\ mod\ p$ to Bob (by intercepting Alice's $g^a\ mod\ p$)

- She can send $g^e \ mod \ p$ to Alice (by intercepting Bob's $g^b \ mod \ p$)
- She can compute both $k_A = g^{ea} \ mod \ p$ and $k_B = g^{eb} \ mod \ p$, which will be the keys Alice and Bobs will end up computing with the bad information they got from Eve.
    - Note that k_A and k_B will be different, therefore Eve has to extra work.
    - Anytime Alice or Bob send a message, Eve has to intercept it, decrypt it using the right key, re-encrypt it under the other key, and finally let the altered ciphertext pass through to its destination.

The discrete logarithm problem

Motivation: Eve has $x = g^a \ mod \ p$ and $y = g^b \ mod \ p$ but she can't compute $k = g^{ab} \ mod \ p$. Why?

Eve has to solve the **discrete log problem** in order to figure out the value of a or b. Eve can't just take $\log x$ and/or $\log y$ because x and y were obtained by raising g to a certain power modulo p. If modular arithmetic were not used, then a simple $\log$ would have obviously worked.

Discrete log problem: Given g, p and $g^a \ mod \ p$ compute a. This is **computationally hard.**

The Extended Euclidian Algorithm

Algorithm: Given a and b, compute $m = \gcd(a, b)$ and x, y such that $ax + by = m$

Example: $a = 96$ and $b = 38$, $\gcd(a, b) = \gcd(96,38) = 2$

x	y	$c = ax + by$	c_{old}/c_{new} (written as $c_{new} \times \boldsymbol{q} + r = c_{old}$)
1	0	$1 \times 96 + 0 \times 38 = 96$	
0	1	$0 \times 96 + 1 \times 38 = 38$	$\mathbf{2} \times 38 + 20 = 96$
The goal now is to get r to equal 0 and get the corresponding x and y values. We do this by subtracting/adding multiples of the rows in this table. Since $\mathbf{2} \times 38 + 20 = 96$, we subtract $2r_2$ from r_1. We repeat.			
1	-2	$1 \times 96 - 2 \times 38 = 20$	$\mathbf{1} \times 20 + 18 = 38$
-1	3	$-1 \times 96 + 3 \times 38 = 18$	$\mathbf{1} \times 18 + 2 = 20$
2	-5	$2 \times 96 - 5 \times 38 = 2$	$\mathbf{9} \times 2 + 0 = 18$
x	y	$r = 0$	**Done.**

Number theory concepts

- **Definition:** $a \equiv b \ (\mathrm{mod}\ n) \Leftrightarrow n \mid a - b$
- **Theorem:** If $\gcd(a, b) = 1$, then there exists y such that $ay \equiv 1 \ (\mathrm{mod}\ b) \Leftrightarrow y = a^{-1} \ (\mathrm{mod}\ b)$.
    - **Proof:** Run EEA and obtain $ay + bx = 1 \Rightarrow bx = 1 - ay \Rightarrow b \mid 1 - ay \Rightarrow ay \equiv 1 \ (\mathrm{mod}\ b)$
- **Theorem:** If $\gcd(a, n) = 1$ and $\gcd(b, n) = 1$ then $\gcd(ab, n) = 1$
- **Definition:** $\mathbb{Z}_n^* = \{x \in \mathbb{Z}_n | \gcd(x, n) = 1\}$ = set of all numbers that are coprime to n
- **Definition:** The totient function, $\varphi(n) = |\mathbb{Z}_n^*|$ is the size of the $\mathbb{Z}_n^*$ set.
- **Theorem:** If $a \in \mathbb{Z}_n^*$, then $a^{\varphi(n)} \equiv 1 \ (\mathrm{mod}\ n)$.
    - **Implication:** If p is prime, then $a^{p-1} \equiv 1 \ (\mathrm{mod}\ p), \forall a$
- **Computing powers modulo n:** $b^c \equiv (b \bmod n)^{c \ mod \ \varphi(n)} \ (\mathrm{mod}\ n)$
    - You can reduce the base mod n
    - You can reduce the exponent $\bmod \varphi(n)$. Note that when $n =$ prime, $\varphi(p) = p - 1$
    - On top of this you can use **logarithmic exponentiation,** after reducing the base and exponent

- **Computing the totient function $\varphi(n)$**
  - $\varphi(p) = p - 1$, when p is prime
  - $\varphi(p^k) = p^k - p^{k-1} = p^{k-1}(p-1)$, when p is prime
  - $\gcd(a,b) = 1 \Rightarrow \varphi(ab) = \varphi(a)\varphi(b)$
  - If a factorizes as $a = a_1^{r_1} \times a_2^{r_2} \times \ldots \times a_k^{r_k}$ then $\varphi(a) = \varphi(a_1^{r_1}) \times \varphi(a_2^{r_2}) \times \ldots \times \varphi(a_k^{r_k})$

Miller-Rabin primality testing

- uses the fact that if p is prime there are only two **roots of unity** in $\mathbb{Z}_p$: the trivial roots 1 and -1.
- if you can find **non-trivial roots** then p is not prime
- The **Miller-Rabin test**, assume n is odd and not a perfect power of a prime (since this can be easily tested for).
- $n - 1 = 2^s \times d$, where d will be odd
- choose a random base a and check the value of $a^{n-1} \pmod n$, but…
- perform this computation by first determining $a^d \pmod n$, and then repeatedly squaring to get the sequence:
  - $a^d, a^{2d}, a^{2^2 d}, \ldots, a^{2^{s-1} d}, a^{2^s d} = a^{n-1} \pmod n$
- if $a^{n-1} \neq 1 \pmod n \Leftrightarrow \gcd(a,n) \neq 1$, then n is composite
- if $a^{n-1} = 1 \pmod n$, we conduct a follow-up test, looking for a non-primitive root in the sequence of squares
- a is a **Miller-Rabin liar** for n if n is composite and $MR(a,n)$ outputs prime, instead of composite.
  - $\Pr[a = MR\ liar\ for\ n] = \frac{1}{4}$
  - To overcome this, you repeat the test k times, so the $\Pr[accepting\ a\ composite\ as\ a\ prime] \leq 4^{-k}$

RSA encryption

Alice:

- pick large primes p and q, compute $n = pq$
- pick e such that $\gcd(e, \varphi(n)) = 1$
- compute d such that $ed = 1 \pmod{\varphi(n)} \Leftrightarrow d = e^{-1} \pmod{\varphi(n)}$
  - use the Extended Euclidian Algorithm to compute d
  - run $EEA(e, \varphi(n))$, getting $ex + \varphi(n)y = 1 \Rightarrow \varphi(n) \mid ex - 1 \Rightarrow ex = 1 \bmod \varphi(n)$
    - $d = x$
- send (n, e) to Bob as the public key
- keep (n, d) as the private key

Bob:

- has a message $m \in \mathbb{Z}_n^*$
- encrypts m as $c = m^e \bmod n$
- sends c to Alice

Alice:

- decrypts c by computing $c^d = m^{ed} \bmod n$
- $m^{ed} \bmod n = m^{ed \bmod \varphi(n)} \bmod n = m \bmod n$

Why can't the adversary compute d given (n, e)? Because he needs to compute $\varphi(n)$ in order to compute d using EEA, and that is equivalent to factoring n, a hard problem.

RSA attacks

Encrypting short messages using small e

Say $e = 3,$ and $m < N^{\frac{1}{3}}$ is unknown to the attacker. Then $c = m^3 \bmod N = m^3$ and so $m = \sqrt[3]{c}$

More general attack with small e

Let's extend the above attack for any message length of m.

Let $e = 3$. Suppose, three messages are sent to three parties encrypted with public keys $(N_1, 3), (N_2, 3)$ and $(N_3, 3)$

$$c_i = m^3 \bmod N_i$$

We'll assume $\gcd(N_i, N_j) = 1, \forall i, j$ since if that were not the case then you could easily factor one of the N_i's and easily recover m.

Let $N^* = N_1 N_2 N_3$. An extended version of the Chinese Remainder Theorem says there exists a $c < N^*$ such that:

$$c = c_i \bmod N_i, \forall i$$

This c can be computed easily (no clue how) given the public keys and the ciphertexts.

Note that $c = m^3 \bmod N^*$.

Since $m < \min\{N_1, N_2, N_3\}$ we have $m^3 < N^*$. We can now apply the previous attack to get m from c.

$$m = \sqrt[3]{c}$$

Quadratic improvement in recovering m

If $1 \leq m < L$ (when interpreting m as an integer), then we can recover m in $\sqrt{L}$ time.

Attack: assume $m < 2^l$, so $L = 2^l$ and that the attacker knows l. $\alpha = constant, \frac{1}{2} < \alpha < 1$

- **Input:** Public key (N, e), ciphertext c and parameter l
- **Output:** $m < 2^l$ such that $m^e = c \bmod N$

```
T = 2^(αl)
for r = 1 to T:
        x_i = c/r^e mod N

sort the pairs {(r, x_r)}_{r=1}^T by their second component

for s = 1 to T:
        if s^e mod N = x_r for some r
                return rs mod N
```

Time complexity is dominated by the time taken to sort the $2^{\alpha l}$ pairs. Binary search is used to find whether $\exists r, x_r = s^e \bmod N$.

If m is chosen as a random l-bit integer, it can be shown that with good probability $\exists r, s$ with $1 < r, s < 2^{\alpha l}$ and $m = rs$. The algorithm essentially looks for these r and s values.

Common modulus attack I

Company shares keys to each employee i as $pk_i = (N, e_i)$ and $sk_i = (N, d_i)$

$$e_i d_i = 1 \bmod \phi(n), \forall i$$

So each employee has their e_i, d_i pair which means they can easily factor N, which allows them to obtain the decryption key of all the other employees by computing:

$$d_j = e_j \bmod \phi(n)$$

Common modulus attack II

Suppose m is encrypted and sent to two different employees with public keys (N, e_1) and(N, e_2) where $e_1 \neq e_2$. Further assume, without loss of generality, that $\gcd(e_1, e_2) = 1$

Eve sees two ciphertexts:

$c_1 = m^{e_1} \bmod N$ and $c_2 = m^{e_2} \bmod N$

$$\gcd(e_1, e_2) = 1 \Rightarrow \exists x, y, e_1 x + e_2 y = 1$$

Eve computes x and y using EEA. Then…

$$c_1^x c_2^y = (m^{e_1} \bmod N)^x (m^{e_2} \bmod N)^y = (m^{e_1 x} \bmod N)(m^{e_2 y} \bmod N) = m^{e_1 x} m^{e_2 y} \bmod N = m^{e_1 x + e_2 y} \bmod N = m$$

Weak and strong RSA assumptions

Weak RSA assumption: Given x, e, n (composite n) computing y such that $y^e = x \bmod n$ is really hard.

If I give you a composite n and you can find a d such that $3d = 1 \bmod \varphi(n)$, then you know that $\varphi(n) \mid 3d - 1$. It turns out that given this you can compute $\varphi(n)$, and given $\varphi(n)$ you can factor n.

Equivalence: Given e, n compute d (RSA problem), given n compute $\varphi(n)$, and factoring n are all equivalent problems.

Interesting fact: It is not known whether decrypting RSA is equivalent to factoring n. It is assumed decrypting RSA is a little easier actually.

Strong RSA assumption: Given x, n computing any $y, e \neq \pm 1$ s.t. $y^e = x \bmod n$ is really hard. The attacker is given more freedom here: he can actually choose e.

El Gamal encryption

We have **Alice**, **Bob** and global parameters p and g

Alice:

- picks a random a, her secret key
- she sends the public key $g^a \bmod p$ to Bob

Bob:

- To encrypt $m \in \mathbb{Z}_p^*$, Bob picks a random s
- sends $(g^s \bmod p, g^{as} \times m \bmod p)$ to Alice

Alice:

- decrypts by computing: $\frac{g^{as} \times m \bmod p}{(g^s \bmod p)^a} = \frac{g^{as} \times m \bmod p}{g^{as} \bmod p} = m \bmod p$

CCA attack on El Gamal

Eve can always win the CCA game because she can decrypt the received challenge:

Let g and p be the public parameters. Alice has her secret key a and g^a is known by everyone

$$c = (g^s \bmod p, g^{as} \times m \bmod p)$$

Eve computes $c' = (g^s \bmod p, g^{as} \times m \times 2 \bmod p)$

Eve queries the decryption oracle with c' which will gladly decrypt it to $m' = m \times 2 \bmod p$.

Eve can now compute $m' \times 2^{-1} = m \bmod p$ getting m.

One-way functions

Definition: f is a (t, ε)-one-way function if $\forall A$ running in time $\leq t$ we have:

$$Adv\ A = \Pr[A(y) = x \text{ where } f(x) = y] \leq \varepsilon$$

- Modular exponentiation is a **one-way permutation.**

$$f(a) = g^a \bmod p$$

  - With the right, g and p, and for a not so large a, $f(a)$ is injective and invertible, but its inverse is really hard to compute so it's a **one way function**.

Definition: A (t, q, ε)-trap door one way permutation (**TOWP**) is a one way permutation with a trap-door.

- RSA is a **trap door one way permutation**.

RoR security with PKE

Definition: A public key encryption scheme is (t, q, ε) IND-CPA-secure if $\forall A$ running in time $\leq t$ and making $\leq q$ queries has:

$$Adv\ A = \left|\Pr[A^{E_{pk}}(pk) = 1\ ] - \Pr\left[A^{E_{pk}*\$}(pk) = 1\right]\right| \leq \varepsilon$$

Theorem: If (E, D) is $(t, 1, \varepsilon)$-secure then it is $(t, q, (q+1)\varepsilon)$-secure. This depends on the fact that the adversary can perform encryption himself.

The birthday attack on the discrete log problem

Our goal: Given $y = g^x \bmod p$, g and p, compute x.

Randomized algorithm: Baby-step, giant-step algorithm.

1. Pick random $r_1, r_2, \ldots, r_q$ and $s_1, s_2, \ldots, s_q$ relatively prime to p
2. Compute $g^{r_1}, g^{r_2}, \ldots, g^{r_q}$ mod p and $y^{s_1}, \ldots, y^{s_q}$ mod p
3. Suppose that $g^{r_i} = y^{s_j} \pmod p \Rightarrow g^{\frac{r_i}{s_i}} = y \pmod p$
   a. $g^x = g^{x \bmod \varphi(p)} \bmod p$

4. What is $r_i s_j^{-1} \bmod \varphi(p)$?

How big should q be to ensure a collision? $E[\#col] = \frac{q^2}{p}$, so $q = \sqrt{p}$