Cryptography
CS 555

Week 11:

- Formalizing Public Key Crypto
  - Fixes for Plain RSA
- Applications of DDH
- Factoring Algorithms, Discrete Log Attacks + NIST Recommendations for Concrete Security Parameters

Readings: Katz and Lindell Chapter 8.4 & Chapter 9

Spring 2021

Recap CCA-Security $\left(PrivK_{A,\Pi}^{cca}(n)\right)$

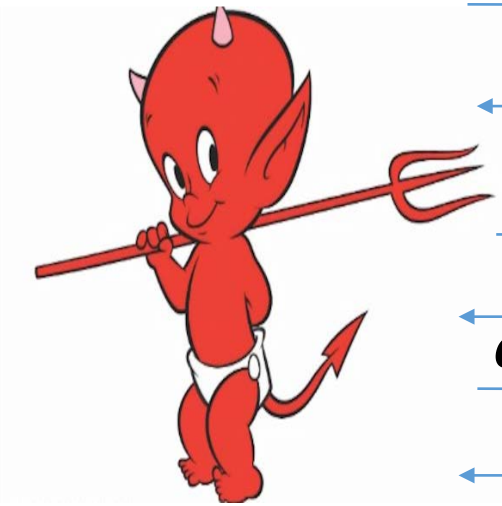
1. Challenger generates a secret key k and a bit b
2. Adversary (A) is given oracle access to Enc_k and Dec_k
3. Adversary outputs m_0,m_1
4. Challenger sends the adversary $c=Enc_k(m_b)$.
5. Adversary maintains oracle access to Enc_k and Dec_k ,however the adversary is not allowed to query $Dec_k(c)$.
6. Eventually, Adversary outputs b'.

$$PrivK_{A,\Pi}^{cca}(n) = 1 \text{ if } \mathrm{b} = \mathrm{b}'; \text{otherwise } 0.$$

CCA-Security: For all PPT A exists a negligible function negl(n) s.t.

$$\Pr\left[PrivK_{A,\Pi}^{cca}(n) = 1\right] \leq \frac{1}{2} + negl(n)$$

CCA-Security ($\mathrm{PubK}_{A,\Pi}^{\mathrm{cca}}(\mathrm{n})$)



Public Key: pk

$\boldsymbol{c_{-1}}$

$\boldsymbol{m_{-1} = \mathrm{Dec}_{sk}(c_{-1})}$

⋮

$m_0\ , m_1$

$\boldsymbol{c_b = \mathrm{Enc}_{pk}(m_b\ )}$

$\boldsymbol{c_k}$

$\boldsymbol{m_k = \mathrm{Dec}_{sk}(c_k)}$

b'

⋮

Random bit b

(pk,sk) = Gen(.)

$$\forall PPT\ A\ \exists \mu\ (\text{negligible})\ \text{s.t}$$

$$\Pr\left[\mathrm{PubK}_{A,\Pi}^{\mathrm{cca}}(\mathrm{n}) = 1\right] \leq \frac{1}{2} + \mu(n)$$

Encrypting Longer Messages

Claim 11.7: Let $\Pi = (Gen, Enc, Dec)$ denote a CPA-Secure public key encryption scheme and let $\Pi' = (Gen, Enc', Dec')$ be defined such that

$$\mathbf{Enc}'_{\mathbf{pk}}(\boldsymbol{m_1} \parallel \boldsymbol{m_2} \parallel \cdots \parallel \boldsymbol{m_\ell}) = \mathbf{Enc}_{\mathbf{pk}}(\boldsymbol{m_1}) \parallel \cdots \parallel \mathbf{Enc}_{\mathbf{pk}}(\boldsymbol{m_\ell})$$

Then Π' is also CPA-Secure.

Claim? Let $\Pi = (Gen, Enc, Dec)$ denote a CCA-Secure public key encryption scheme and let $\Pi' = (Gen, Enc', Dec')$ be defined such that

$$\mathbf{Enc}'_{\mathbf{pk}}(\boldsymbol{m_1} \parallel \boldsymbol{m_2} \parallel \cdots \parallel \boldsymbol{m_\ell}) = \mathbf{Enc}_{\mathbf{pk}}(\boldsymbol{m_1}) \parallel \cdots \parallel \mathbf{Enc}_{\mathbf{pk}}(\boldsymbol{m_\ell})$$

Then Π' is also CCA-Secure.

Is this second claim true?

Encrypting Longer Messages

Claim? Let $\Pi = (Gen, Enc, Dec)$ denote a CCA-Secure public key encryption scheme and let $\Pi' = (Gen, Enc', Dec')$ be defined such that

$$\mathbf{Enc}'_{\mathbf{pk}}(\boldsymbol{m}_1 \parallel \boldsymbol{m}_2 \parallel \cdots \parallel \boldsymbol{m}_\ell) = \mathbf{Enc}_{\mathbf{pk}}(\boldsymbol{m}_1) \parallel \cdots \parallel \mathbf{Enc}_{\mathbf{pk}}(\boldsymbol{m}_\ell)$$

Then Π' is also CCA-Secure.

Is this second claim true?

Answer: No!

Encrypting Longer Messages

Fact: Let $\Pi = (Gen, Enc, Dec)$ denote a CCA-Secure public key encryption scheme and let $\Pi' = (Gen, Enc', Dec')$ be defined such that

$$\mathbf{Enc}'_{\mathbf{pk}}(\boldsymbol{m_1} \parallel \boldsymbol{m_2} \parallel \cdots \parallel \boldsymbol{m_\ell}) = \mathbf{Enc}_{\mathbf{pk}}(\boldsymbol{m_1}) \parallel \cdots \parallel \mathbf{Enc}_{\mathbf{pk}}(\boldsymbol{m_\ell})$$

Then Π' is **Provably Not** CCA-Secure.

1. Attacker sets $\boldsymbol{m_0} = \mathbf{0}^n \parallel \mathbf{1}^n \parallel \mathbf{1}^n$ and $\boldsymbol{m_1} = \mathbf{0}^n \parallel \mathbf{0}^n \parallel \mathbf{1}^n$ and gets $\boldsymbol{c_b} = \mathbf{Enc}'_{\mathbf{pk}}(\boldsymbol{m_b}) = \boldsymbol{c_{b,1}} \parallel \boldsymbol{c_{b,2}} \parallel \boldsymbol{c_{b,3}}$
2. Attacker sets $\boldsymbol{c'} = \boldsymbol{c_{b,2}} \parallel \boldsymbol{c_{b,3}} \parallel \boldsymbol{c_{b,1}}$, queries the decryption oracle and gets

$$\mathbf{Dec}'_{\mathbf{sk}}(\boldsymbol{c'}) = \begin{cases} \mathbf{1}^n \parallel \mathbf{1}^n \parallel \mathbf{0}^n & \textbf{if b=0} \\ \mathbf{0}^n \parallel \mathbf{1}^n \parallel \mathbf{0}^n & \boldsymbol{otherwise} \end{cases}$$

Achieving CPA and CCA-Security

- Plain RSA is not CPA Secure (therefore, not CCA-Secure)

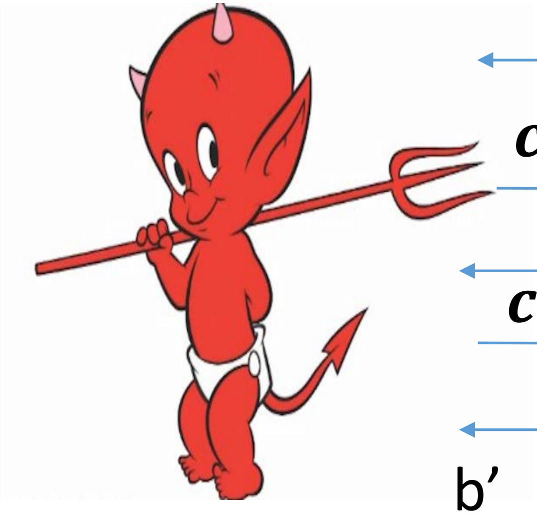
- El-Gamal (future) is CPA-Secure, but not CCA-Secure

- Tools to obtain CCA-Security in Public Key Setting
  - Key Encapsulation Mechanism
  - RSA-OAEP (proof in random oracle model)
  - Cramer-Shoup (first provably secure construction using standard assumptions (DDH))

Key Encapsulation Mechanism (KEM)

- Three Algorithms
  - $\text{Gen}(1^n; R)$ (Key-generation algorithm)
    - Input: Random Bits R
    - Output: $(\boldsymbol{pk}, \boldsymbol{sk}) \in \boldsymbol{\mathcal{K}}$
  - $\text{Encaps}_{\text{pk}}(1^n; R)$
    - Input: public key $\boldsymbol{pk}$, security parameter 1^n, random bits R
    - Output: Symmetric key $\text{k} \in \{0,1\}^{\ell(n)}$ and a ciphertext c
  - $\text{Decaps}_{\text{sk}}(c)$ (Deterministic algorithm)
    - Input: Secret key $\text{sk} \in \mathcal{K}$ and a ciphertext c
    - Output: a symmetric key $\text{k} \in \{0,1\}^{\ell(n)}$ or ⊥ (fail)
- **Invariant**: $\text{Decaps}_{\text{sk}}$(c)=k whenever (c,k) = $\text{Encaps}_{\text{pk}}(1^{n'}; R)$

KEM CCA-Security ($\mathrm{KEM}^{\mathrm{cca}}_{\mathrm{A},\Pi}(\mathrm{n})$)



$(\boldsymbol{pk}, \boldsymbol{c}, \boldsymbol{k_b})$

$\boldsymbol{c_1} \neq \boldsymbol{c}$

$\mathbf{Decaps}_{\boldsymbol{sk}}(\boldsymbol{c_1})$

$\boldsymbol{c_2} \neq \boldsymbol{c}$

$\mathbf{Decaps}_{\boldsymbol{sk}}(\boldsymbol{c_2})$

b'

⋮

Random bit b

(pk,sk) = Gen(.)

$(\boldsymbol{c}, \boldsymbol{k_0}) = \mathbf{Encaps}_{\boldsymbol{pk}}(.)$

$\boldsymbol{k_1} \leftarrow \{\mathbf{0}, \mathbf{1}\}^{\boldsymbol{n}}$

$$\forall PPT\ A\ \exists \mu\ (\mathrm{negligible})\ \mathrm{s.t}$$

$$\Pr\left[\mathrm{KEM}^{\mathrm{cca}}_{\mathrm{A},\Pi} = 1\right] \leq \frac{1}{2} + \mu(n)$$

CCA-Secure Encryption from CCA-Secure KEM

$$\mathbf{Enc_{pk}}(\boldsymbol{m}; \boldsymbol{R_1}, \boldsymbol{R_2}) = \langle \boldsymbol{c}, \mathbf{Enc_k^*}(\boldsymbol{m}; \boldsymbol{R_2}) \rangle$$

Where

- $(\boldsymbol{c}, \boldsymbol{k}) = \mathbf{Encaps_{pk}}(\mathbf{1}^{\boldsymbol{n}}; \boldsymbol{R_1})$,
- $\mathbf{Enc_k^*}$ is a CCA-Secure symmetric key encryption algorithm, and
- $\mathbf{Encaps_{pk}}$ is a CCA-Secure KEM.

Theorem 11.14: $\mathbf{Enc_{pk}}$ is CCA-Secure public key encryption scheme.

CCA-Secure Encryption from CCA-Secure KEM

$$\mathbf{Enc_{pk}}(\boldsymbol{m}; \boldsymbol{R_1}, \boldsymbol{R_2}) = \langle \boldsymbol{c}, \mathbf{Enc^*_k}(\boldsymbol{m}; \boldsymbol{R_2}) \rangle$$
$$\mathbf{Dec_{pk}}((\boldsymbol{c}, \boldsymbol{c'})) = \mathbf{Dec^*_k}(\boldsymbol{c'})$$

where

$$(\boldsymbol{c}, \boldsymbol{k}) = \mathbf{Encaps_{pk}}(\mathbf{1}^{\boldsymbol{n}}; \boldsymbol{R_1}) \quad \textbf{\textit{and}} \quad \boldsymbol{k} = \mathbf{Decaps_{sk}}(\boldsymbol{c})$$

Theorem 11.14: $\mathbf{Enc_{pk}}$ is CCA-Secure public key encryption scheme.

CCA-Secure Encryption from CCA-Secure KEM

$\mathbf{Enc_{pk}}(\boldsymbol{m}; \boldsymbol{R}) = \langle \boldsymbol{c}, \mathbf{Enc_k^*}(\boldsymbol{m}) \rangle$ where $(\boldsymbol{c}, \boldsymbol{k}) = \mathbf{Encaps_{pk}}(\mathbf{1}^{\boldsymbol{n}}; \boldsymbol{R})$,

- $\mathbf{Enc_k^*}$ is a CCA-Secure symmetric key encryption algorithm, and

Theorem 11.14: $\mathbf{Enc_{pk}}$ is CCA-Secure public key encryption scheme.

Proof: Assume for contradiction that PPT attacker **A** wins the CCA-Security Game against $\mathbf{Enc_k}$ with non-negligible probability $\frac{1}{2} + f(n)$. Design an attacker **B** that break CCA-Security of KEM $\mathbf{Encaps_{pk}}$

1. **B** receives public key pk from KEM challenger, along with challenge $(\boldsymbol{c}, \boldsymbol{k_b})$ and forwards public key pk it to **A**
2. **B** flips a coin b' and simulates CCA attacker **A**
3. Whenever **A** submits the challenge pair of messages $(\boldsymbol{m_0}, \boldsymbol{m_1})$ **B** responds with (c, $\mathbf{Enc}^*_{\boldsymbol{k_b}}(\boldsymbol{m_{b'}})$)
4. Whenever **A** queries for $\mathbf{Dec_{sk}}(\boldsymbol{c'}, \boldsymbol{t'})$ attacker **B** forwards $\boldsymbol{c'}$ to KEM challenger to get $\mathbf{k}' = \mathbf{Decaps_{sk}}(\boldsymbol{c})$ and sends $\mathbf{Dec}^*_{\mathbf{k}'}(\boldsymbol{t'})$ to attacker.
5. Whenever A outputs a guess b'' B outputs 1 if and only if b''=b'.

CCA-Secure Encryption from CCA-Secure KEM

Theorem 11.14: $\mathbf{Enc_{pk}}$ is CCA-Secure public key encryption scheme.

Proof: Assume for contradiction that PPT attacker **A** wins the CCA-Security Game against $\mathbf{Enc_k}$ with non-negligible probability $\frac{1}{2} + f(n)$. Design an attacker **B** that break CCA-Security of KEM $\mathbf{Encaps_{pk}}$

1. **B** receives public key pk from KEM challenger, along with challenge $(\boldsymbol{c}, \boldsymbol{k_b})$ and forwards public key pk it to **A**

2. **B** flips a coin b' and simulates CCA attacker **A**

3. Whenever **A** submits the challenge pair of messages $(\boldsymbol{m_0}, \boldsymbol{m_1})$ **B** simply responds with $(\mathrm{c}, \mathbf{Enc}^*_{\boldsymbol{k_b}}(\boldsymbol{m_{b'}}))$

4. Whenever **A** queries for $\mathbf{Dec_{sk}}(\boldsymbol{c'}, \boldsymbol{t'})$ attacker **B** forwards $\boldsymbol{c'}$ to KEM challenger to get $\mathbf{k}' = \mathbf{Decaps_{sk}}(\boldsymbol{c})$ and sends $\mathbf{Dec}^*_{\mathbf{k}'}(\boldsymbol{t'})$ to attacker.

5. Whenever **A** outputs a guess b'' **B** outputs 0 if and only if b''=b'.

Analysis: If b=0 then $\Pr[\mathrm{b}'' = \mathrm{b}'] = \frac{1}{2} + f(n)$ as this is just the regular CCA-Security game

If b=1 then $\Pr[\mathrm{b}'' = \mathrm{b}'] \geq \frac{1}{2} - \mu(n)$ for some negligible function $\mu(n)$

(Follows by CCA-Security of $\mathbf{Enc}^*_{\boldsymbol{k_1}}$ since $\boldsymbol{k_1}$ is random and is unrelated to c)

B outputs correct guess with non-negligible probability at least

$$\Pr[b = 1]\left(\frac{1}{2} + f(n)\right) + \Pr[b = 0]\left(\frac{1}{2} - \mu(n)\right) = \frac{1}{2} + \frac{f(n) - \mu(n)}{2}$$

Recap RSA-Assumption

RSA-Experiment: RSA-INV$_{A,n}$

1. **Run KeyGeneration**(1^n) **to obtain (N,e,d)**
2. **Pick uniform** $\mathrm{y} \in \mathbb{Z}_{\mathrm{N}}^{*}$
3. Attacker A is given N, e, y and outputs $\mathrm{x} \in \mathbb{Z}_{\mathrm{N}}^{*}$
4. Attacker wins (RSA−INV$_{A,n}$=1) if $x^e = \mathrm{y} \bmod \mathrm{N}$

$$\forall PPT\ A\ \exists \mu\ (\text{negligible})\ \text{s.t}\ \Pr\left[\text{RSA−INV}_{A,n} = 1\right] \leq \mu(n)$$

CCA-Secure KEM in the Random Oracle Model

- Let (N,e,d) be an RSA key (pk =(N,e), sk=(N,d)).

$$\mathrm{Encaps}_{\mathrm{pk}}(1^n, R) = \left(r^e \, mod \, N, k = H(r)\right)$$
$$\mathrm{Decaps}_{\mathrm{sk}}(c) = H(r) \quad \text{where} \quad r = c^d \, mod \, N$$

- Remark 1: k is completely random string unless the adversary can query random oracle H on input r.
- Remark 2: If RSA-Inversion assumption holds (Plain-RSA is hard to invert for a random input) then any PPT attacker finds queries H(r) with negligible probability.

Using a CCA-Secure KEM

- Let (N,e,d) be an RSA key (pk =(N,e), sk=(N,d)).

$$\mathrm{Enc}_{\mathrm{pk}}(m;R) = (r^e \ mod\ N, \mathrm{AEnc}_{\mathrm{k}}(m)) \ where\ k = H(r)$$
$$\mathrm{Dec}_{\mathrm{sk}}(c,t) = (c^d \ mod\ N, \mathrm{ADec}_{\mathrm{k}}(t)) \ where\ k = H\left(c^d \ mod\ N\right)$$

- Remark 1: k is completely random string unless the adversary can query random oracle H on input r.
- Remark 2: If RSA-Inversion assumption holds (Plain-RSA is hard to invert for a random input) then any PPT attacker finds queries H(r) with negligible probability.

Using a CCA-Secure KEM

- Let (N,e,d) be an RSA key (pk =(N,e), sk=(N,d)).

$$\mathrm{Enc}_{\mathrm{pk}}(m;R) = (r^e \; mod \; N, \mathrm{AEnc}_{\mathrm{k}}(m)) \; where \; k = H(r)$$

$$\mathrm{Dec}_{\mathrm{sk}}(c,t) = (c^d \; mod \; N, \mathrm{ADec}_{\mathrm{k}}(t)) \; where \; k = H\left(c^d \; mod \; N\right)$$

Theorem: If RSA-Inversion assumption holds and H is a random oracle then encryption scheme above is CCA-Secure.

RSA-OAEP
(Optimal Asymmetric Encryption Padding)

- $\mathbf{Enc}_{\boldsymbol{pk}}\,(m; r) = [(x \parallel y)^e \;\; mod\; N]$
- Where $x \parallel y \;\leftarrow \mathrm{OAEP}(m \parallel 0^{k_1} \parallel r)$
- $\mathbf{Dec}_{\boldsymbol{sk}}\,(c) =$

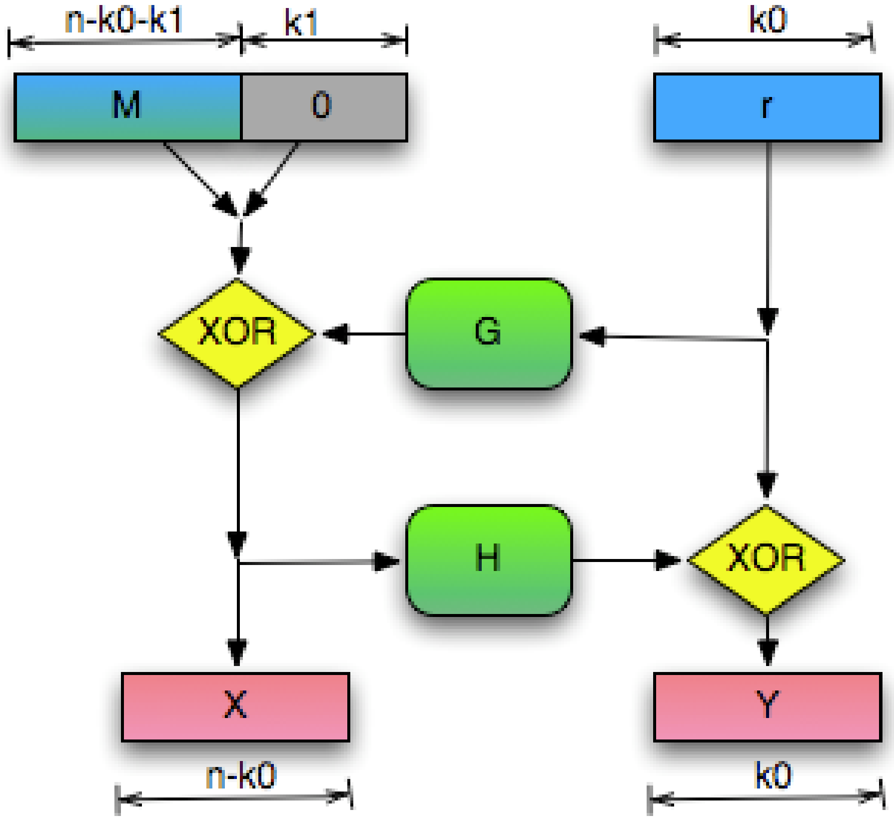
  $\tilde{m} \leftarrow [(c)^d\, mod\; N]$

  If $\|\tilde{m}\| > n$ **return** fail

  $m \parallel z \parallel r \leftarrow \mathrm{OAEP}^{-1}(\tilde{m})$

  If $z \neq 0^{k_1}$ then **return** fail

  return m

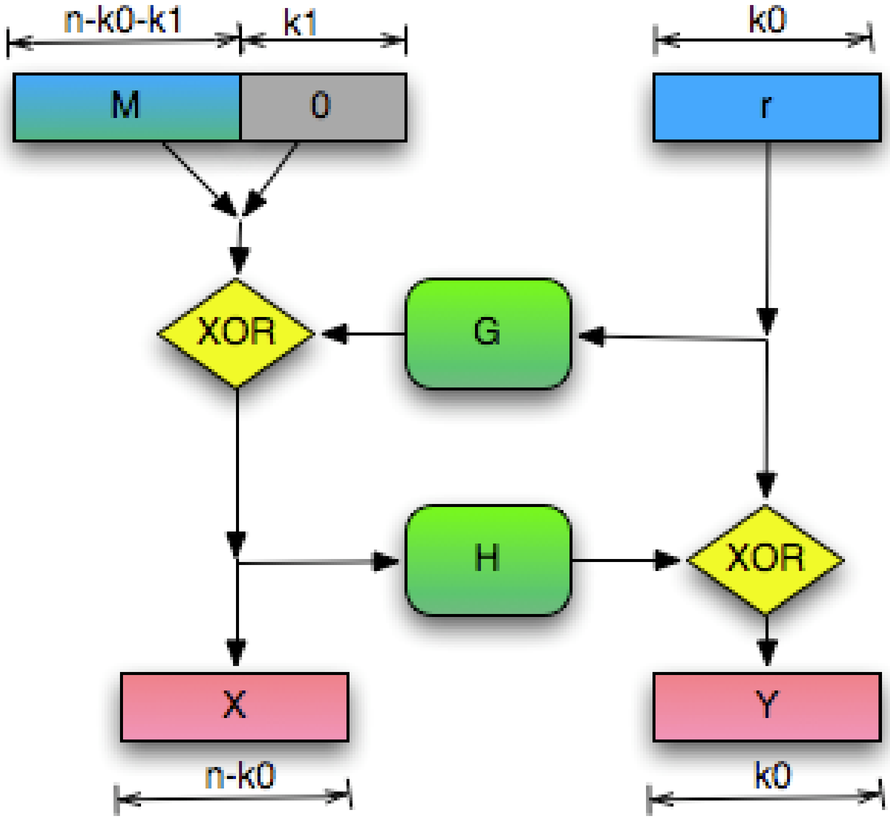


$\mathbf{OAEP}(\boldsymbol{m} \parallel \mathbf{0}^{\boldsymbol{k}_1} \parallel \boldsymbol{r})$

RSA-OAEP
(Optimal Asymmetric Encryption Padding)

Theorem: If we model G and H as Random oracles then RSA-OAEP is a CCA-Secure public key encryption scheme (given RSA-Inversion assumption).

Bonus: One of the fastest in practice!



PKCS #1 v2.0

- Implementation of RSA-OAEP

- James Manger found a chosen-ciphertext attack.

- What gives?

PKCS #1 v2.0 (Bad Implementation)

- $\mathbf{Enc}_{pk}\,(m; r) = [(x \parallel y)^e \;\; mod\; N]$
- Where $x \parallel y \;\leftarrow \mathrm{OAEP}(m \parallel 0^{k_1} \parallel r)$
- $\mathbf{Dec}_{sk}\,(c) =$

  $\widetilde{m} \leftarrow [(c)^d\, mod\; N]$

  If $\|\widetilde{\boldsymbol{m}}\| > \boldsymbol{n}$ return Error Message 1

  $m \parallel z \parallel r \leftarrow \mathrm{OAEP}^{-1}(\widetilde{m})$

  If $\boldsymbol{z} \neq \mathbf{0}^{\boldsymbol{k}_1}$ then output Error Message 2

  return m

PKCS #1 v2.0 (Attack)

- Manger's CCA-Attack recovers secret message
  - **Step 1:** Use decryption oracle to check if $2\widetilde{m} \geq 2^n$ *(i.e., if we get error message 1*
  - $c = [(\widetilde{m})^e \, mod \, N]$ →$2^e c = [(2\widetilde{m})^e \, mod \, N]$
  - If we get error message 1 when decrypting $2^e c$ then $2\widetilde{m} \geq 2^n$
- Generalization ($x > 2$): can check if $x\widetilde{m} \geq 2^n$ by submitting query $x^e c$ to decryption oracle
- Can extract $\widetilde{m}$ using $\mathrm{O}(\|\boldsymbol{N}\|)$ queries to decryption oracle
- Run $m \parallel z \parallel r \leftarrow \mathrm{OAEP}^{-1}(\widetilde{m})$ to recover message
- Attack also works as a side channel attack
  - Even if error messages are the same the time to respond could be different in each case.
- **Fixes:** Implementation should return same error message and should make sure that the time to return each error is the same in all cases.

Week 11: Topic 1: Discrete Logarithm Applications

Diffie-Hellman Key Exchange

Collision Resistant Hash Functions

Password Authenticated Key Exchange

Diffie-Hellman Key Exchange

1. Alice picks x_A and sends $h_A := g^{x_A}$ to Bob
2. Bob picks x_B and sends $h_B := g^{x_B}$ to Alice
3. Alice and Bob can both compute $K_{A,B} = g^{x_B\, x_A}$

Alice Computes: $(h_B)^{x_A} = (g^{x_B})^{x_A} = g^{x_B\, x_A} = K_{A,B}$

Bob Computes: $(h_A)^{x_B} = (g^{x_A})^{x_B} = g^{x_A\, x_B} = K_{A,B}$

Key-Exchange Experiment $KE_{A,\Pi}^{eav}(n)$:

- Two parties run Π to exchange secret messages (with security parameter 1^n).
- Let **trans** be a transcript which contains all messages sent and let k be the secret key output by each party.
- Let b be a random bit and let $\mathbf{k_b}$ = k if b=0; otherwise $\mathbf{k_b}$ is sampled uniformly at random.
- Attacker A is given **trans** and $\mathbf{k_b}$ (passive attacker).
- Attacker outputs b' ($KE_{A,\Pi}^{eav}(n)$=1 if and only if b=b')

Security of Π against an eavesdropping attacker: For all PPT A there is a negligible function **negl** such that

$$\Pr\left[KE_{A,\Pi}^{eav}(n)\right] \leq ½ + \mathbf{negl}(\mathrm{n}).$$

Diffie-Hellman Key-Exchange is Secure

Theorem: If the decisional Diffie-Hellman problem is hard relative to group generator $\mathcal{G}$ then the Diffie-Hellman key-exchange protocol Π is secure in the presence of a (passive) eavesdropper (*).

(*) Assuming keys are chosen uniformly at random from the cyclic group $\mathbb{G}$

Protocol Π

1. Alice picks x_A and sends g^{x_A} to Bob
2. Bob picks x_B and sends g^{x_B} to Alice
3. Alice and Bob can both compute $K_{A,B} = g^{x_B x_A}$

Diffie-Hellman Assumptions

Computational Diffie-Hellman Problem (CDH)

- Attacker is given $\mathrm{h}_1 = g^{x_1} \in \mathbb{G}$ and $\mathrm{h}_2 = g^{x_2} \in \mathbb{G}$.
- Attackers goal is to find $g^{x_1 x_2} = (\mathrm{h}_1)^{x_2} = (\mathrm{h}_2)^{x_1}$
- **CDH Assumption**: For all PPT A there is a negligible function negl upper bounding the probability that A succeeds

Decisional Diffie-Hellman Problem (DDH)

- Let $\mathrm{z}_0 = g^{x_1 x_2}$ and let $\mathrm{z}_1 = g^{r}$, where x_1,x_2 and r are random
- Attacker is given g^{x_1}, g^{x_2} and z_b (for a random bit b)
- Attackers goal is to guess b
- **DDH Assumption**: For all PPT A there is a negligible function negl such that A succeeds with probability at most ½ + negl(n).

Diffie-Hellman Key Exchange

1. Alice picks x_A and sends g^{x_A} to Bob
2. Bob picks x_B and sends g^{x_B} to Alice
3. Alice and Bob can both compute $K_{A,B} = g^{x_B x_A}$

Intuition: Decisional Diffie-Hellman assumption implies that a passive attacker who observes g^{x_A} and g^{x_B} still cannot distinguish between $K_{A,B} = g^{x_B x_A}$ and a random group element**.**

Remark: Modified protocol sets $K_{A,B} = H(g^{x_B x_A})$ which is provably secure under the weaker CDH assumption assuming that H is a random oracle.

Diffie-Hellman Key-Exchange is Secure

Theorem: If the decisional Diffie-Hellman problem is hard relative to group generator $\mathcal{G}$ then the Diffie-Hellman key-exchange protocol Π is secure in the presence of an eavesdropper (*).

Proof: Diffie-Hellman transcript: (g^x, g^y)

$$\Pr\left[KE_{A,\Pi}^{eav}(n) = 1\right]$$
$$=½\Pr\left[KE_{A,\Pi}^{eav}(n) = 1|b = 1\right] + ½\Pr\left[KE_{A,\Pi}^{eav}(n) = 1|b = 0\right]$$
$$=½\Pr[A(\mathbb{G}, g, q, g^x, g^y, g^{xy}) = 1] + ½\Pr[A(\mathbb{G}, g, q, g^x, g^y, g^z) = 0]$$
$$=½+½(\Pr[A(\mathbb{G}, g, q, g^x, g^y, g^{xy}) = 1] - \Pr[A(\mathbb{G}, g, q, g^x, g^y, g^z) = 1]).$$
$$\leq ½ + ½\text{negl}(n) \text{ (by DDH)}$$

(*) Assuming keys are chosen uniformly at random from the cyclic group $\mathbb{G}$

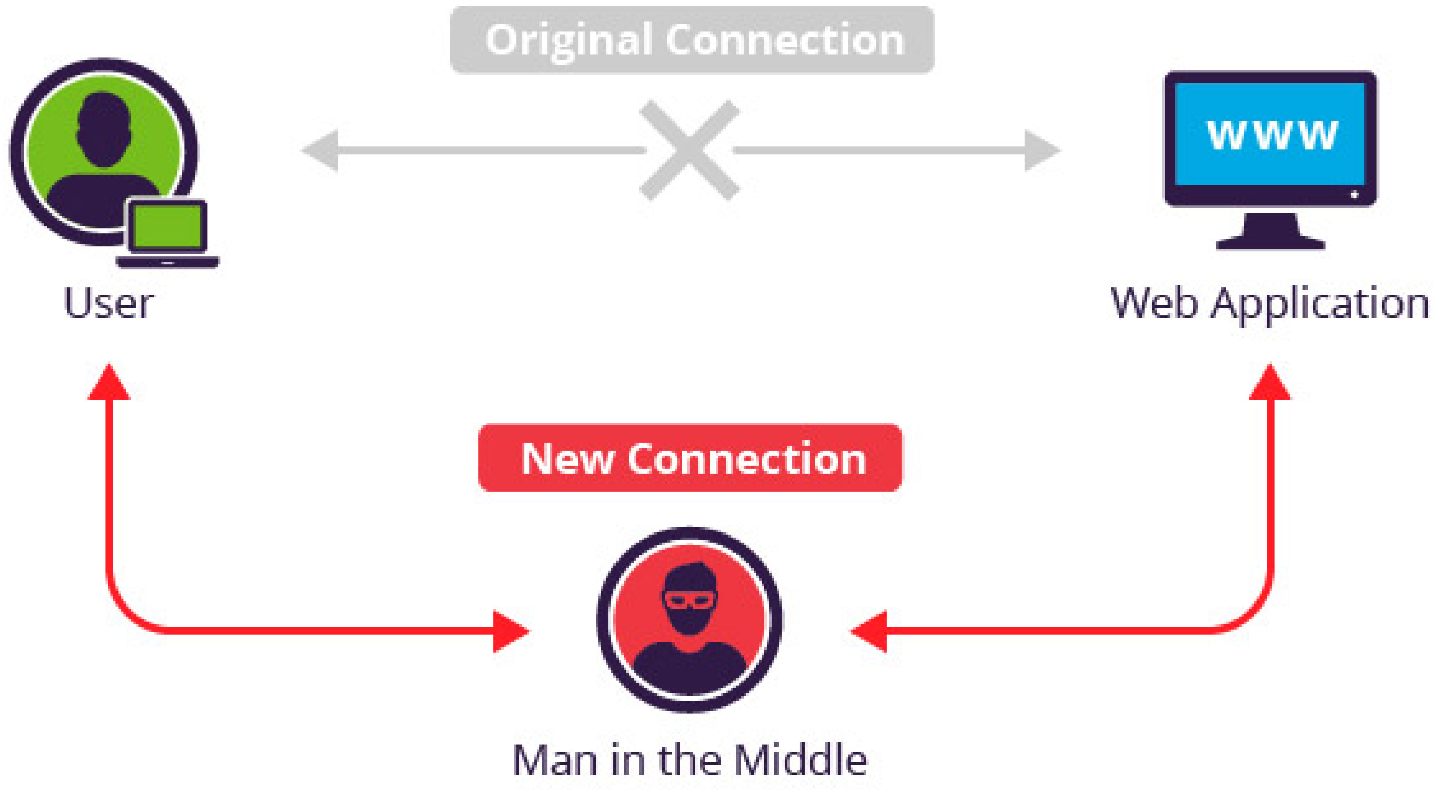
Diffie-Hellman Key Exchange

1. Alice picks x_A and sends g^{x_A} to Bob
2. Bob picks x_B and sends g^{x_B} to Alice
3. Alice and Bob can both compute $K_{A,B} = g^{x_B x_A}$

Intuition: Decisional Diffie-Hellman assumption implies that a passive attacker who observes g^{x_A} and g^{x_B} still cannot distinguish between $K_{A,B} = g^{x_B x_A}$ and a random group element**.**

Remark: The protocol is vulnerable against active attackers who can tamper with messages.

Man in the Middle Attack (MITM)

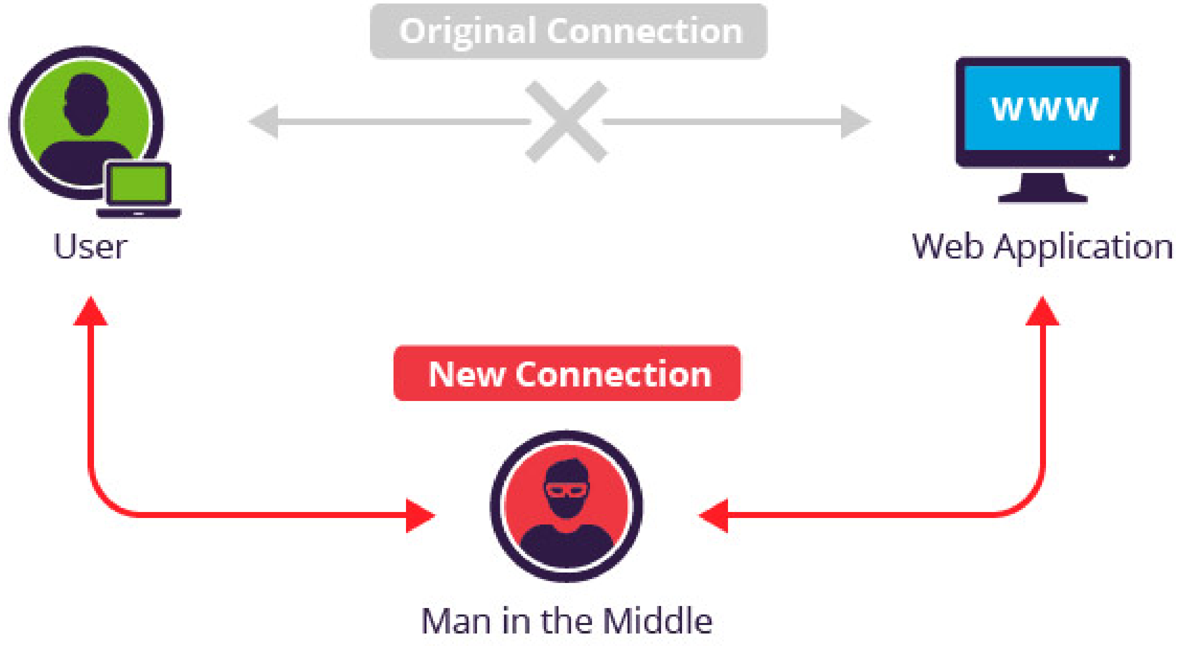


Man in the Middle Attack (MITM)

1. Alice picks x_A and sends g^{x_A} to Bob
   - Mallory intercepts g^{x_A}, picks x_E and sends g^{x_E} to Bob instead
2. Bob picks x_B and sends g^{x_B} to Alice
   1. Mallory intercepts g^{x_B}, picks $x_{E'}$ and sends $g^{x_{E'}}$ to Alice instead
3. Eve computes $g^{x_{E'} x_A}$ and $g^{x_E x_B}$
   1. Alice computes secret key $g^{x_{E'} x_A}$ (shared with Eve not Bob)
   2. Bob computes $g^{x_E x_B}$ (shared with Eve not Alice)
4. Mallory forwards messages between Alice and Bob (tampering with the messages if desired)
5. Neither Alice nor Bob can detect the attack

Man in the Middle Attack (MITM)

Defense: If Alice and Bob already know g^{x_B} and g^{x_A} (respectively) then MITM attackdoes not work.



Certificate Authorities (CA):
Users/Companies can register & lookup public keys e.g., Alice asks CA to send Bob's public key.

Corrupt/Breached CA: does not learn secret keys x_A and x_B

Corrupt CA could send Alice (resp. Bob) the wrong key for Bob

Discrete Log Experiment $\mathrm{DLog}_{A,G}(n)$

1. Run $\mathcal{G}(1^n)$ to obtain a cyclic group $\mathbb{G}$ of order q (with $\|q\| = n$) and a generator g such that $< \mathrm{g} >= \mathbb{G}$.
2. Select $\mathrm{h} \in \mathbb{G}$ uniformly at random.
3. Attacker A is given $\mathbb{G}$, q, g, h and outputs an integer x.
4. Attacker wins ($\mathrm{DLog}_{A,G}(n)=1$) if and only if $g^x=h$.

We say that the discrete log problem is hard relative to generator $\mathcal{G}$ if

$$\forall PPT\ A\ \exists \mu\ (\text{negligible})\ \text{s.t}\ \Pr[\mathrm{DLog}_{A,n} = 1] \le \mu(n)$$

Collision Resistant Hash Functions (CRHFs)

- Recall: not known how to build CRHFs from OWFs
- Can build collision resistant hash functions from Discrete Logarithm Assumption
- Let $\mathcal{G}(1^n)$ output $(\mathbb{G}, q, g)$ where $\mathbb{G}$ is a cyclic group of order q and g is a generator of the group.
- Suppose that discrete log problem is hard relative to generator $\mathcal{G}$.

$$\forall PPT\ A\ \exists \mu \text{ (negligible) s.t } \Pr[\mathsf{DLog}_{\mathsf{A,n}} = 1] \leq \mu(n)$$

Collision Resistant Hash Functions

- Let $\mathcal{G}(1^n)$ output $(\mathbb{G}, q, g)$ where $\mathbb{G}$ is a cyclic group of prime order q and g is a generator of the group.

Collision Resistant Hash Function (Gen,H):

- $Gen(1^n)$
  1. $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^n)$
  2. Select random $\mathrm{h} \leftarrow \mathbb{G}$
  3. Output public seed $\mathrm{s} = (\mathbb{G}, q, g, h)$
- $H^s(x_1, x_2) = g^{x_1} h^{x_2}$ (where, $x_1, x_2 \in \mathbb{Z}_q$)

Claim: (Gen,H) is collision resistant if the discrete log assumption holds for $\mathcal{G}$

Collision Resistant Hash Functions

- $H^s(x_1, x_2) = g^{x_1} h^{x_2}$ (where, $x_1, x_2 \in \mathbb{Z}_q$)

Claim: (Gen,H) is collision resistant

Proof (sketch): Suppose we find a collision $H^s(x_1, x_2) = H^s(y_1, y_2)$ then we have $g^{x_1} h^{x_2} = g^{y_1} h^{y_2}$ which implies

$$h^{x_2 - y_2} = g^{y_1 - x_1}$$

Use extended GCD to find $(x_2 - y_2)^{-1}$ mod q then

$$h = h^{(x_2 - y_2)(x_2 - y_2)^{-1}} = g^{(y_1 - x_1)(x_2 - y_2)^{-1}}$$

Which means that $(y_1 - x_1)(x_2 - y_2)^{-1} \, mod \, q$ is the discrete log of h.

Collision Resistant Hash Functions

•

Cl

What if $x_2 = y_2$ *so that inverse* $(x_2 - y_2)^{-1}$ *does not exist?*
Claim: This cannot happen.
Proof: If $(x_2 - y_2)$ then $h^{x_2-y_2} = h^0$ is the identity ➔ $g^{y_1-x_1}$ is the identity ➔ $y_1 = x_1$ ➔ (x_1, x_2)=(y_1, y_2) (Contradiction)

Proof (sketch): Suppose we find a collision $H^s(x_1, x_2) = H^s(y_1, y_2)$ then we have $g^{x_1}h^{x_2} = g^{y_1}h^{y_2}$ which implies

$$h^{x_2-y_2} = g^{y_1-x_1}$$

Use extended GCD to find $(x_2 - y_2)^{-1}$ mod q then

$$h = h^{(x_2-y_2)(x_2-y_2)^{-1}} = g^{(y_1-x_1)(x_2-y_2)^{-1}}$$

Which means that $(y_1 - x_1)(x_2 - y_2)^{-1}$ *mod q* is the discrete log of h.

Week 11: Topic 2: Factoring Algorithms, Discrete Log Attacks + NIST Recommendations for Concrete Security Parameters

Pollard’s p-1 Algorithm (Factoring)

- Let $N = pq$ where (p-1) has only “small” prime factors.
- Pollard’s p-1 algorithm can factor N.
  - **Remark 1**: This happens with very small probability if p is a random n bit prime.
  - **Remark 2**: One convenient/fast way to generate big primes it to multiply many small primes, add 1 and test for primality.
    - Example: $2 \times 3 \times 5 \times 7 + 1 = 211$ is prime

Claim: Suppose we are given an integer B such that (p-1) divides B but (q-1) does not divide B then we can factor N.

Pollard's p-1 Algorithm (Factoring)

Claim: Suppose we are given an integer B such that (p-1) divides B but (q-1) does not divide B then we can factor N.

Proof: Suppose B=c(p-1) for some integer c and let

$$y = [x^B - 1 \; mod \; N]$$

Applying the Chinese Remainder Theorem we have

$$y \leftrightarrow (x^B - 1 \bmod \mathrm{p}, x^B - 1 \bmod \mathrm{q})$$
$$= \left(0, x^{B \; mod \; (q-1)} - 1 \bmod \mathrm{q}\right)$$

This means that p divides y, but q does not divide y (unless $x^B = 1 \bmod \mathrm{q}$, which is unlikely when x is random since $0 \neq B \; mod \; (q-1)$).

Thus, GCD(y,N) = p

Pollard's p-1 Algorithm (Factoring)

- Let $N = pq$ where (p-1) has only "small" prime factors.
- Pollard's p-1 algorithm can factor N.

Claim: Suppose we are given an integer B such that (p-1) divides B but (q-1) does not divide B then we can factor N.

- **Goal**: Find B such that (p-1) divides B but (q-1) does not divide B.
- **Remark**: This is difficult if (p-1) has a large prime factor.

$$B = \prod_{i=1}^{k} p_i^{[n/\log p_i]}$$

Pollard's p-1 Algorithm (Factoring)

- **Goal**: Find B such that (p-1) divides B but (q-1) does not divide B.
- **Remark**: This is difficult if (p-1) has a large prime factor.

$$B = \prod_{i=1}^{k} p_i^{[n/\log p_i]}$$

Here p_1=2,p_2=3,…p_k are the first k prime numbers.

Fact: If (q-1) has prime factor larger than p_k then (q-1) does not divide B.

Fact: If (p-1) does not have prime factor larger than p_k then (p-1) does divide B.

Pollard’s p-1 Algorithm (Factoring)

- **Option 1:** To defeat this attack we can choose strong primes p and q
  - A prime p is strong if (p-1) has a large prime factor
- **Drawback:** It takes more time to generate (provably) strong primes

- **Option 2:** A random prime is strong with high probability

- **Current Consensus:** Just pick a random prime

Pollard's Rho Algorithm

- General Purpose Factoring Algorithm
  - Doesn't assume (p-1) has no large prime factor
  - **Goal**: factor N=pq (product of two n-bit primes)
- **Running time:** $O\left(\sqrt[4]{N}\ \text{pol}ylog(N)\right)$
  - **Contrast:** Naïve Algorithm takes time $O\left(\sqrt{N}\ \text{pol}ylog(N)\right)$ to factor

- **Core idea:** find distinct $\mathrm{x}, \mathrm{x}' \in \mathbb{Z}_N^*$ such that $x = x' \bmod p$
  - Implies that x-x' is a multiple of p and, thus, GCD(x-x',N)=p (whp)

Pollard's Rho Algorithm

- General Purpose Factoring Algorithm
  - Doesn't assume (p-1) has no large prime factor

- Running time: $O\left(\sqrt[4]{N}\ \text{pol}ylog(N)\right)$

- **Core idea:** find distinct $\mathrm{x}, \mathrm{x}' \in \mathbb{Z}_N^*$ such that $x = x' \bmod p$ (but $x \neq x' \bmod q$)
  - Implies that x-x' is a multiple of p and, thus, GCD(x-x',N)=p
- **Question**: If we pick $\mathrm{k} = \mathrm{O}\left(\sqrt{p}\right)$ random $x^{(1)}, \dots, x^{(k)} \in \mathbb{Z}_N^*$ then what is the probability that we can find distinct i and j such that

$$x^{(i)} = x^{(j)} \bmod \mathrm{p}?$$

Pollard's Rho Algorithm

- **Question**: If we pick $\mathrm{k} = \mathrm{O}\left(\sqrt{p}\right)$ random $x^{(1)}, \ldots, x^{(k)} \in \mathbb{Z}_N^*$ then what is the probability that we can find distinct i and j such that $x^{(i)} = x^{(j)} \mathrm{mod\ p}$?
- **Answer:** $\geq 1/2$
- **Proof (sketch):** Use the Chinese Remainder Theorem + Birthday Bound

$$x^{(i)} = \left(x^{(i)}\ mod\ p, x^{(i)}\ mod\ q\right)$$

Note: We will also have $x^{(i)} \neq x^{(j)}$ mod q (whp)

Pollard's Rho Algorithm

- **Question**: If we pick $\mathrm{k} = \mathrm{O}\left(\sqrt{p}\right)$ random $x^{(1)}, \dots, x^{(k)} \in \mathbb{Z}_N^*$ then what is the probability that we can find distinct i and j such that $x^{(i)} = x^{(j)} \mathrm{mod\ p}$?
- **Answer:** $\geq {}^{1}/_{2}$
- **Challenge:** We do not know p or q so we cannot sort the $x^{(i)}$'s using the Chinese Remainder Theorem Representation

$$x^{(i)} = \left(x^{(i)}\ mod\ p, x^{(i)}\ mod\ q\right)$$

Problem: How can we identify the pair i and j such that $x^{(i)} = x^{(j)} \mathrm{mod\ p}$?

Pollard's Rho Algorithm

- Pollard's Rho Algorithm is similar the low-space version of the birthday attack

$F\left(x^{(i-1)}\right) = x^{(i)} \leftrightarrow \left(x^{(i)} \text{mod p}, x^{(i)} \text{mod q}\right)$

Input: N (product of two n bit primes)

$x^{(0)} \leftarrow \mathbb{Z}_N^*, \mathrm{x} = \mathrm{x}' = x^{(0)}$

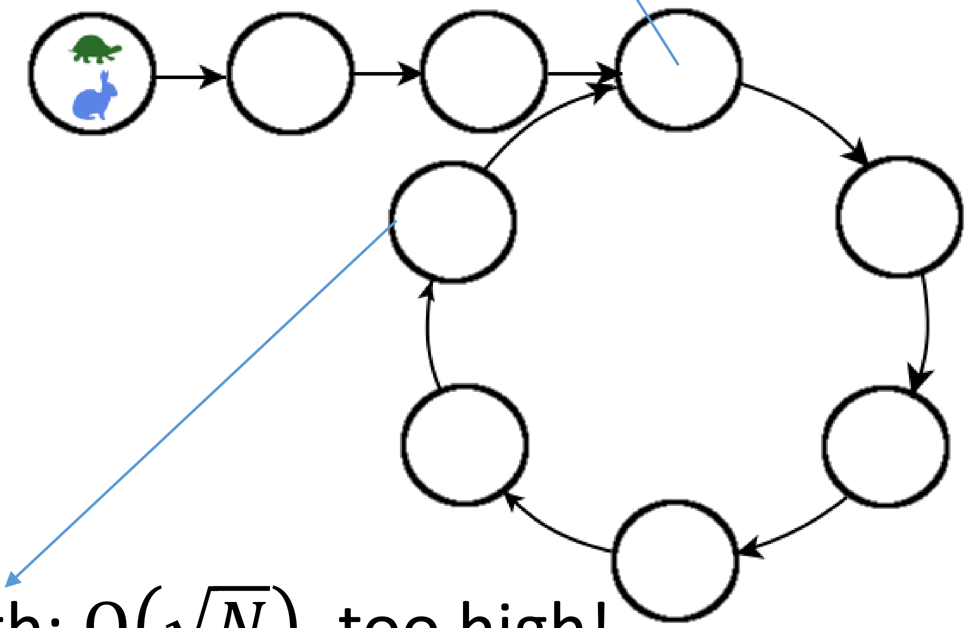
For i=1 to $2^{n/2}$

$x \leftarrow F(x)$

$x' \leftarrow F\left(F(x')\right)$

p = **GCD**(x-x',N)

if 1< p < N **return** p



Expected Cycle Length: $O\left(\sqrt{N}\right)$ too high!

Pollard's Rho Algorithm

- Pollard's Rho Algorithm is similar the low-space version of the birthday attack

Input: N (product of two n bit primes)

$x^{(0)} \leftarrow \mathbb{Z}_N^*$, $\mathrm{x} = \mathrm{x}' = x^{(0)}$

For i=1 to $2^{n/2}$

$x \leftarrow F(x)$

$x' \leftarrow F\big(F(x')\big)$

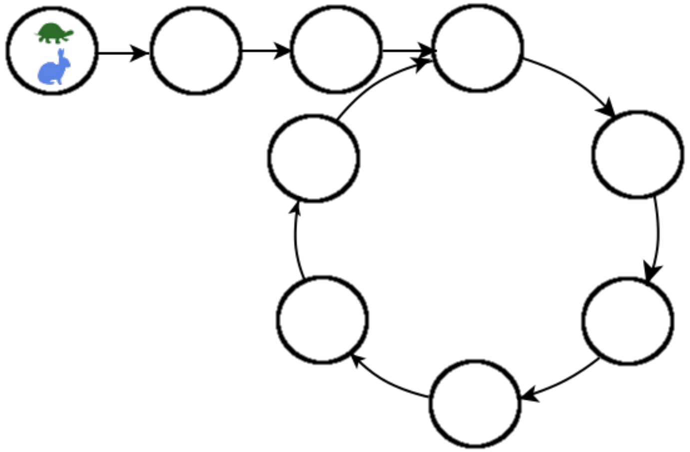
p = **GCD**(x-x',N)

if 1< p < N **return** p

Remark 1: F should have the property that

$\mathrm{F(x)} = F(x \bmod p) \bmod p$ i.e.,

$F(x) \leftrightarrow (F(x \bmod p) \bmod p, F_2(x) \bmod q)$



$x^{(i)} \mathrm{mod\ p}$

$(x^{(i)} \mathrm{mod\ p}, ~~x^{(i)} \mathrm{mod\ q}~~)$

Pollard's Rho Algorithm

- Pollard's Rho Algorithm is similar the low-space version of the birthday attack

Input: N (product of two n bit primes)

$x^{(0)} \leftarrow \mathbb{Z}_N^*$, $\mathrm{x} = \mathrm{x}' = x^{(0)}$

For i=1 to $2^{n/2}$

$x \leftarrow F(x)$

$x' \leftarrow F\big(F(x')\big)$

p = **GCD**(x-x',N)

if 1< p < N **return** p

Remark 1: F should have the property that

$\mathrm{F(x)} = F(x \; mod \; p) \; mod \; p$ i.e.,

$F(x) \leftrightarrow (F(x \; mod \; p) \; mod \; p, F(x) \; mod \; q)$

Remark 2: $F(x) = [x^2 + 1 \; mod \; N]$ will work since

$F(x) = [x^2 + 1 \; mod \; N]$

$\leftrightarrow (x^2 + 1 \; mod \; p, x^2 + 1 \; mod \; q)$

$\leftrightarrow (F([x \; \mathrm{mod} \; p]) \; mod \; p, F([x \; \mathrm{mod} \; q])mod \; q)$

Pollard's Rho Algorithm

- Pollard's Rho Algorithm is similar the low-space version of the birthday attack

Input: N (product of two n bit primes)

$x^{(0)} \leftarrow \mathbb{Z}_N^*$, $\mathrm{x} = \mathrm{x}' = x^{(0)}$

For i=1 to $2^{n/2}$

$x \leftarrow F(x)$

$x' \leftarrow F(F(x'))$

p = **GCD**(x-x',N)

if 1< p < N **return** p

Claim: Let $x^{(i+1)} = F\left(x^{(i)}\right)$ and suppose that for some distinct $\mathrm{i}, \mathrm{j} < 2^{n/2}$ we have $x^{(i)} = x^{(j)} \bmod \mathrm{p}$ but $x^{(i)} \neq x^{(j)}$. Then the algorithm will find p.

$x^{(j)} \equiv x^{(i)}$ mod p

$x^{(3)}$ mod p

$x^{(j)} \equiv x^{(i)}$ mod p

Expected Cycle Length: $O\left(\sqrt{p}\right)$

Pollard's Rho Algorithm (Summary)

- General Purpose Factoring Algorithm
  - Doesn't assume (p-1) has no large prime factor

- Expected Running Time: $O\left(\sqrt[4]{N}\ \text{pol}ylog(N)\right)$
  - (Birthday Bound)
  - (still exponential in number of bits $\sim 2^{n/4}$)
- Required Space: $O(\log(N))$

Quadratic Sieve Algorithm

- Runs in sub-exponential time $2^{O\left(\sqrt{\log N \ \log \log N}\right)} = 2^{O\left(\sqrt{n \log n}\right)}$
  - Still not polynomial time but $2^{\sqrt{n \log n}}$ is sub-exponential and grows much slower than $2^{n/4}$.

- **Core Idea**: Find $\mathrm{x}, \mathrm{y} \in \mathbb{Z}_N^*$ such that

$$x^2 = y^2 \bmod N$$

and

$$x \neq \pm y \bmod N$$

Quadratic Sieve Algorithm

- **Core Idea**: Find $\mathrm{x}, \mathrm{y} \in \mathbb{Z}_N^*$ such that

$$x^2 = y^2 \bmod N \quad (1)$$

and

$$x \neq \pm y \bmod N \quad (2)$$

Claim: gcd(x-y,N)∈ $\{p, q\}$

→N=pq divides $x^2 - y^2 = (x - y)(x + y)$. (by (1)).

→$(x - y)(x + y) \neq 0$ (by (2)).

→N does not divide $(x - y)$ (by (2)).

→N does not divide $(x + y)$. (by (2)).

→*p is a factor of exactly one of the terms* $(x - y)$ *and* $(x + y)$.

→*(q is a factor of the other term)*

Quadratic Sieve Algorithm

- **Core Idea**: Find $\mathrm{x}, \mathrm{y} \in \mathbb{Z}_N^*$ such that

$$x^2 = y^2 \bmod N$$

and

$$x \neq \pm y \bmod N$$

- **Key Question**: How to find such an $\mathrm{x}, \mathrm{y} \in \mathbb{Z}_N^*$?
- **Step 1: (Initialize** j=0);

For $\mathrm{x} = \sqrt{N} + 1, \sqrt{N} + 2, \ldots, \sqrt{N} + i, \ldots$

$$\mathrm{q} \leftarrow \left[\left(\sqrt{N} + i\right)^2 \, mod \, N\right] = \left[2i\sqrt{N} + i^2 \, mod \, N\right]$$

Check if q is B-smooth (all prime factors of q are in $\{p_1,\ldots,p_k\}$ where $p_k < B$).

If q is B smooth then factor q, increment j and define

$$\mathrm{q_j} \leftarrow q = \prod_{i=1}^{k} p_i^{e_{j,i}}, \qquad \text{and} \qquad \mathrm{x_j} \leftarrow x$$

Quadratic Sieve Algorithm

- **Core Idea**: Find $\mathrm{x}, \mathrm{y} \in \mathbb{Z}_N^*$ such that
$$x^2 = y^2 \bmod N$$
and
$$x \neq \pm y \bmod N$$
- **Key Question**: How to find such an $\mathrm{x}, \mathrm{y} \in \mathbb{Z}_N^*$?
- **Step 2:** Once we have $\ell > k$ equations of the form
$$\mathrm{q_j} \leftarrow q = \prod_{i=1}^{k} p_i^{e_{j,i}},$$

We can use linear algebra to find subset S such that for each $i \leq k$ we have
$$\sum_{j \in S} e_{j,i} = 0 \; mod \; 2.$$

Quadratic Sieve Algorithm

- **Key Question**: How to find $x, y \in \mathbb{Z}_N^*$ such that $x^2 = y^2 \bmod N$ and $x \neq \pm y \bmod N$?
- **Step 2:** Once we have $\ell > k$ equations of the form

$$q_j \leftarrow q = \prod_{i=1}^{k} p_i^{e_{j,i}},$$

We can use linear algebra to find a subset S such that for each $i \leq k$ we have

$$\sum_{j \in S} e_{j,i} = 0 \; mod \; 2.$$

Thus,

$$\prod_{j \in S} q_j = \prod_{i=1}^{k} p_i^{\sum_{j \in S} e_{j,i}} = \left(\prod_{i=1}^{k} p_i^{\frac{1}{2} \sum_{j \in S} e_{j,i}} \right)^2 = y^2$$

Quadratic Sieve Algorithm

- **Key Question**: How to find $\mathrm{x}, \mathrm{y} \in \mathbb{Z}_N^*$ such that $x^2 = y^2 \bmod N$ and $x \neq \pm y \bmod N$?

Thus,

$$\prod_{j \in S} \mathrm{q_j} = \prod_{i=1}^{k} p_i^{\sum_{j \in S} e_{j,i}} = \left(\prod_{i=1}^{k} p_i^{\frac{1}{2} \sum_{j \in S} e_{j,i}} \right)^2 = y^2$$

But we also have

$$\prod_{j \in S} \mathrm{q_j} = \prod_{j \in S} ({x_j}^2) = \left(\prod_{j \in S} x_j \right)^2 = x^2 \, mod \, N$$

Quadratic Sieve Algorithm (Summary)

- Appropriate parameter tuning yields sub-exponential time algorithm $2^{O\left(\sqrt{\log N\ \log\log N}\right)} = 2^{O\left(\sqrt{n \log n}\right)}$
  - Still not polynomial time but $2^{\sqrt{n \log n}}$ grows much slower than $2^{n/4}$.

Discrete Log Attacks

- Pohlig-Hellman Algorithm
  - Given a cyclic group $\mathbb{G}$ of non-prime order q=| $\mathbb{G}$ |=rp
  - Reduce discrete log problem to discrete problem(s) for subgroup(s) of order p (or smaller).
  - Preference for prime order subgroups in cryptography
- Baby-step/Giant-Step Algorithm
  - Solve discrete logarithm in time $O\left(\sqrt{q}\, polylog(q)\right)$
- Pollard's Rho Algorithm
  - Solve discrete logarithm in time $O\left(\sqrt{q}\, polylog(q)\right)$
  - Bonus: Constant memory!
- Index Calculus Algorithm
  - Similar to quadratic sieve
  - Runs in sub-exponential time $2^{O\left(\sqrt{\log q \log\log q}\right)}$
  - Specific to the group $\mathbb{Z}_p^*$ (e.g., attack doesn't work elliptic-curves)

Discrete Log Attacks

- **Pohlig-Hellman Algorithm**
  - Given a cyclic group $\mathbb{G}$ of non-prime order q=| $\mathbb{G}$ |=rp
  - Reduce discrete log problem to discrete problem(s) for subgroup(s) of order p (or smaller).
  - Preference for prime order subgroups in cryptography
- Let $\mathbb{G} = \langle g \rangle$ and $\mathrm{h} = g^x \in \mathbb{G}$ be given. For simplicity assume that r is prime and r < p.
- Observe that $\langle g^r \rangle$ generates a subgroup of size p and that $\mathrm{h^r} \in \langle g^r \rangle$.
  - Solve discrete log problem in subgroup $\langle g^r \rangle$ with input $\mathrm{h^r}$.
  - Find z such that $\mathrm{h^{rz}} = g^{rz}$.
- Observe that $\langle g^p \rangle$ generates a subgroup of size r and that $\mathrm{h^p} \in \langle g^p \rangle$.
  - Solve discrete log problem in subgroup $\langle g^p \rangle$ with input $\mathrm{h^p}$.
  - Find y such that $\mathrm{h^{yp}} = g^{yp}$.
- Chinese Remainder Theorem $\mathrm{h} = g^x$ where $\mathrm{x} \leftrightarrow ([z \bmod p], [y \bmod r])$

Baby-step/Giant-Step Algorithm

- Input: $\mathbb{G} = \langle g \rangle$ of order q, generator g and $\mathrm{h} = g^x \in \mathbb{G}$
- Set $t = \lfloor \sqrt{q} \rfloor$

For i =0 to $\left\lfloor \frac{q}{t} \right\rfloor$

$$g_i \leftarrow g^{it}$$

Sort the pairs (i,g_i) by their second component

For i =0 to t

$$h_i \leftarrow hg^i$$

if $h_i = g_k \in \{g_0, \ldots, g_t\}$ then

return [kt-i mod q]

$$h_i = hg^i = g^{kt}$$

$$\rightarrow h = g^{kt-i}$$

Discrete Log Attacks

- Baby-step/Giant-Step Algorithm
  - Solve discrete logarithm in time $O\left(\sqrt{q}\, polylog(q)\right)$
  - Requires memory $O\left(\sqrt{q}\, polylog(q)\right)$
- Pollard's Rho Algorithm
  - Solve discrete logarithm in time $O\left(\sqrt{q}\, polylog(q)\right)$
  - Bonus: Constant memory!
- **Key Idea:** Low-Space Birthday Attack (*) using our collision resistant hash function

$$H_{g,h}(x_1, x_2) = g^{x_1} h^{x_2}$$

$$H_{g,h}(y_1, y_2) = H_{g,h}(x_1, x_2) \rightarrow\ h^{y_2 - x_2} = g^{x_1 - y_1}$$

$$\rightarrow h = g^{(x_1 - y_1)(y_2 - x_2)^{-1}}$$

(*) A few small technical details to address

Discrete Log Attacks

- Baby-step/Giant-Step Algorithm
  - Solve discrete logarithm in time $O\left(\sqrt{q}\, polylog(q)\right)$
  - Requires memory $O\left(\sqrt{q}\, polylog(q)\right)$
- Pollard's Rho Algorithm
  - Solve discrete logarithm in time $O\left(\sqrt{q}\, pol$
  - Bonus: Constant memory!
- **Key Idea:** Low-Space Birthday Attack (*)

$$H_{g,h}(x_1, x_2) = g^{x_1} h^{x_2}$$
$$H_{g,h}(y_1, y_2) = H_{g,h}(x_1, x_2)$$

$$\rightarrow\ h^{y_2 - x_2} = g^{x_1 - y_1}$$
$$\rightarrow h = g^{(x_1 - y_1)(y_2 - x_2)^{-1}}$$

(*) A few small technical details to address

Remark: We used discrete-log problem to construct collision resistant hash functions.

Security Reduction showed that attack on collision resistant hash function yields attack on discrete log.

→Generic attack on collision resistant hash functions (e.g., low space birthday attack) yields generic attack on discrete log.

Discrete Log Attacks

- Index Calculus Algorithm
  - Similar to quadratic sieve
  - Runs in sub-exponential time $2^{O\left(\sqrt{\log q \log\log q}\right)}$
  - Specific to the group $\mathbb{Z}_p^*$ (e.g., attack doesn't work elliptic-curves)

- As before let $\{p_1,\dots,p_k\}$ be set of prime numbers < B.
- **Step 1.A:** Find $\ell > k$ distinct values $x_1, \dots, x_k$ such that $g_j = [g^{x_j} \mod p]$ is B-smooth for each j. That is

$$g_j = \prod_{i=1}^{k} p_i^{e_{i,j}} .$$

Discrete Log Attacks

- As before let $\{p_1,\ldots,p_k\}$ be set of prime numbers < B.
- **Step 1.A:** Find $\ell > k$ distinct values $x_1, \ldots, x_k$ such that $g_j = [g^{x_j} \: mod \: p]$ is B-smooth for each j. That is

$$g_j = \prod_{i=1}^{k} p_i^{e_{i,j}} .$$

- **Step 1.B:** Use linear algebra to solve the equations

$$x_j = \sum_{i=1}^{k} \left(\mathbf{log_g \, p_i} \right) \times e_{i,j} \ \text{ mod } (p-1).$$

(Note: the $\mathbf{log_g p_i}$'s are the unknowns)

Discrete Log

- As before let $\{p_1,\ldots,p_k\}$ be set of prime numbers < B.
- **Step 1 (precomputation):** Obtain $y_1,\ldots,y_k$ such that $p_i = g^{y_i}\ mod\ p$.
- **Step 2:** Given discrete log challenge $h=g^x$ mod p.
  - Find y such that $[g^y \mathrm{h\ mod\ p}]$ is B-smooth

$$[g^y \mathrm{h\ mod\ p}] = \prod_{i=1}^{k} p_i^{e_i}$$

$$= \prod_{i=1}^{k} (g^{y_i})^{e_i} = g^{\sum_i e_i y_i}$$

Discrete Log

- As before let $\{p_1,\ldots,p_k\}$ be set of prime numbers < B.
- **Step 1 (precomputation):** Obtain $y_1,\ldots,y_k$ such that $p_i = g^{y_i} \, mod \, p$.
- **Step 2:** Given discrete log challenge $h=g^x$ mod p.
  - Find z such that $[g^z \text{h mod p}]$ is B-smooth

$$[g^z \text{h mod p}] = g^{\sum_i e_i y_i} \rightarrow h = g^{\sum_i e_i y_i - z}$$

$$\rightarrow x = \sum_i e_i y_i - z$$

- **Remark:** Precomputation costs can be amortized over many discrete log instances
  - In practice, the same group $\mathbb{G} = \langle g \rangle$ and generator g are used repeatedly.

Reference: https://www.weakdh.org/

NIST Guidelines (Concrete Security)

Best known attack against 1024 bit RSA takes time (approximately) 2^{80}

Symmetric Key Size (bits)	RSA and Diffie-Hellman Key Size (bits)	Elliptic Curve Key Size (bits)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	521

Table 1: NIST Recommended Key Sizes

NIST Guidelines (Concrete Security)

Diffie-Hellman uses subgroup of $\mathbb{Z}_p^*$ size q

Symmetric Key Size (bits)	RSA and Diffie-Hellman Key Size (bits)		Elliptic Curve Key Size (bits)
80	1024		160
112	2048	q=224 bits	224
128	3072	q=256 bits	256
192	7680	q=384 bits	384
256	15360	q=512 bits	521

Table 1: NIST Recommended Key Sizes

<table>
<tr><th colspan="2">Security Strength</th><th>2011 through 2013</th><th>2014 through 2030</th><th>2031 and Beyond</th></tr>
<tr><td rowspan="2">80</td><td>Applying</td><td>Deprecated</td><td colspan="2">Disallowed</td></tr>
<tr><td>Processing</td><td colspan="3">Legacy use</td></tr>
<tr><td rowspan="2">112</td><td>Applying</td><td rowspan="2">Acceptable</td><td rowspan="2">Acceptable</td><td>Disallowed</td></tr>
<tr><td>Processing</td><td>Legacy use</td></tr>
<tr><td>128</td><td rowspan="3">Applying/Processing</td><td>Acceptable</td><td>Acceptable</td><td>Acceptable</td></tr>
<tr><td>192</td><td>Acceptable</td><td>Acceptable</td><td>Acceptable</td></tr>
<tr><td>256</td><td>Acceptable</td><td>Acceptable</td><td>Acceptable</td></tr>
</table>

NIST's security strength guidelines, from Specialist Publication SP 800-57
Recommendation for Key Management – Part 1: General (Revision 3)