Cryptography
CS 555

Week 12:

- Discrete Log Attacks + NIST Recommendations for Concrete Security Parameters
- Identification Schemes + Schnorr Signatures
- El Gamal

Readings: Katz and Lindell Chapter 10 & Chapter 11.1-11.2, 11.4

Homework 4 Due Thursday (4/8) at 11:59PM on Gradescope

Spring 2021

Week 12: Topic 0: Discrete Log Attacks + NIST Recommendations for Concrete Security Parameters

Factoring Algorithms (Summary)

- Pollard's p-1 Algorithm
  - Works when $N = pq$ where (p-1) has only "small" prime factors
  - **Defense**: Ensure that p (resp. q) is a strong prime (p-1) has no "small" prime factors.
  - **Note**: A random prime is strong with high probability.
- Pollard's Rho Algorithm
  - General purpose factoring algorithm
  - **Core:** Low Space Cycle Detection
  - **Time:** $\mathrm{T(N)} = O\left(\sqrt[4]{N}\ \mathrm{poly}log(N)\right)$
  - Naïve Algorithm takes time $O\left(\sqrt{N}\ \mathrm{poly}log(N)\right)$ to factor
- Quadratic Sieve
  - **Time:** $2^{O\left(\sqrt{\log N\ \log\log N}\right)} = 2^{O\left(\sqrt{n \log n}\right)}$ (sub-exponential, but not polynomial time)
  - Preprocessing + Linear Algebra: find $\mathrm{x}, \mathrm{y} \in \mathbb{Z}_N^*$ such that $x^2 = y^2 \bmod N$ and $x \neq \pm y \bmod N$?

Discrete Log Attacks

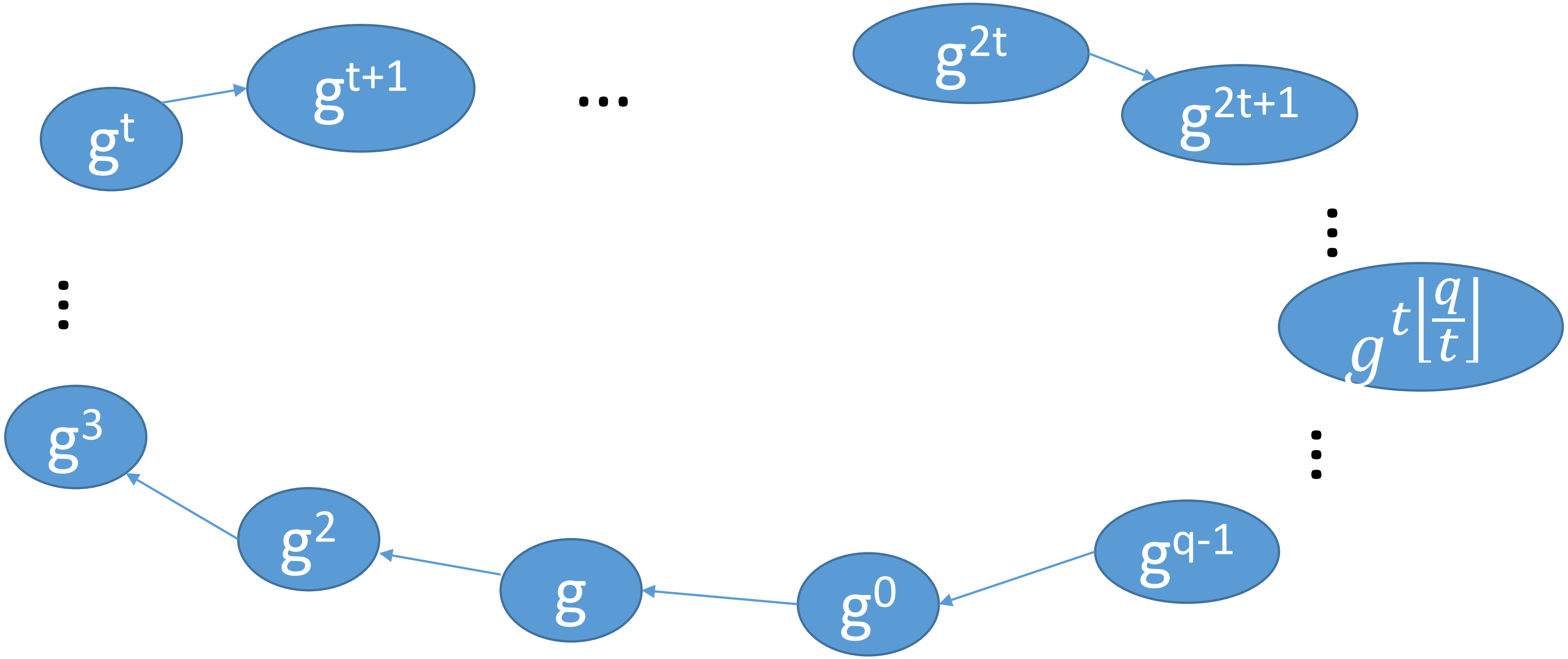
- Pohlig-Hellman Algorithm
  - Given a cyclic group $\mathbb{G}$ of non-prime order q=| $\mathbb{G}$ |=rp
  - Reduce discrete log problem to discrete problem(s) for subgroup(s) of order p (or smaller).
  - Preference for prime order subgroups in cryptography
- Baby-step/Giant-Step Algorithm
  - Solve 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 $\mathbb{Z}_q^*$ (e.g., attack doesn't work against elliptic-curve groups)

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 $\mathrm{h}^\mathrm{r} \in \langle g^r \rangle$.
  - Solve discrete log problem in subgroup $\langle g^r \rangle$ with input h^r.
  - Find z such that $\mathrm{h}^\mathrm{r} = g^{rz} \rightarrow rz = rx \bmod \mathrm{p}$.
- Observe that $\langle g^p \rangle$ generates a subgroup of size r and that $\mathrm{h}^\mathrm{p} \in \langle g^p \rangle$.
  - Solve discrete log problem in subgroup $\langle g^p \rangle$ with input h^p.
  - Find y such that $\mathrm{h}^\mathrm{p} = g^{yp} \rightarrow \mathrm{p}z = px \bmod \mathrm{r}$.
- Chinese Remainder Theorem $\mathrm{h} = g^x$ where $\mathrm{x} \leftrightarrow ([z \bmod p], [y \bmod r])$

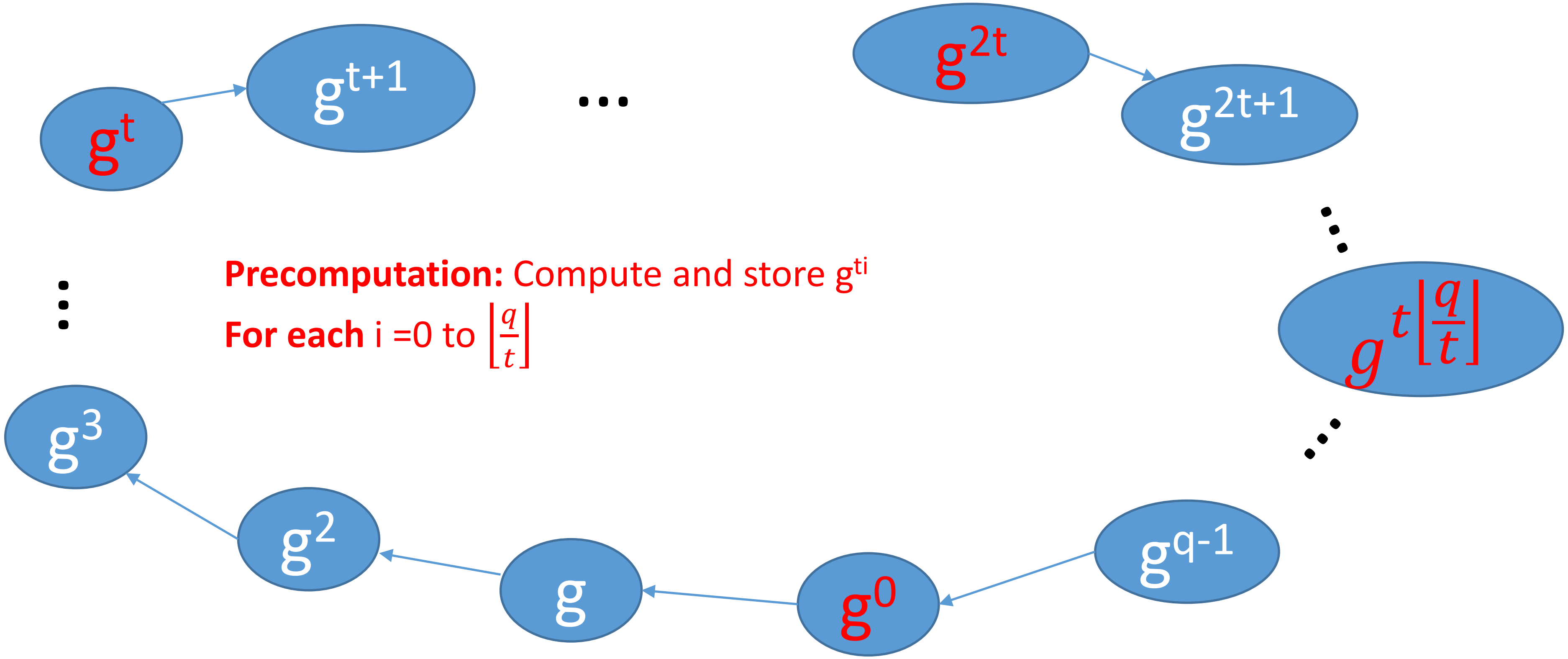
Baby-step/Giant-Step Algorithm

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



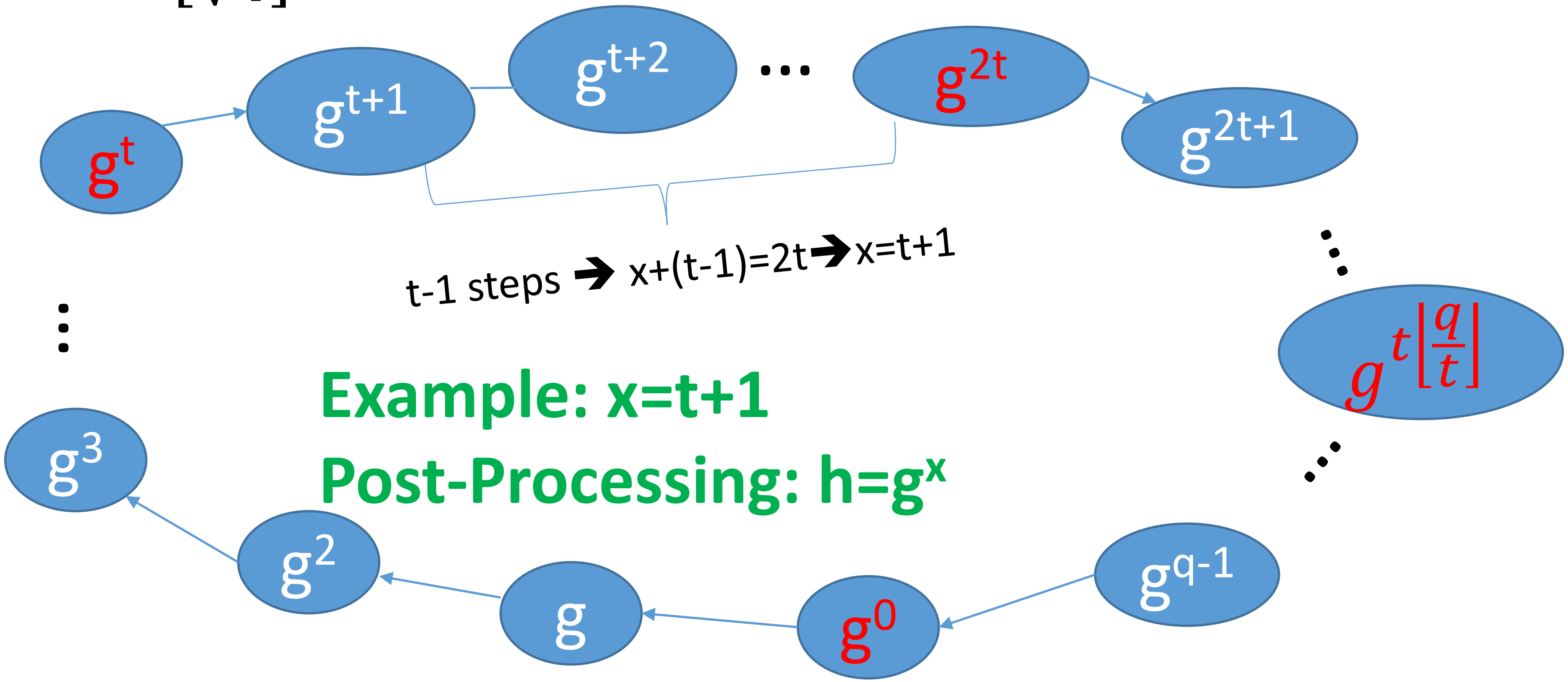
Baby-step/Giant-Step Algorithm

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



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 $t = \lfloor \sqrt{q} \rfloor$



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 $h_i = g_k \in \{g_0, \dots, g_t\}$ 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 $O\left(\sqrt{q}\, pol\right.$
  - 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 $2^{O\left(\sqrt{\log p \log\log p}\right)}$
  - Specific to the group $\mathbb{Z}_p^*$ (e.g., attack doesn't work on elliptic-curve groups)

- As before let $\{p_1,\ldots,p_k\}$ denote the 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

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

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

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

(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 z such that $[g^z \mathrm{h} \bmod \mathrm{p}]$ is B-smooth

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

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

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 $p_i = g^{y_i} \bmod p$.
- **Step 2:** Given discrete log challenge $h=g^x \bmod p$.
  - Find z such that $[g^z \mathrm{h} \bmod \mathrm{p}]$ is B-smooth

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

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

- **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

NIST Guidelines (Concrete Security)

$$112 \text{ bits} = \frac{\log 2^{224}}{2} = \log\sqrt{2^{224}} \text{ bits (Pollard's Rho)}$$

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

NIST Guidelines (Concrete Security)

$112 \text{ bits} \approx \sqrt{2048 \log 2048}$ bits (Index Calculus)

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)

Signature Length

- RSA-FDH
  - 128-bit security ➔ $\log_2(N) > 3072$
  - RSA-FDH Signatures are at least 3Kb long
  - Are shorter signatures possible?

- RSA Ciphertexts/RSA KEM
  - At least 3Kb long for 128-bit security
  - Shorter Ciphertexts

Identification Scheme

- Interactive protocol that allows one party to prove its identify (authenticate itself) to another
- Two Parties: Prover and Verifier
  - Prover has secret key sk and Verifier has public key pk

1. Prover runs P_1(sk) to obtain (I,st) ---- initial message I, state st
   - Sends I to Verifier
2. Verifier picks random message r from distribution Ω_{pk} and sends r to Prover
3. Prover runs P_2(sk,st,r) to obtain s and sends s to verifier
4. Verifier checks if V(pk,r,s)=I

Identification Scheme

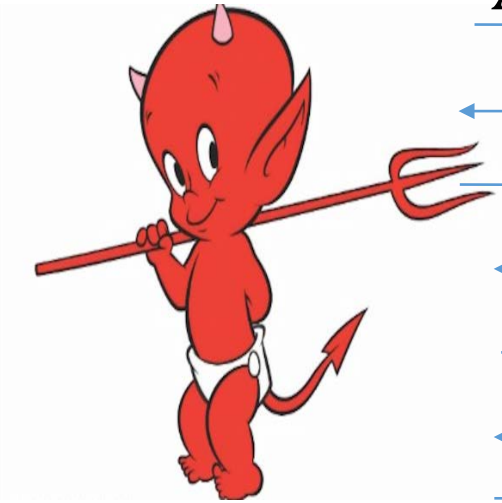
1. Prover runs P_1(sk) to obtain (I,st) ---- initial message I, state st
    1. Sends I to Verifier
2. Verifier picks random message r from distribution Ω_{pk} and sends r to Prover
3. Prover runs P_2(sk,st,r) to obtain s and sends s to verifier
4. Verifier checks if V(pk,r,s)=I

An eavesdropping attacker obtains a transcript (I,r,s) of all the message sent.

Transcript Oracle: $\mathrm{Trans}_{sk}(.)$ runs honest execution and outputs transcript.

Identification Game ($\mathrm{Ident}_{A,\Pi}(n)$)

Public Key: pk



$Again$

$\mathbf{Trans}_{sk}(.)$

$Again$

⋮

$\mathbf{Trans}_{sk}(.)$

I

$r \leftarrow \Omega_{pk}$

$Again$

$\mathbf{Trans}_{sk}(.)$

s

$$Ident_{A,\Pi}(n) = \begin{cases} 1 \; if \; V(pk, r, s) = I \\ 0 \; otherwise \end{cases}$$

(pk,sk) = Gen(.)

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

Schnorr Identification Scheme

- Verifier knows $h=g^x$
- Prover knows x such that $h=g^x$

1. Prover runs $P_1(x)$ to obtain $\left(k \in \mathbb{Z}_q, I = g^k\right)$ and sends initial message I to verifier
2. Verifier picks random $r \in \mathbb{Z}_q$ (q is order of the group) and sends r to prover
3. Prover runs $P_2(x,k,r)$ to obtain $s \coloneqq [rx + k \ mod\ q]$ and sends s to Verifier
4. Verifier checks if $g^s * (h^{-1})^r = I = g^k$

Schnorr Identification Scheme

- Verifier knows $h=g^x$
- Prover knows x such that $h=g^x$

1. Prover runs $P_1(x)$ to obtain $\left(k \in \mathbb{Z}_q, I = g^k\right)$ and sends initial message I to verifier
2. Verifier picks random $r \in \mathbb{Z}_q$ (q is order of the group) and sends r to prover
3. Prover runs $P_2(x,k,r)$ to obtain $s \coloneqq [rx + k \; mod \; q]$ and sends s to Verifier
4. Verifier checks if $g^s * (h^{-1})^r = I = g^k$

$$g^s * (h^{-1})^r = g^{rx+k \; mod \; q} * g^{-xr} = g^k$$

Schnorr Identification Scheme

- Verifier knows $h = g^x$
- Prover knows x such that $h = g^x$
- Prover runs $P_1(x)$ to obtain $\left(k \in \mathbb{Z}_q, I = g^k\right)$ and sends initial message I to verifier
- Verifier picks random $r \in \mathbb{Z}_q$ (q is order of the group) and sends r to prover
- Prover runs P1(x,k,r) to obtain $s := [rx + k \bmod q]$ and sends s to Verifier
- Verifier checks if $g^s * (h^{-1})^r = I = g^k$

Theorem 12.11: If the discrete-logarithm problem is hard (relative to group generator) then Schnorr identification scheme is secure.

Fiat-Shamir Transform

- Identification Schemes can be transformed into signatures
- $\mathrm{Sign}_{sk}(m)$
  - First compute $(I,st)= P_1(sk)$ (as prover)
  - Next compute the challenge $\boldsymbol{r} = \boldsymbol{H}(\boldsymbol{I}, \boldsymbol{m})$ (as verifier)
  - Compute the response $s = P_2(sk,st,r)$
  - Output signature (r,s)
- $\mathrm{Vrfy}_{pk}(m,(r,s))$
  - Compute $I := V(pk,r,s)$
  - Check that $H(I,m)=r$

Theorem 12.10: If the identification scheme is secure and H is a random oracle then the above signature scheme is secure.

Schnorr Signatures via Fiat-Shamir

- Public Key: $h=g^x$ in cyclic group $\langle g \rangle$ of order q.
- Secret Key: x
- $Sign_{sk}(m)$
  1. Select random $k \in \mathbb{Z}_q$ and set $I = g^k$.
  2. $\boldsymbol{r = H(I, m)}$
  3. Return $\sigma = (r, s)$ where $s \coloneqq [rx + k \ mod \ q]$
- $Verify_{pk}\big(m, \sigma = (r, s)\big)$
  - Compute $g^s * (h^{-1})^r = g^{s-rx}$ and check if $r = H(g^{s-rx}, m)$

Schnorr Signatures

- $Sign_{sk}(m)$
  1. Select random $k \in \mathbb{Z}_q$ and set $I = g^k$.
  2. $\boldsymbol{r = H(I, m)}$
  3. Return $\sigma = (r, s)$ where $s := [rx + k \ mod \ q]$
- $Verify_{pk}\big(m, \sigma = (r, s)\big)$
  - Compute $g^s * (h^{-1})^r = g^{s-rx}$ and check if $r = H(g^{s-rx}, m)$

Corollary (of Thms 12.10 + 12.11): If the discrete-logarithm problem is hard (relative to group generator) then Schnorr Signatures are secure in the random oracle model.

- Independent of size of original group
(r^th residue subgroup).

- Independent of #bits to represent group element
(Elliptic Curve Pairs)

Depends only on order of the subgroup q!

DLOG 128 bit security: $\lceil \log_2 q \rceil \approx 256$

$= g^k$.

$+ k \; mod \; q$

check if r =

Advantages:

- **Short Signatures** $\|\sigma\| = \|r\| + \|s\| = 2\lceil \log_2 q \rceil$ bits
- Fast and Efficient
- Patent Expired: February 2008

- Independent of size of original group (r^{th} residue subgroup).
- Independent of #bits to represent group element (Elliptic Curve Pairs)

Depends only on order of the subgroup q!

DLOG 128 bit security: $\lceil \log_2 q \rceil \approx 256$

≈ 512 bit signatures

$= g^k.$

$+ k \; mod \; q$

check if r =

Advantages:

- **Short Signatures** $\|\sigma\| = \|r\| + \|s\| = 2\lceil \log_2 q \rceil$ bits
- Fast and Efficient
- Patent Expired: February 2008

Short Schnorr Signatures

- $Sign_{sk}(m)$
  1. Select random $k \in \mathbb{Z}_q$ and set $I = g^k$.
  2. $\boldsymbol{r = H(I, m)}$ **//** $\boldsymbol{r \leq \sqrt{q}}$
  3. Return $\sigma = (r, s)$ where $s := [rx + k \ mod \ q]$
- $Verify_{pk}\big(m, \sigma = (r, s)\big)$
  - Compute $g^s * (h^{-1})^r = g^{s-rx}$ and check if $r = H(g^{s-rx}, m)$
- **Short Signatures** $\|\sigma\| = \|r\| + \|s\| = \mathbf{1.5}\lceil \log_2 q \rceil$ bits
  - New Result: Short Schnorr Signatures are also secure in Generic Group+ Random Oracle Model https://eprint.iacr.org/2019/1105.pdf
  - 384 bit signatures for 128-bit security
  - BLS Signatures: 256 bit signatures for 128-bit security (computational overhead is much higher)

Digital Signature Algorithm (DSA)

DSA: $\langle \boldsymbol{g} \rangle$ is subgroup of $\mathbb{Z}_p^*$ of order q

ECDSA: $\langle \boldsymbol{g} \rangle$ is order q subgroup of elliptic curve

- Secret key is x, public key is $h=g^x$ along with generator g (of order q)
- $\text{Sign}_{sk}(m)$
  - Pick random $\left(k \in \mathbb{Z}_q\right)$ and set $r = F\left(g^k\right) \in \mathbb{Z}_q$
  - Compute s $\coloneqq \left[k^{-1}(xr + H(m))\ mod\ q\right]$
  - Output signature (r,s)
- $\text{Vrfy}_{pk}(m,(r,s))$ check to make sure that

$$r = F\left(g^{H(m)s^{-1}} h^{rs^{-1}}\right)$$

Digital Signature Algorithm (DSA)

- $\mathrm{Sign}_{sk}(m)$
  - Pick random $\left(k \in \mathbb{Z}_q\right)$ and set $r = F\left(g^k\right) = [g^k \; mod \; q]$
  - Compute $s \coloneqq [k^{-1}(xr + H(m)) \; mod \; q]$
  - Output signature (r,s)
- $\mathrm{Vrfy}_{pk}(m,(r,s))$ check to make sure that

$$
\begin{aligned}
r &= F\left(g^{H(m)s^{-1}} h^{rs^{-1}}\right) \\
&= F\left(g^{H(m)k(xr+H(m))^{-1}} g^{xrk(xr+H(m))^{-1}}\right) \\
&= F\left(g^{(H(m)+xr)k(xr+H(m))^{-1}}\right) \\
&= F\left(g^k\right) \coloneqq r
\end{aligned}
$$

Digital Signature Algorithm (DSA)

- Secret key is x, public key is $h=g^x$ along with generator g (of order q)
- $\mathrm{Sign}_{sk}(m)$
  - Pick random $(k \in \mathbb{Z}_q)$ and set $r = F(g^k) = [g^k \ mod\ q]$
  - Compute $s \coloneqq [k^{-1}(xr + H(m))\ mod\ q]$
  - Output signature (r,s)
- $\mathrm{Vrfy}_{pk}(m,(r,s))$ check to make sure that

$$r = F\left(g^{H(m)s^{-1}} h^{rs^{-1}}\right)$$

Theorem: If H and F are modeled as random oracles then DSA is secure.

Weird Assumption for F(.)?

- **Theory:** DSA Still lack compelling proof of security from standard crypto assumptions
- **Practice:** DSA has been used/studied for decades without attacks

Digital Signature Algorithm (DSA)

- Secret key is x, public key is $h=g^x$
- $\text{Sign}_{sk}(m)$
  - Pick random $\left(k \in \mathbb{Z}_q\right)$ and set $r = F\left(g^k\right) = [g^k \; mod \; q]$
  - Compute s $\coloneqq [k^{-1}(xr + H(m)) \; mod \; q]$
  - Output signature (r,s)
- $\text{Vrfy}_{pk}(m,(r,s))$ check to make sure that

$$r = F\left(g^{H(m)s^{-1}} h^{rs^{-1}}\right)$$

Remark: If signer signs two messages with same random $k \in \mathbb{Z}_q$ then attacker can find secret key sk!

- **Theory:** Negligible Probability this happens
- **Practice:** Will happen if a weak PRG is used
- Sony PlayStation (PS3) hack in 2010.

Certificate Authority

- Trusted Authority (CA)
  - $m_{CA \rightarrow Amazon}$="Amazon's public key is pk_{Amazon} (date,expiration,###)"
  - $cert_{CA \rightarrow Amazon} = Sign_{SK_{CA}}(m)$
- Delegate Authority to other CA_1
  - Root CA signs m= "CA_1 public key is pk_{CA1} (date,expiration,###) can issue certificates"
  - Verifier can check entire certification chain
- Revocation List Signed Daily
- Decentralized Web of Trust (PGP)

One-Time Signature Scheme

- Weak notion of one-time secure signature schemes
  - Attacker makes one query to oracle $Sign_{sk}(.)$ and then attempts to output forged signature for m'
  - If attacker sees two different signatures then guarantees break down
- Achievable from Hash Functions
  - No number theory!
  - No Random Oracles!

Lamport's Signature Scheme (from OWFs)

$$sk = \begin{bmatrix} x_{1,0} & x_{2,0} & x_{3,0} \\ x_{1,1} & x_{2,1} & x_{3,1} \end{bmatrix}$$

$$pk = \begin{bmatrix} y_{1,0} & y_{2,0} & y_{3,0} \\ y_{1,1} & y_{2,1} & y_{3,1} \end{bmatrix}$$

$$x_{i,j} \in \{0,1\}^n \ (uniform)$$
$$y_{i,j} = f\left(x_{i,j}\right)$$

Assumption: f is a One-Way Function

Lamport's Signature Scheme (from OWFs)

$$sk = \begin{bmatrix} x_{1,0} & x_{2,0} & x_{3,0} \\ x_{1,1} & x_{2,1} & x_{3,1} \end{bmatrix}$$

$$pk = \begin{bmatrix} y_{1,0} & y_{2,0} & y_{3,0} \\ y_{1,1} & y_{2,1} & y_{3,1} \end{bmatrix}$$

$$Sign_{sk}(011) = \left(x_{1,0}, x_{2,1}, x_{3,1}\right)$$

Lamport's Signature Scheme (from OWFs)

$$sk = \begin{bmatrix} x_{1,0} & x_{2,0} & x_{3,0} \\ x_{1,1} & x_{2,1} & x_{3,1} \end{bmatrix}$$

$$pk = \begin{bmatrix} y_{1,0} & y_{2,0} & y_{3,0} \\ y_{1,1} & y_{2,1} & y_{3,1} \end{bmatrix}$$

$$Sign_{sk}(011) = \left(x_{1,0}, x_{2,1}, x_{3,1}\right)$$

$$\mathrm{Vrfy}_{pk}\left(011, (x_1, x_2, x_3)\right) = \begin{cases} 1 & \text{if } f(x_1) = y_{1,0} \wedge f(x_2) = y_{2,1} \wedge f(x_3) = y_{3,1} \\ 0 & \text{otherwise} \end{cases}$$

Lamport's Signature Scheme

Theorem 12.16: Lamport's Signature Scheme is a secure one-time signature scheme (assuming f is a one-way function).

Proof Sketch: Signing a fresh message requires inverting $f\left(x_{i,j}\right)$ for random $x_{i,j}$.

Remark: Attacker can break scheme if he can request two signatures.

How?

Request signatures of both 0^n and 1^n.

Lamport's Signature Scheme

Remark: Attacker can break scheme if he can request two signatures.

How?

Request signatures of both 0^n and 1^n.

$$sk = \begin{bmatrix} x_{1,0} & x_{2,0} & x_{3,0} \\ x_{1,1} & x_{2,1} & x_{3,1} \end{bmatrix}$$

$$Sign_{sk}(000) = (x_{1,0}, x_{2,0}, x_{3,0})$$

$$Sign_{sk}(111) = (x_{1,1}, x_{2,1}, x_{3,1})$$

Secure Signature Scheme from OWFs

Theorem 12.22: secure/stateless signature scheme from collision-resistant hash functions.

- Collision Resistant Hash Functions do imply OWFs exist

Remark: Possible to construct signature scheme Π which is existentially unforgeable under an adaptive chosen message attacks using the minimal assumption that one-way functions exist.

Week 13 Topic 1: El-Gamal Encryption

El-Gamal Encryption

- **Key Generation:**
  - **Generate cyclic group <g> of prime order q**
  - Pick random $x \leq q$ and compute $h = g^x$
- **Public Key:** g, h
- **Secret Key:** $x = \mathrm{dlog}_g(h)$

El-Gamal Encryption

- **Public Key:** g, h
- **Secret Key:** $x = \mathrm{dlog}_g(h)$
- $\mathrm{Enc}_{\mathrm{pk}}(m) = \langle g^y, m \cdot h^y \rangle$ for a random y $\in \mathbb{Z}_q$
- $\mathrm{Dec}_{\mathrm{sk}}(c = (c_1, c_2)) = c_2 c_1^{-x}$

$$\begin{aligned} \mathrm{Dec}_{\mathrm{sk}}(g^y, m \cdot h^y) &= m \cdot h^y (g^y)^{-x} \\ &= m \cdot h^y (g^y)^{-x} \\ &= m \cdot (g^x)^y (g^y)^{-x} \\ &= m \cdot g^{xy} g^{-xy} \\ &= m \end{aligned}$$

El-Gamal Encryption

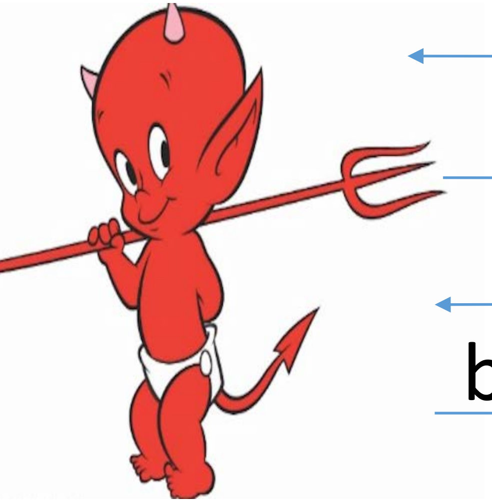
- $\mathrm{Enc}_{\mathrm{pk}}(m) = \langle g^y, m \cdot h^y \rangle$ for a random y $\in \mathbb{Z}_q$
- $\mathrm{Dec}_{\mathrm{sk}}(c = (c_1, c_2)) = c_2 c_1^{-x}$

Theorem 11.18: Let $\Pi = (Gen, Enc, Dec)$ be the El-Gamal Encryption scheme (above) then if DDH is hard relative to $\mathcal{G}$ then Π is CPA-Secure.

Proof: Recall that CPA-security and eavesdropping security are equivalent for public key crypto. Therefore, it suffices to show that for all PPT A there is a negligible function **negl** such that

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

Eavesdropping Security ($\mathrm{PubK}_{A,\Pi}^{\mathrm{eav}}(\mathrm{n})$)



Public Key: pk

$m_0\ , m_1$

$\boldsymbol{c_1 = \mathrm{Enc}_{pk}(m_b\ )}$

b’

Random bit b

(pk,sk) = Gen(.)

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

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

El-Gamal Encryption

Theorem 11.18: Let $\Pi = (Gen, Enc, Dec)$ be the El-Gamal Encryption scheme (above) then if DDH is hard relative to $\mathcal{G}$ then Π is CPA-Secure.

Proof: First introduce an `encryption scheme' $\widetilde{\Pi}$ in which $\widetilde{\mathrm{Enc}_{\mathrm{pk}}}(m) = \langle g^y, m \cdot g^z \rangle$ for random $\mathrm{y}, \mathrm{z} \in \mathbb{Z}_q$ (there is actually no way to do decryption, but the experiment $\mathrm{PubK}^{\mathrm{eav}}_{\mathrm{A},\widetilde{\Pi}}(\mathrm{n})$ is still well defined).

Claim: $\Pr\left[\mathrm{PubK}^{\mathrm{eav}}_{\mathrm{A},\widetilde{\Pi}}(\mathrm{n}) = 1\right] = \frac{1}{2}$

El-Gamal Encryption

Claim: $\Pr\left[\mathrm{PubK}^{\mathrm{eav}}_{\mathrm{A},\widetilde{\Pi}}(\mathrm{n}) = 1\right] = \frac{1}{2}$

Proof: (using Lemma 11.15)

$$\Pr\left[\mathrm{PubK}^{\mathrm{eav}}_{\mathrm{A},\widetilde{\Pi}}(\mathrm{n}) = 1\right]$$
$$= \frac{1}{2}\Pr\left[\mathrm{PubK}^{\mathrm{eav}}_{\mathrm{A},\widetilde{\Pi}}(\mathrm{n}) = 1 | b = 1\right] + \frac{1}{2}\left(1 - \Pr\left[\mathrm{PubK}^{\mathrm{eav}}_{\mathrm{A},\widetilde{\Pi}}(\mathrm{n}) = 0 | b = 0\right]\right)$$
$$= \frac{1}{2} + \frac{1}{2}\left(\Pr_{\mathrm{y,z}\leftarrow\mathbb{Z}_q}[A(\langle g^y, m_1 \cdot g^z\rangle) = 1] - \Pr_{\mathrm{y,z}\leftarrow\mathbb{Z}_q}[A(\langle g^y, m_0 \cdot g^z\rangle) = 1]\right)$$
$$= \frac{1}{2}$$

El-Gamal Encryption

Theorem 11.18: Let $\Pi = (Gen, Enc, Dec)$ be the El-Gamal Encryption scheme (above) then if DDH is hard relative to $\mathcal{G}$ then Π is CPA-Secure.

Proof: We just showed that

$$\Pr\left[\mathrm{PubK}^{\mathrm{eav}}_{\mathrm{A},\widetilde{\Pi}}(\mathrm{n}) = 1\right] = \frac{1}{2}$$

Therefore, it suffices to show that

$$\left|\Pr\left[\mathrm{PubK}^{\mathrm{eav}}_{\mathrm{A},\Pi}(\mathrm{n}) = 1\right] - \Pr\left[\mathrm{PubK}^{\mathrm{eav}}_{\mathrm{A},\widetilde{\Pi}}(\mathrm{n}) = 1\right]\right| \leq \mathbf{negl}(n)$$

This, will follow from DDH assumption.

El-Gamal Encryption

Theorem 11.18: Let $\Pi = (Gen, Enc, Dec)$ be the El-Gamal Encryption scheme (above) then if DDH is hard relative to $\mathcal{G}$ then Π is CPA-Secure.

Proof: We can build $B(g^x, g^y, Z)$ to break DDH assumption if Π is not CPA-Secure. Simulate eavesdropping attacker A

1. Send attacker public key $\mathrm{pk} = \langle \mathbb{G}, q, g, h = g^x \rangle$
2. Receive m_0,m_1 from A.
3. Send A the ciphertext $\langle g^y, m_b \cdot Z \rangle$.
4. Output 1 if and only if attacker outputs b'=b; otherwise output 0.

$$\left|\Pr\left[B(g^x, g^y, Z) = 1 \middle| Z = g^{xy}\right] - \Pr\left[B(g^x, g^y, Z) = 1 \middle| Z = g^z\right]\right|$$
$$= \left|\Pr\left[\mathrm{PubK}_{\mathrm{A},\Pi}^{\mathrm{eav}}(\mathrm{n}) = 1\right] - \Pr\left[\mathrm{PubK}_{\mathrm{A},\widetilde{\Pi}}^{\mathrm{eav}}(\mathrm{n}) = 1\right]\right|$$
$$= \left|\Pr\left[\mathrm{PubK}_{\mathrm{A},\Pi}^{\mathrm{eav}}(\mathrm{n}) = 1\right] - {}^{1}/_{2}\right|$$

El-Gamal Encryption

- $\mathrm{Enc}_{\mathrm{pk}}(m) = \langle g^y, m \cdot h^y \rangle$ for a random y $\in \mathbb{Z}_q$ and $h = g^x$,
- $\mathrm{Dec}_{\mathrm{sk}}(c = (c_1, c_2)) = c_2 c_1^{-x}$

Fact: El-Gamal Encryption is malleable.

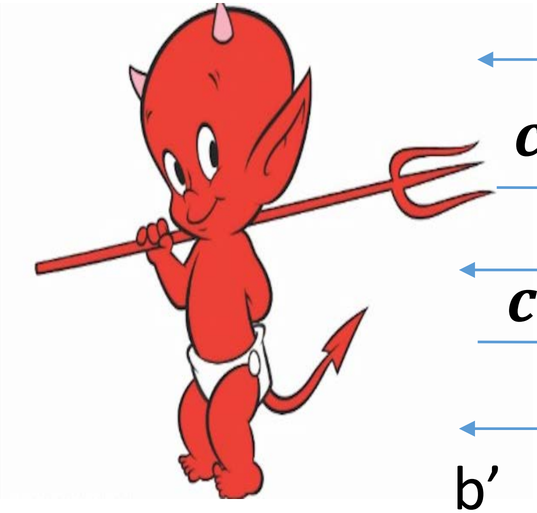
$$\mathrm{c} = \mathrm{Enc}_{\mathrm{pk}}(m) = \langle g^y, m \cdot h^y \rangle$$
$$c' = \langle g^y, 2 \cdot m \cdot h^y \rangle$$
$$\mathrm{Dec}_{\mathrm{sk}}(c') = 2 \cdot m \cdot h^y \cdot g^{-xy} = 2m$$

Hint: This observation may be relevant for homework 4.

Key Encapsulation Mechanism (KEM)

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

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



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

⋮

$\boldsymbol{c_1 \neq c}$

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

$\boldsymbol{c_2 \neq c}$

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

b'

⋮

Random bit b

(pk,sk) = Gen(.)

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

$\boldsymbol{k_1 \longleftarrow \{0,1\}^n}$

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

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

KEM from RSA and El-Gamal

- Recap: CCA-Secure KEM from RSA in Random Oracle Model

- El-Gamal yields CPA-Secure KEM in Random Oracle Model
  - $\left(\boldsymbol{g}^{\boldsymbol{y}}, \boldsymbol{H}(\boldsymbol{h}^{\boldsymbol{y}})\right) \leftarrow \mathbf{Encaps}_{\mathbf{pk}}(\mathbf{1}^{\boldsymbol{n}}; \boldsymbol{R})$ and $\mathbf{Decaps}_{\mathbf{sk}}(\boldsymbol{g}^{\boldsymbol{y}}) = \boldsymbol{H}(\boldsymbol{g}^{\boldsymbol{xy}})$
  - **CDH** assumption must hold.
- Above construction is also a CPA-Secure KEM in standard model
  - As long as $Pr_{x\in\mathbb{G}}[H(x) = k] \approx 2^{-\ell}$ for each key $k \in \{0,1\}^{\ell}$ and ***DDH*** holds
  - **Disadvantage:** weaker security notion for KEM, stronger DDH assumption
  - **Advantage:** Proof in standard model

CCA-Secure Variant in Random Oracle Model

- Key Generation ($\mathrm{Gen}(1^n)$):
  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. Choose a random $\mathrm{x} \in \mathbb{Z}_q$ and set $h = g^x$
  3. Public Key: $\mathrm{pk} = \langle \mathbb{G}, q, g, h \rangle$
  4. Private Key: $\mathrm{sk} = \langle \mathbb{G}, q, g, x \rangle$
- $\mathrm{Enc}_{\mathrm{pk}}(m) = \left\langle g^y, c', Mac_{K_M}(c') \right\rangle$ for a random $\mathrm{y} \in \mathbb{Z}_q$ where

$$K_E \| K_M = H(h^y) \quad (\mathrm{KEM})$$

and

$$c' = \mathrm{Enc}'_{\mathrm{K_E}}(m) \quad (\text{Encrypt then MAC})$$

CCA-Secure Variant in Random Oracle Model

Public Key: $\mathrm{pk} = \langle \mathbb{G}, q, g, h \rangle$
Private Key: $\mathrm{sk} = \langle \mathbb{G}, q, g, x \rangle$

- $\mathrm{Enc}_{\mathrm{pk}}(m) = \left\langle g^y, c', Mac_{K_M}(c') \right\rangle$ for a random y $\in \mathbb{Z}_q$ and $K_E \| K_M = H(h^y)$ and $c' = \mathrm{Enc}'_{\mathrm{K_E}}(m)$
- $\mathrm{Dec}_{\mathrm{sk}}(\langle c, c', t \rangle)$

1. $K_E \| K_M = H(c^x)$

2. If $\mathrm{Vrfy}_{\mathrm{K_M}}(c', t) \neq 1$ or $c \notin \mathbb{G}$ output $\perp$; otherwise output $\mathrm{Dec}'_{\mathrm{K_E}}(c')$

CCA-Secure Variant in Random Oracle Model

Theorem: If $\mathrm{Enc}'_{\mathrm{K_E}}$ is CPA-secure, $\mathrm{Mac}_{\mathrm{K_M}}$ is a strong MAC and a problem called gap-CDH is hard then this a CCA-secure public key encryption scheme in the random oracle model.

- $\mathrm{Enc}_{\mathrm{pk}}(m) = \langle g^y, c', \mathrm{Mac}_{\mathrm{K_M}}(c') \rangle$ for a random y $\in \mathbb{Z}_q$ and $K_E \| K_M = H(h^y)$ and $c' = \mathrm{Enc}'_{\mathrm{K_E}}(m)$
- $\mathrm{Dec}_{\mathrm{sk}}(\langle c, c', t \rangle)$

1. $K_E \| K_M = H(c^x)$
2. If $\mathrm{Vrfy}_{\mathrm{K_M}}(c', t) \neq 1$ or $c \notin \mathbb{G}$ output $\perp$; otherwise output $\mathrm{Dec}'_{\mathrm{K_E}}(c')$

CCA-Secure Variant in Random Oracle Model

Remark: The CCA-Secure variant is used in practice in the ISO/IEC 18033-2 standard for public-key encryption.

- Diffie-Hellman Integrated Encryption Scheme (DHIES)
- Elliptic Curve Integrated Encryption Scheme (ECIES)
- $\mathrm{Enc}_{\mathrm{pk}}(m) = \langle g^y, c', \mathrm{Mac}_{\mathrm{K_M}}(c') \rangle$ for a random y $\in \mathbb{Z}_q$ and $K_E \| K_M = H(h^y)$ and $c' = \mathrm{Enc}'_{\mathrm{K_E}}(m)$
- $\mathrm{Dec}_{\mathrm{sk}}(\langle c, c', t \rangle)$

1. $K_E \| K_M = H(c^x)$
2. If $\mathrm{Vrfy}_{\mathrm{K_M}}(c', t) \neq 1$ or $c \notin \mathbb{G}$ output $\perp$; otherwise output $\mathrm{Dec}'_{\mathrm{K_E}}(c')$