Computer Science 308-547A
Cryptography and Data Security

Claude Crépeau

These notes are, largely, transcriptions by Anton Stiglic of class notes from the former course *Cryptography and Data Security (308-647A)* that was given by prof. Claude Crépeau at McGill University during the autumn of 1998-1999. These notes are updated and revised by Claude Crépeau.

11 Discrete Logarithm Problems and the Diffie-Hellman Key Exchange

11.1 Discrete Logarithm Problems

Definition 11.1 *(Generalized Discrete Logarithm Problem (GDLP)) Given a group G of order n, α a generator of G and $\beta \in G$, find e such that $\alpha^e \equiv \beta$, $0 \leq e \leq n-1$.*

Definition 11.2 *(Discrete Logarithm Problem (DLP)) DLP is GDLP but in a group of order p where p is prime.*

An even greater generalization is to consider GDLP but with G not necessarily cyclic (α not necessarily a generator) and to find e if such an integer exists. This problem is considered to be harder than GDLP.

11.2 Diffie-Hellman key exchange

In a landmark paper [**?**], W. Diffie and M.E. Hellman introduced an algorithm for exchanging a symmetric key in public. They also proposed the existence of public symmetric encryption, authentication and digital signature schemes. In the Diffie-Hellman key exchange scheme, *Alice* and *Bob* both share a large public prime p and a primitive element g of $\mathbb{Z}_p$. *Alice*, (and independently *Bob*) picks a random integer $x \in_R \mathbb{Z}_p$ (*Bob* picks $y \in_R \mathbb{Z}_p$) and computes $u = g^x \ mod \ p$ (*Bob* computes $v = g^y \ mod \ p$). *Alice* then sends u to *Bob*, via a public channel, and *Bob* sends his value v to *Alice*. *Alice*s private key is computed as $K_{Alice} = v^x \bmod p$ ($K_{Bob} = u^y \bmod p$).
We easily see that $K_{Alice} = K_{Bob}$, this is their symmetric key.

11.2.1 Cryptanalysis: Diffie-Hellman problem

Oscar's job is to compute K given g, p, u and v, which gives rise to the following problem

Definition 11.3 *(Diffie-Hellman Problem DHP) Given a prime p, a generator $\alpha \in \mathbb{Z}_p$, $\alpha^x \bmod p$ and $\alpha^y \bmod p$, find $\alpha^{xy} \bmod p$.*

Diffie-Hellman assumption: no efficient algorithm can solve the above problem.
This assumption is at least as strong as the DLP (DHP $\leq_P$ DLP), for if *Oscar* could easily compute x and y from α^x and α^y, he can then easily compute $K = (\alpha^x)^y$. The Generalized DHP (GDHP) is like DHP but working on a finite cyclic group G (instead of just $\mathbb{Z}_p{}^*$).
GDHP is also at least as strong as GDLP. It is still unknown if GDLP is as strong as GDHP, however some equivalence (for some specific classes of groups) are known (see [**?**] and [**?**]).

12 Public Key Cryptography: RSA

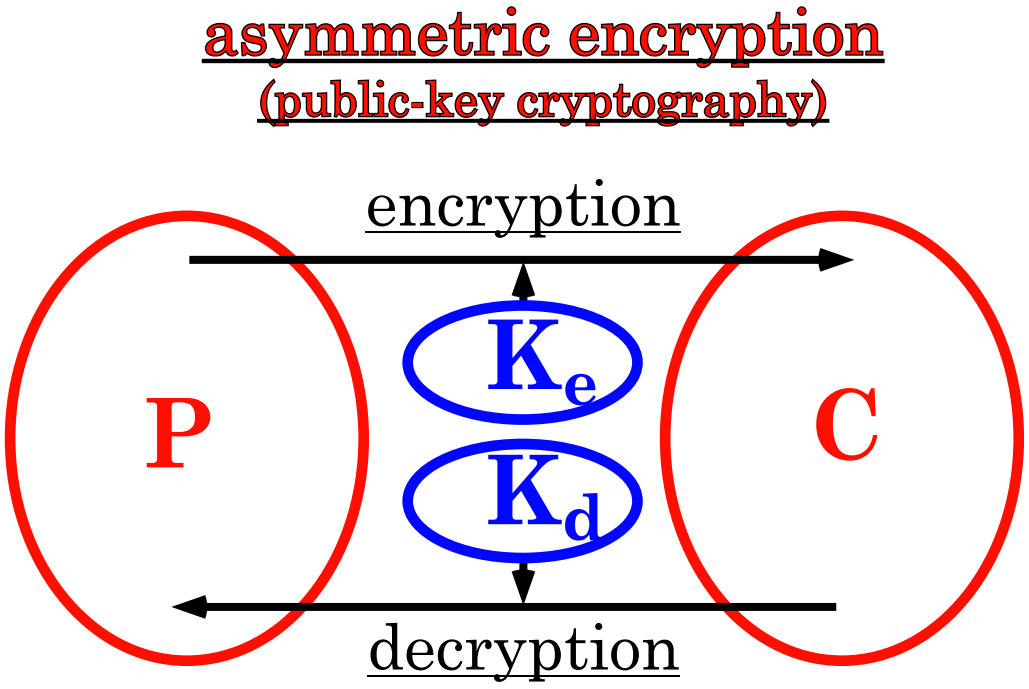
R.L. Rivest, A. Shamir and L.M. Adleman introduced, in a landmark paper [**?**], the first[1] valid public key cryptographic system, it is named RSA. The security of RSA lies in the famous factoring problem (breaking RSA is not known to be equivalent to factoring, but if factoring was easy, so would breaking RSA). We will elaborate more on the security of RSA in section 12.3.

Formally, a Public Key Cryptosystem is defined as follows:

Definition 12.1 *Let $\mathcal{P}$ be a finite set of messages and $\mathcal{C}$ a finite set of ciphertexts such that for each $(k_e, k_d) \in \mathcal{K}$, there is an encryption algorithm E_{k_e} and a corresponding decryption algorithm D_{k_d} such that $E_{k_e} : \mathcal{P} \to \mathcal{C}$ and $D_{k_d} : C \to \mathcal{P}$ are polynomial-time computable functions and for all $m \in \mathcal{P}$*

$$D_{k_d}(E_{k_e}(m)) = m.$$

The problem of finding m given only $E_{k_e}(m)$ and k_e must be difficult to solve.



[1] Merkle and Hellman [**?**] had also proposed a public key system, but it can be broken in polynomial time ([**?**]).

12.1 Factoring Problem

Definition 12.2 *(FACTORING) Given n, find prime integers $p_1, p_2, \ldots, p_k$ such that $n = p_1 \cdot p_2 \ldots p_k$.*

A special "hard case" is when $n = p \cdot q$.
The best known solutions have time of the form

$$\Omega(e^{c|n|^{\beta}(lg|n|)^{1-\beta}})$$

12.2 RSA public-key cryptosystem

Algorithm 12.1 (RSA key generation)

1: *Pick two large primes p and q, $n \leftarrow p \cdot q$.*

2: *Pick e such that $gcd(e, \phi(n)) = 1$.*

3: *Compute d such that $e \cdot d \equiv 1 \pmod{\phi(n)}$.*

4: RETURN $K_e = (n, e)$, $K_d = (n, d)$.

K_e is the encryption (public) key, K_d is the decryption (private) key.
RSA-Encryption: $E_{K_e}(m) = m^e \ mod \ n$.
RSA-Decryption: $D_{K_d}(c) = c^d \ mod \ n$.
We now show that decryption is the inverse of encryption. Let $x \in \mathbb{Z}_n^*$, then

$$\begin{aligned}(x^e)^d &\equiv x^{t\phi(n)+1} \pmod{n}, \text{for some } t. \\ &\equiv (x^{\phi(n)})^t x \pmod{n} \\ &\equiv 1^t x \pmod{n} \\ &\equiv x \pmod{n}\end{aligned}$$

12.3 Cryptanalysis of RSA

Theorem 12.3 *Knowledge of $\phi(n)$ is sufficient to factorize n.*

Proof. If $n = pq$, then

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

$$\begin{aligned} &= pq - p - q + 1 \\ &= n - p - q + 1 \end{aligned}$$

So

$$\begin{aligned} p &= n - \phi(n) - q + 1 \\ &= n - \phi(n) - n/p + 1 \\ &= (n - \phi(n) + 1) - n/p \end{aligned}$$

Which is the same as

$$p^2 = (n - \phi(n) + 1)p - n$$

or

$$p^2 - (n - \phi(n) + 1)p + n = 0.$$

p is one of the two solutions to this degree 2 equation.

12.3.1 Factoring given d, e

Theorem 12.4 *Knowledge of a square root modulo n extraction algorithm S is sufficient to factor n.*

Proof.

Algorithm 12.2 (Factoring n given S)

1: **REPEAT** *choose* $r \in_R \mathbb{Z}_n^*$ *at random*

2: $z \leftarrow r^2 \bmod n$

3: $y \leftarrow S(z)$

4: **UNTIL** $y \not\equiv \pm r \pmod{n}$.

5: **RETURN** $(y - r) \bmod n$, $(y + r) \bmod n$.

Theorem 12.5 *Knowledge of d corresponding to (n, e) enables to factor n.*

Before we give the proof of the above theorem, we state a few facts:

- $\phi(n) = (p-1)(q-1)$ is an *even number* and so we can write

$$ed \equiv 1 \pmod{\phi(n)} \Rightarrow ed - 1 = 2^s r$$

  Although p and q might be hard to find, s and r are easily computable.

- If $n = pq$, then $x^2 \equiv 1 \pmod{p}$ has two solutions, namely $x \equiv \pm 1 \pmod{p}$. Same for the solutions modulo q. Since $x^2 \equiv 1 \pmod{n}$ iff $x^2 \equiv 1 \pmod{p}$ and $x^2 \equiv 1 \pmod{q}$, it follows that $x^2 \equiv 1 \pmod{n}$ iff $x \equiv \pm 1 \pmod{p}$ and $x \equiv \pm 1 \pmod{q}$. So there are 4 square roots of 1 modulo n, two of them are trivial: 1 and -1. The other two solutions are called *non-trivial*, and they are such that they are additive inverses of each other.

Proof. The following algorithm starts by picking a number w at random. If we are (very) lucky and $1 < gcd(w, n) < n$, then we have found a factor. If not, we continue on by computing $w^r, w^{2r}, w^{4r}, \ldots$ by successive squaring, until $w^{2^t r} \equiv 1 \pmod{n}$ for some t. This will take at most $t \leq \lg n$ tries. If, at the beginning, $w^r \equiv 1 \pmod{n}$ we quit because this is a trivial solution. Also, at the end, once we have found t such that $w^{2^t r} \equiv 1 \pmod{n}$, if $w^{2^{t-1} r} \not\equiv -1 \pmod{n}$ we also quit, because we ended up with a non-trivial square root, then we are able to factor n.

Algorithm 12.3 (Factoring n given e, d using w)

1: $x \leftarrow gcd(w, n)$.

2: **IF** $1 < x < n$ **THEN RETURN** *SUCCESS:* x.

3: *write* $ed - 1 = 2^s r$, r *odd,* $s > 0$; $v \leftarrow w^r \bmod n$.

4: **IF** $v \equiv 1 \pmod{n}$ **THEN RETURN** *FAILURE.*

5: **WHILE** $v \not\equiv 1 \pmod{n}$ **DO**

6: $v_0 \leftarrow v; v \leftarrow v^2 \bmod n$.

7: **IF** $v_0 \equiv -1 \pmod{n}$ **THEN RETURN** *FAILURE.*

8: **ELSE** $x \leftarrow gcd(v_0 + 1, n)$; **RETURN** *SUCCESS:* x.

Notice the similarity to the Rabin-Miller primality test.

Theorem 12.6

$$\#\{w|\textit{Factoring } n \textit{ given } e, d \textit{ using } w \textit{ succeds}\} > n/2.$$

Indeed this algorithm will also work with identical probability for any multiple of $\phi(n)$ instead of $ed-1$ because the success of an element w only depends on its order.

Theorem 12.7

$$\textit{if } d < n^{1/4}/3 \textit{ then } d \textit{ is esay to recover from } (n, e).$$

12.3.2 Homomorphic property of RSA

Theorem 12.8 *(RSA is multiplicative)*

$$RSA_e(m_1) \cdot RSA_e(m_2) \equiv RSA_e(m_1 \cdot m_2) \pmod{n}.$$

[**?**] showed how to use the multiplicative property of RSA to come up with the following result:

Theorem 12.9 *(Self random reducible) Any algorithm A that can break RSA on a $\epsilon = \frac{1}{poly(|n|)}$ fraction of the instances may be used to invert RSA on all instances.*

Proof. Consider the following algorithm:

Algorithm 12.4 (INVRSA$(n, e, \underbrace{m^e \bmod n}_{c})$ **)**

1: REPEAT:

2: *Pick a random* $\overline{m} \in_R \mathbb{Z}_n^*$.

3: $w \leftarrow c \cdot RSA_e(\overline{m}) \bmod n \ \ (= RSA_e(m \cdot \overline{m}))$.

4: $z \leftarrow A(w)$. *(if A is correct, z is $m \cdot \overline{m}$).*

5: UNTIL $RSA_e(z \cdot (\overline{m})^{-1}) \equiv c \pmod{n}$.

6: RETURN $(z \cdot (\overline{m})^{-1}) \bmod n$.

In step 3, since $\overline{m}$ is picked at random, w is also. Since A will find a correct answer for a $\frac{1}{poly(|n|)}$ fraction of the $\phi(n)$ elements of $\mathbb{Z}_n$, the algorithm will run in an expected time of $poly(|n|)$ **REPEAT** loops.

12.4 Security of certain bits in RSA

Given $y = RSA_{(e,N)}(x)$, it is possible to gain some information on x. For example, given y, it is possible to compute the Jacobi symbol $\left(\frac{x}{N}\right)$ of x. Thankfully (for the RSA scheme), computing certain bits of y is computationally equivalent to computing x (inverting RSA). This is true, for example, for the least significant bit of y.

Theorem 12.10 *Given $y = RSA_e(x)$, one can easily compute $\left(\frac{x}{N}\right)$.*

Proof. The RSA exponent e is always odd, so

$$\left(\frac{x}{N}\right)^e = \left(\frac{x}{N}\right).$$

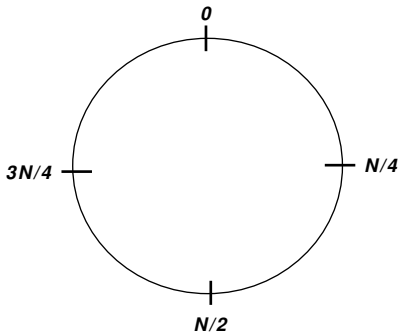
Knowing that $\left(\frac{x \cdot x}{N}\right) = \left(\frac{x}{N}\right)\left(\frac{x}{N}\right)$ (section 1.4), we can deduce that $\left(\frac{x^e}{N}\right) = \left(\frac{x}{N}\right)^e$. We thus easily conclude that

$$\left(\frac{x}{N}\right) = \left(\frac{y}{N}\right)$$

.

12.4.1 Security of RSA least significant bit

Let N be the RSA modulus and $n = |N|$. $lsb_N(m)$ will denote the least significant bit of m. For the following subsections, it will be convenient to view the elements of $\mathbb{Z}_N$ as points on a circle:



Cyclic representation of Z_n

Theorem 12.11 *([?]) If we have an algorithm to determine $lsb_N(m)$ given $\underbrace{RSA_e(m)}_{m^e \bmod N}$, then we can compute m.*

Proof.

We first define the following function

$$half_N(m) = \begin{cases} 1 & : \quad \frac{N}{2} < m < N \\ 0 & : \quad 0 < m < \frac{N}{2} \end{cases}$$

Notice that we have

$$half_N(m) = lsb_N(2 * m)$$

This can be seen by the fact that if $N/2 < m < N$ then if $N < 2m < 2N$, $2m - N \equiv 2m \pmod N$ is odd. If $0 \leq m \leq N/2$, then $2m < N$ and $2m \bmod N$ is necessarily even. So we can use $half_M$ instead of lsb_M.
Now observe that

$$\begin{aligned} half_N(m) = 0 &\iff x \in [0, \frac{N}{2}[\\ half_N(2m) = 0 &\iff x \in [0, \frac{N}{4}[\cup [\frac{N}{2}, \frac{3N}{4}[\\ half_N(4m) = 0 &\iff x \in [0, \frac{N}{8}[\cup [\frac{N}{4}, \frac{3N}{8}[\cup [\frac{N}{2}, \frac{5N}{8}[\cup [\frac{3N}{4}, \frac{7N}{8}[\\ &\cdots \end{aligned}$$

Thus giving us the following algorithm:

Algorithm 12.5 (INVRSA$(n, e, \underbrace{m^e \bmod n}_{c})$ **given** LSB **)**

1: **FOR** $i = 0$ **TO** $\lg N$ **DO**

2: $c \leftarrow c \cdot RSA(2) \bmod N$.

3: $y_i \leftarrow LSB(c, e, N)$ $(= half_N(2^i m))$.

4: $low \leftarrow 0$.

5: $high \leftarrow N$.

6: **FOR** $i = 0$ **TO** k **DO**

7: $mid = \lfloor (high + low + 1)/2 \rfloor$.

8: **IF** $y_i = 1$ **THEN** $low = mid$ **ELSE** $high = mid - 1$.

9: **RETURN** $high$

where we use a speculative algorithm LSB that is defined such that $LSB(m^e \ mod \ N, e, N)$ gives $lsb_N(m)$.

The algorithm LSB used in the above proof had to be error free. That is, we had to be able to predict $lsb_N(m)$ exactly. There is a variant of this protocol that can use an oracle that is correct with probability $1 - \frac{\alpha}{n}$ ($\alpha < 1$ a constant), but we can invert RSA with much lesser assumptions.

Theorem 12.12 *([?]) Predicting $lsb_N(m)$ with probability $> 1/2 + \epsilon$, $1/\epsilon < poly(|N|)$, $\Rightarrow$ inverting RSA.*

The proof of this very important result is outside the scope of these notes.

13 Rabin Public-Key scheme

Rabin's cryptosystem ([**?**]) was the first example of a provably secure encryption scheme, it is equivalent to the FACTORING problem.

Algorithm 13.1 (Rabin key generation)

1: *Pick two large prime numbers* p, q

2: $n \leftarrow pq$

3: **RETURN** $K_e = n$, $K_d = (p, q)$

Encryption: $c = m^2 \bmod n$.
Decryption: Find the four[2] square roots m_1, m_2, m_3, m_4, and pick the one that makes sense.

13.1 Cryptanalysis

The task of recovering m from c, not knowing the factorization of n, is exactly the SQROOT problem (1.8) which is polynomialy equivalent to the FACTORING problem (12.1).

13.1.1 Chosen-ciphertext attack

Rabin's scheme is vulnerable to a chosen-ciphertext attack. The attack goes as follows: *Oscar* picks a random $m \in_R \mathbb{Z}_n^*$ and computes $c = m^2 \bmod n$. He then gives c to the decryption box, which in turn will give him y that is not necessarily equal to m. With probability $\frac{1}{2}$, $y \not\equiv \pm m \pmod{n}$, in which case $gcd(m - y, n)$ is a prime factor of n. If $m \equiv \pm y \pmod{n}$, then *Oscar* just picks another m.

13.2 Public-key Cryptography from PRBG

In order to take advantage of this theory and build a semantically secure public-key cryptosystem, Blum-Goldwasser had the idea of combining the Blum-Blum-Shub generator with the public-key set up.

[2]In the unlikely case where $gcd(m, n) \neq 1$, the ciphertext c has only one or two square roots.

Algorithm 13.2 (BG/E(m))

1: *Pick a random* $x \in QR_n$

2: **FOR** $i \leftarrow 1$ **TO** $|m|/t$ **DO** $k_i \leftarrow x \bmod 2^t$ *;* $x \leftarrow x^2 \bmod n$

3: $k \leftarrow k_1||k_2||\ldots||k_{|m|/t}$

4: **RETURN** $\langle (m \oplus k), x \rangle$

The parameter t indicates how many bits are extracted from each iteration of the generator. It is known that $t \in O(\log|n|)$ bits of $x^2 \bmod n$ (and of RSA) are simultaneously secure. What makes this cryptosystem reasonably efficient is that instead of recovering the message m by successive square root extractions, which would be costly, a single root extraction of higher order is applied to recover the start point of the sequence which is then calculated in the forward direction.

Algorithm 13.3 (BG/D(c, y))

1: $x \leftarrow \sqrt[2^{\frac{|c|}{t}}]{y} \bmod n$

2: **FOR** $i \leftarrow 1$ **TO** $|m|/t$ **DO** $k_i \leftarrow x \bmod 2^t$ *;* $x \leftarrow x^2 \bmod n$

3: $k \leftarrow k_1||k_2||\ldots||k_{|c|/t}$

4: **RETURN** $c \oplus k$

The $\sqrt[2^\ell]{x} \bmod n$ operation can be computed efficiently using the following algorithm. Notice that for a fixed ℓ the first two steps can be pre-computed once and for all. Alternatively, for a standard set of sizes the first step can be pre-computed for all of them and a table of the standard values can be pre-computed for step 2.

Algorithm 13.4 ($\sqrt[2^\ell]{x} \bmod n$)

1: *find* a, b *such that* $ap + bq = 1$

2: $i \leftarrow [(p+1)/4]^\ell \bmod p-1$ *;* $j \leftarrow [(q+1)/4]^\ell \bmod q-1$

3: **RETURN** $(x^j \bmod q)ap + (x^i \bmod p)bq \bmod n$

A similar cryptosystem roughly as efficient can be implemented from RSA with exponent 3 instead. The same remarks as for the pre-computation involved in $\sqrt[2^\ell]{x} \bmod n$ apply here as well.

Algorithm 13.5 (RSA/E(m))

1: *Pick a random* $x \in Z_n^*$

2: **FOR** $i \leftarrow 1$ **TO** $|m|/t$ **DO** $k_i \leftarrow x \bmod 2^t$ *;* $x \leftarrow x^3 \bmod n$

3: $k \leftarrow k_1||k_2||\ldots||k_{|m|/t}$

4: **RETURN** $\langle (m \oplus k), x \rangle$

Algorithm 13.6 (RSA/D(c, y))

1: $x \leftarrow \sqrt[\frac{|c|}{t}]{3}{y} \bmod n$

2: **FOR** $i \leftarrow 1$ **TO** $|m|/t$ **DO** $k_i \leftarrow x \bmod 2^t$ *;* $x \leftarrow x^3 \bmod n$

3: $k \leftarrow k_1||k_2||\ldots||k_{|c|/t}$

4: **RETURN** $c \oplus k$

Algorithm 13.7 ($\sqrt[3^\ell]{x} \bmod n$)

1: *find* d *such that* $3d \equiv 1 \pmod{\phi(n)}$

2: $i \leftarrow d^\ell \bmod \phi(n)$

3: **RETURN** $x^i \bmod n$

14 El Gamal

This scheme was invented by El Gamal [**?**]. Its security is based on the DH and DL problems.

Algorithm 14.1 (El Gamal key generation)

1: *Pick a large random prime* p.

2: *Pick a generator* α *of* $\mathbb{Z}_p^*$ *and a random integer* a, $1 \leq a \leq p-2$.

3: $\beta \leftarrow \alpha^a \bmod p$.

4: RETURN $K_e = (p, \alpha, \beta)$ *and* $K_d = a$.

Algorithm 14.2 (El Gamal public-key encryption)

1: *Pick a random integer* k, $1 \leq k \leq p-2$.

2: $\gamma \leftarrow \alpha^k \bmod p$, $\delta \leftarrow m \cdot \beta^k \bmod p$.

3: RETURN$c = (\gamma, \delta)$.

Decryption: $m \leftarrow \delta(\gamma)^{-a} \bmod p$.
Proof that $d(e(x)) = x$:

$$\begin{aligned} d(\gamma, \delta) &\equiv \delta \cdot (\gamma)^{-a} \bmod p \\ &\equiv m \cdot \beta^k (\alpha^k)^{-a} \bmod p \\ &\equiv m \cdot \beta^k (\beta)^{-k} \bmod p \\ &\equiv m \bmod p. \end{aligned}$$

Theorem 14.1 *Random integers* k *must be used if no information about the cleartexts is to be revealed.*

Proof. Say we have

$$\begin{aligned} e(m_1, k) &= (\alpha^k \bmod p, \underbrace{m_1 \cdot \beta^k \bmod p}_{\delta_1}) \text{ and} \\ e(m_2, k) &= (\alpha^k \bmod p, \underbrace{m_2 \cdot \beta^k \bmod p}_{\delta_2}) \end{aligned}$$

we can then compute $\delta_1/\delta_2 = m_1/m_2 \bmod p$.

Theorem 14.2 *The security of the El Gamal System is based on the DH problem.*

Proof. Denote $O_{ElGamal}$ to be an oracle for decrypting El Gamal encrypted messages, given $p, \alpha, \alpha^a, \alpha^k, m \cdot \alpha^{ak}$ and O_{DH} an oracle for solving the DH (that is $DH(p, \alpha, \alpha^a, \alpha^b)$ gives α^{ab}).

- ($O_{ElGamal}$ from O_{DH}) Compute α^{ak} using O_{DH}. We then have $m \leftarrow (\alpha^{ak})^{-1} \cdot \delta$.

- (O_{DH} from $O_{ElGamal}$) Pick a random $\delta \in \mathbb{Z}_p$. Compute $O_{ElGamal}(p, \alpha, \alpha^a, \alpha^b, \delta)$, which gives m such that $m \cdot \alpha^{ab} = \delta$. We have $\alpha^{ab} \leftarrow \delta \cdot m^{-1} \bmod p$.

14.1 Generalizing El Gamal

We described the El Gamal system in a group $\mathbb{Z}_p^*$, but it can be generalized to work in any finite cyclic group G. It's security is then *based* on the DL problem of that particular group. Examples of groups in which the DL problem is believed to be hard and in which operations can be efficiently executed are

$$\mathbb{Z}_p^*, \mathcal{F}_{2^m}^*, \mathcal{F}_{q^m}^*, \mathbb{Z}_{pq}^*$$

.

Theorem 14.3 *In the case of $\mathbb{Z}_N^*$ where $N = pq$ and $p \equiv q \equiv 3$ (mod 4), the DL problem is as hard as FACTORING.*

Note: for the case of $\mathbb{Z}_N^*$ where $N = pq$, it is recommended to pick p and q in such a way that $p-1$ and $q-1$ do not have small factors, so as to guard against *Pollard's* factoring algorithm.