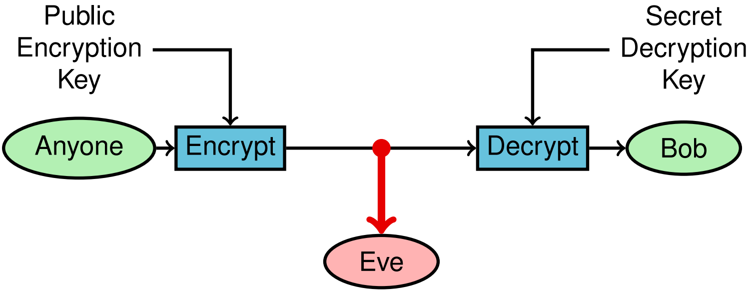
Cryptography Lecture 7

RSA continued, Knapsack, Diffie-Hellman, ElGamal

LINKÖPING UNIVERSITY

Public key cryptography

Asymmetric key systems can be used in public key cryptography



Trapdoor one-way functions

- A trapdoor one-way function is a function that is easy to compute but computationally hard to reverse
  - Easy to calculate $f(x)$ from x
  - Hard to invert: to calculate x from $f(x)$
- A trapdoor one-way function has one more property, that with certain knowledge it *is* easy to invert, to calculate x from $f(x)$
- There is no proof that trapdoor one-way functions exist, or even real evidence that they can be constructed. Examples:
- A few examples will follow (anyway)

Trapdoor one-way function candidate: exponentiation modulo $n = pq$

A trapdoor one-way function is a function that is easy to compute but computationally hard to reverse

- Easy to calculate (x^e mod n) from x
- Hard to invert: to calculate x from (x^e mod n)?

The trapdoor is that with another exponent d it *is* easy to invert, to calculate $x = (x^e \bmod n)^d \bmod n$

We have shown (using the Chinese remainder theorem) that solving $x^2 = c \bmod pq$ is equally hard as factoring $n = pq$.

Choose p and q: Test for primality

Theorem (Fermat's little theorem): If n is prime and $a \neq 0 \bmod n$, then $a^{n-1} = 1 \bmod n$

Fermat primality test: Take a random $a \neq 0, \pm 1 \bmod n$.
If $a^{n-1} \neq 1$, then n is composite, otherwise n is prime with high probability

Choose p and q: Test for primality

Miller-Rabin primality test: To test n, take a random $a \neq 0, \pm 1 \bmod n$, and write $n - 1 = 2^k m$ with m odd

- Let $b_0 = a^m$, if this is ± 1 then stop: n is probably prime (because $a^{n-1} = 1$, remember the Fermat primality test)
- Let $b_{j+1} = b_j^2$, if this is $+1$ then stop: n is composite,
  (because $b_j \neq \pm 1$, so we can factor n)
  if this is -1 then stop: n is probably prime
  (because $a^{n-1} = 1$, Fermat again)
- Repeat. If you reach b_k then n is composite
  (if $b_k = +1$ remember that $b_{k-1} \neq \pm 1$ so we can factor n,
  otherwise $b_k = a^{n-1} \neq 1$, remember the Fermat primality test)

Choose p and q: Avoid simple factorization

- The **Fermat factorization method** uses $n = x^2 - y^2 = (x + y)(x - y)$
- Calculate $n + 1^2$, $n + 2^2$, $n + 3^2$, $n + 4^2$, $n + 5^2$, ..., until we reach a square, then we are done.

  Example:
  $295927 + 3^2 = 295936 = 544^2$, so $295927 = 541 \cdot 547$
- This is unlikely to be a problem for a many-digit $n = pq$, but usually p and q are chosen to be of slightly different size, to be on the safe side

Choose p and q: Avoid simple factorization

The **Pollard** $p-1$ **factorization** method uses $b = a^{B!} \bmod n$ for chosen a and B. Calculate $d = \gcd(b-1, n)$. If d is not 1 or n, we have factored n.

This works if one prime factor p of n is such that $p-1$ has only small factors. If B is big enough, $B! = k(p-1)$, and $b = a^{B!} = 1 \bmod p$ Then, $b-1$ contains a factor p, as does n.

Solution: choose p and q so that $p-1$ and $q-1$ has at least one large prime factor

Rivest Shamir Adleman (1977)

- Bob chooses secret primes p and q, and sets $n = pq$
  - Choose primes p and q using, say, the Miller-Rabin test
  - Choose primes of slightly different size
  - Choose p and q so that $p - 1$ and $q - 1$ has at least one large prime factor
- Bob chooses e with $\gcd(e, \phi(n)) = 1$
- Bob computes d so that $de = 1 \bmod \phi(n)$
- Bob makes n and e public but keeps p, q and d secret
- Alice encrypts m as $c = m^e \bmod n$
- Bob decrypts c as $m = c^d \bmod n$

What about factoring?

Miller-Rabin primality test: To test n, take a random $a \neq 0, \pm 1 \bmod n$, and write $n-1 = 2^k m$ with m odd

- Let $b_0 = a^m$, if this is ± 1 then stop: n is probably prime (because $a^{n-1} = 1$, remember the Fermat primality test)
- Let $b_{j+1} = b_j^2$, if this is $+1$ then stop: n is composite,
  (because $b_j \neq \pm 1$, so we can factor n)
  if this is -1 then stop: n is probably prime
  (because $a^{n-1} = 1$, Fermat again)
- Repeat. If you reach b_k then n is composite
  (if $b_k = +1$ remember that $b_{k-1} \neq \pm 1$ so we can factor n,
  otherwise $b_k = a^{n-1} \neq 1$, remember the Fermat primality test)

What about factoring?

If n is not prime, assume you know r such that all $x \neq 0 \bmod n$ give $x^r = 1 \bmod n$ (in RSA, $r = ed - 1$)

Universal exponent factoring: Take a random a with $1 < a < n - 1$, and write $r = 2^k m$ with m odd

- Let $b_0 = a^m$, if this is ± 1 then stop and try another a
- Let $b_{j+1} = b_j^2$, if this is $+1$ then stop: n is composite, and $\gcd(b_j - 1, n)$ is a factor of n
if this is -1 then stop and try another a
- Repeat. If you reach $b_k = a^r = 1$ then n is composite and $\gcd(b_{k-1} - 1, n)$ is a factor of n

What about factoring?

- If n is not prime, assume you know r such that all $x \neq 0 \bmod n$ give $x^r = 1 \bmod n$ (in RSA, $r = ed - 1$)
- Then, Universal exponent factoring will work with high probability
- So if you know both e and d in an RSA system, then you can factor n efficiently

Trapdoor one-way function example: exponentiation modulo pq

This function is easy to compute but computationally hard to reverse, unless you have certain (secret) knowledge

- It is easy to calculate ($x^e \bmod n$) from x
- It is hard to invert: to calculate x from ($x^e \bmod pq$), equally hard as factoring $n = pq$
- It is easy to invert if you have the decryption exponent d (and then factoring of pq is easy too)

Factoring with the Quadratic Sieve

Theorem: Suppose there exist integers x and y with $x^2 = y^2 \bmod n$ but $x \neq \pm y \bmod n$. Then n is composite, and $\gcd(x - y, n)$ gives a nontrivial factor of n.

So find x and y that has the same square mod n

Method: take numbers that have squares that are small modulo n, and hope that these squares (mod n) combine together to a square.

Factoring with the Quadratic Sieve

Method: take numbers that have squares that are small modulo n, and hope that these squares (mod n) combine together to a square.

Example:

$$41^2 = 32 \bmod 1649,$$
$$43^2 = 200 \bmod 1649,$$

The numbers 32 and 200 are not square, but the product

$$32 \cdot 200 = 6400 = 80^2$$

and

$$41 \cdot 43 = 114 \bmod 1649$$
$$(41 \cdot 43)^2 = 114^2 = 80^2 \bmod 1649.$$

Finally, $\gcd(114 - 80, 1649) = \gcd(34, 1649) = 17$, so $1649 = 17 \cdot 97$

Factoring with the Quadratic Sieve

Method: take numbers that have squares that are small modulo n, and hope that these squares (mod n) combine together to a square.

Problem: finding the numbers. The book suggests trying

$$\sqrt{in} + j$$

rounded down, for small j and various i. This will work sometimes, but using more sophisticated methods will give you the “Quadratic sieve”, and eventually, the “Number field sieve”

Key length

Table 7.4: Security levels (symmetric equivalent)

Security (bits)	Protection	Comment
32	Real-time, individuals	Only auth. tag size
64	Very short-term, small org	Not for confidentiality in new systems
72	Short-term, medium org Medium-term, small org	
80	Very short-term, agencies Long-term, small org	Smallest general-purpose $<$ 4 years protection (E.g., use of 2-key 3DES, $<2^{40}$ plaintext/ciphertexts)
96	Legacy standard level	2-key 3DES restricted to 10^6 plaintext/ciphertexts, $\approx$ 10 years protection
112	Medium-term protection	$\approx$ 20 years protection (E.g., 3-key 3DES)
128	Long-term protection	Good, generic application-indep. Recommendation, $\approx$ 30 years
256	”Foreseeable future”	Good protection against quantum computers unless Shor’s algorithm applies.

From “ECRYPT II Yearly Report on Algorithms and Keysizes (2011-2012)”

Key length

Table 7.2: Key-size Equivalence.

Security (bits)	RSA	DLOG		EC
		field size	subfield	
48	480	480	96	96
56	640	640	112	112
64	816	816	128	128
80	1248	1248	160	160
112	2432	2432	224	224
128	3248	3248	256	256
160	5312	5312	320	320
192	7936	7936	384	384
256	15424	15424	512	512

Table 7.3: Effective Key-size of Commonly used RSA/DLOG Keys.

RSA/DLOG Key	Security (bits)
512	50
768	62
1024	73
1536	89
2048	103

From “ECRYPT II Yearly Report on Algorithms and Keysizes (2011-2012)”

Attacks: Short plaintexts enable a “meet-in-the-middle” attack

A common use is to transmit keys for use in AES or DES

An RSA “block” can have, say ~ 200 (base 10) digits. If $m \approx 10^{19}$ (a DES key), then Eve can make two lists:

cx^{-e} and $y^e \pmod{n}$ for x and $y < 10^9$

a match between the two lists obeys

$c = (xy)^e \bmod n$, or $xy = m$

Simple fix: attach random bits before message. More advanced fix: RSA-OAEP (Optimal Asymmetric Encryption Padding, recommended by ECRYPT), see the book

Attacks: Partial information on p or d enable efficient factoring

Theorem: Let $n = pq$ have m digits. If we know the first $m/4$ or the last $m/4$ digits of p, we can efficiently factor n

- Don't use "simplified" schemes to find primes

Theorem: Suppose (n, e) is an RSA public key and n has m digits. If we know the last $m/4$ digits of d, we can factor n in time linear in $e \log e$

- Even little information on d enables factorization

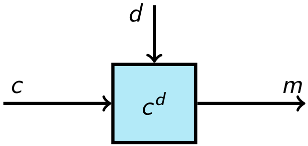
Attacks: Low exponent

- The encryption exponent e is often chosen small to enable fast encryption (a popular value is 65337). Don't do the same choice for d.
- Obviously, d should not be reachable by brute force, but there are other requirements too...

  Theorem: Suppose $q < p < 2q$, e, d, and n as in RSA. If $d < (n^{1/4})/3$, then n can be factored efficiently
- One possibility is to choose d first and then find e

Attacks: Timing

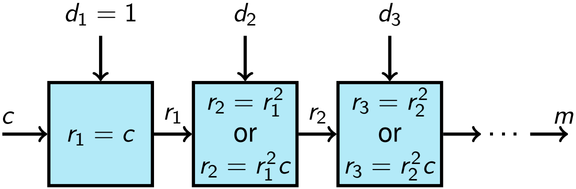
- Even if you choose parameters secure according to all advice, your implementation may still be weak
- The ”fast” modular exponentiation should not be used directly



Attacks: Timing

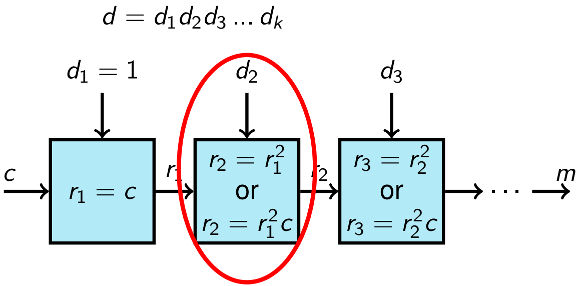
- Even if you choose parameters secure according to all advice, your implementation may still be weak
- The "fast" modular exponentiation should not be used directly

$$d = d_1 d_2 d_3 \dots d_k$$



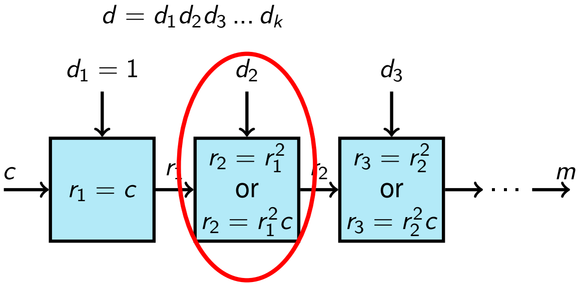
- The time the decryption takes is public in many systems

Attacks: Timing



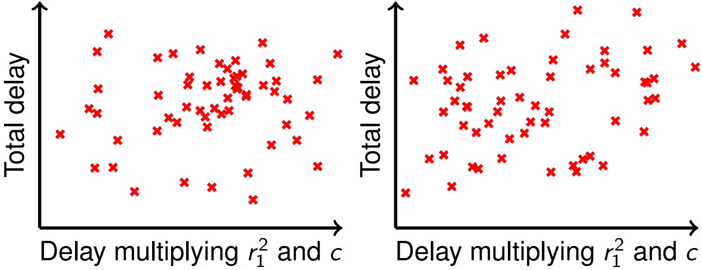
- Eve doesn't know what $d_2, d_3, \ldots$ are
- Eve does know what $r_1 = c$ is
- Eve knows the system: the time it takes for the system to multiply $r_1^2 = c^2$ with c.
- The delay in the second box will depend on c and d_2

Attacks: Timing



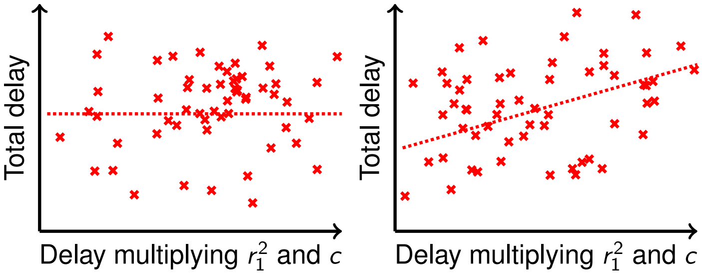
- The delay in the second box will depend on c and d_2
- If $d_2 = 0$, there is no delay
- If $d_2 = 1$, the delay is the time it takes for the system to multiply $r_1^2 = c_2$ with c, which depends on c.

Attacks: Timing



- The delay in the second box will depend on c and d_2
- If $d_2 = 0$, there is no delay
- If $d_2 = 1$, the delay is the time it takes for the system to multiply $r_1^2 = c_2$ with c, which depends on c.

Attacks: Timing



- If there is positive correlation between the delay of multiplying r_1^2 and c and the total delay, then $d_2 = 1$
- Eve now knows d_2, and consequently $r_2^2 = r_1^2 c^{d_2}$
- Lather, rinse, repeat
- Avoid timing attacks by using constant-time implementation, or “blinding”, see the book

Rivest Shamir Adleman (1977)

- Bob chooses secret primes p and q, and sets $n = pq$
- Bob chooses e with $\gcd(e, \phi(n)) = 1$
- Bob computes d so that $de = 1 \bmod \phi(n)$
- Bob makes n and e public but keeps p, q and d secret

- Alice encrypts m as $c = m^e \bmod n$
- Bob decrypts c as $m = c^d \bmod n$
- Choose primes p and q wisely, and implement wisely

Trapdoor one-way functions

- A trapdoor one-way function is a function that is easy to compute but computationally hard to reverse
  - Easy to calculate $f(x)$ from x
  - Hard to invert: to calculate x from $f(x)$
- A trapdoor one-way function has one more property, that with certain knowledge it *is* easy to invert, to calculate x from $f(x)$
- There is no proof that trapdoor one-way functions exist, or even real evidence that they can be constructed. Our example is x^e mod pq.
- What about harder computational problems, say NP problems?

The right kind of problem?

- NP (Nondeterministic Polynomial time)-problems are an important concept in complexity theory
- The necessary effort to solve any problem is usually approximated as some expression involving the size of a parameter
- In cryptography we want to know that there is no better attack method than exhaustive search, which grows exponentially with the size of the key
- Problems that can be solved in polynomial time lie in a smaller class of problems, P
- The problems believed to be in NP but not in P do not have efficient solutions, the known algorithms grow faster than any polynomial expression of the size of a problem parameter

The class of NP-complete problems

- NP-complete problems are the hardest problems in the NP complexity class: any other NP problem can be rewritten as a NP-complete problem in polynomial time
- It is unknown whether P$\neq$NP. If an NP-complete problem can be solved in polynomial time, then P=NP
- One example of a NP-complete problem is “the knapsack problem”
- Sounds like a good problem to base cryptography on…

The knapsack problem, original

- A travelling salesman wants to pack as many items as possible in his knapsack (=bag)
- All items have different sizes
- How can he find the subset that maximizes the total size into the knapsack?
- A physical knapsack is 3D, so let us simplify into one dimension

The one-dimensional knapsack

You are given a set of numbers, all different,

$$D = \{d_1, d_2, \dots , d_n\},$$

and the sum c of the elements in a subset M of D, but the subset M is unknown to you

The knapsack problem is now to deduce what elements are in M (what bits are set in the message)

Example of a one-dimensional knapsack problem

$$D = \{62, 93, 81, 88, 102, 37\},\ \ c = 280$$

In general, solving this is an NP-complete problem.

Methods that solve it include, for example, exhaustive search. In this case, 62+93+88+37=280

The subset $M = \{62, 93, 88, 37\}$ (the message is 110101)

The one-dimensional knapsack

You are given a set of numbers, all different,

$$D = \{d_1, d_2, \dots, d_n\},$$

and the sum c of the elements in a subset M of D, but the subset M is unknown to you

The knapsack problem is now to deduce what elements are in M (what bits are set in the message)

If D is “superincreasing”, the problem is simple to solve, but the knapsack problem in its general version is NP-complete

Example of a superincreasing one-dimensional knapsack problem

A superincreasing knapsack is ordered, and each element is larger than the sum of the previous

$$D_s = \{2, 3, 6, 13, 27, 52\}, \ c_s = 70$$

Solution:

52 is less than 70, so 52 **must** be in M, remains 18
27 is more than 18, so 27 cannot be in M, remains 18
13 is less than 18, so 13 **must** be in M, remains 5
6 is more than 5, so 6 cannot be in M, remains 5
3 is less than 5, so 3 **must** be in M, remains 2
2 is what remains, so 2 **must** be in M, solution found

The subset $M = \{2, 3, 13, 52\}$ (the message is 110101)

Superincreasing and ordinary one-dimensional knapsacks

Examples:

$$D_{\mathsf{s}} = \{2, 3, 6, 13, 27, 52\},\ \ c_{\mathsf{s}} = 70$$

$$D = \{62, 93, 81, 88, 102, 37\},\ \ c = 280$$

Is it possible to map one into the other?

Trapdoor: make an ordinary knapsack out of a superincreasing one

Example:

$$D_{\mathrm{s}} = \{2, 3, 6, 13, 27, 52\}$$

Transform knapsack: Take two numbers s and u with $s >$knapsack total and $\gcd(u, s) = 1$, and multiply each element with u mod s

In our example, use $s = 105$ and $u = 31$

$$D = \left\{ \begin{array}{ccc} 2 \cdot 31 = 62, & 3 \cdot 31 = 93, & 6 \cdot 31 = 81, \\ 13 \cdot 31 = 88, & 27 \cdot 31 = 102, & 52 \cdot 31 = 37 \end{array} \right\}$$

The knapsack problem is NP-complete, so use the new knapsack D as encryption key

In this example $c = 280$ (110101)

Trapdoor: make an ordinary knapsack out of a superincreasing one

The (secret) decryption key is the superincreasing knapsack D and the modular transformation

Decryption is now simple. Divide the cryptotext c with $u \bmod s$ (possible since $\gcd(u, s) = 1$)

$$c_\mathrm{s} = c/31 = 280/31 = 280 \cdot 61 = 70 \bmod 105,$$

then use the superincreasing knapsack to read off the value

$$D_\mathrm{s} = \{2, 3, 6, 13, 27, 52\}, \ c_\mathrm{s} = 70, \ M = \{2, 3, 13, 52\} \ (110101)$$

Weakness of the knapsack

- You can recreate D_s from D using u and s
- The weakness is that *any* u and s that creates a superincreasing knapsack makes the problem simple, it is irrelevant if this is the original one or not
- And such values are easy to find, if D is constructed from a superincreasing knapsack
- Remember, in general, the knapsack problem is NP-complete, but (as it turns out) if the knapsack is *constructed from* a superincreasing one, the problem is much simpler
- The trapdoor is too big

A different idea: double one-way functions

- Use two one-way functions f and g that satisfy the symmetry

$$g(f(a), b) = g(f(b), a)$$

- This cannot be used for encryption/signing because one does not necessarily recover a or b
- But it can be used for key exchange
  - Alice takes a secret random a and makes $f(a)$ public
  - Bob takes a secret random b and makes $f(b)$ public
  - Both can now create $k = g(f(b), a) = g(f(a), b)$

Diffie-Hellman key exchange

Use exponentiation mod p:

$$g(x, y) = x^y \bmod p$$
$$f(x) = g(\alpha, x) = \alpha^x \bmod p$$

where α is a "primitive root of numbers mod p"

The symmetry is

$$g(f(a), b) = (\alpha^a)^b = (\alpha^b)^a = g(f(b), a) \bmod p$$

This can be used for key exchange: parameters p and α

- Alice takes a secret random a and makes $\alpha^a \bmod p$ public
- Bob takes a secret random b and makes $\alpha^b \bmod p$ public
- Both can now create $k = (\alpha^a)^b = (\alpha^b)^a \bmod p$

Security of Diffie-Hellman key exchange

The one-way function is exponentiation mod p, so security depends on the difficulty of calculating discrete logarithms,

$$\text{“log”}_\alpha(t) = L_\alpha(t),\ \text{the solution to } \alpha^x = t \bmod p$$

If discrete logaritms are easy to calculate, Eve can do $\mathrm{L}_\alpha(\alpha^a) = a$

Reminder: RSA also needs this to be a hard problem since $L_c(m) = d$ (but that’s another story)

Security of Diffie-Hellman key exchange

The one-way function is exponentiation mod p, so security depends on the difficulty of calculating discrete logarithms,

$$\text{"log"}_\alpha(t) = L_\alpha(t),\ \text{the solution to } \alpha^x = t \bmod p$$

To ensure existence of the discrete logarithm, p needs to be prime and the number α needs to be a primitive root mod p, and to make it unique, we choose the smallest possible solution to the equation

Behaves like the usual logarithm, in particular

$$L_\alpha(ab) = L_\alpha(a) + L_\alpha(b) \bmod p - 1$$

Calculating the discrete logarithm

The discrete logarithm $L_\alpha(t)$ is the solution to the equation $\alpha^x = t \bmod p$

A simple thing to do is to determine if x is even or odd ($p-1$ is even)

$$\alpha^{p-1} = 1 \bmod p$$
$$\alpha^{(p-1)/2} = \pm 1 \bmod p$$

but that means

$$\alpha^{(p-1)/2} = -1 \bmod p$$
$$t^{(p-1)/2} = \alpha^{x(p-1)/2} = (-1)^x \bmod p$$

In other words, if $t^{(p-1)/2} = 1$, then $x = 0 \bmod 2$, otherwise $x = 1 \bmod 2$

Calculating the discrete logarithm

The discrete logarithm $L_\alpha(t)$ is the solution to the equation $\alpha^x = t \bmod p$

OK, so we know if x is even or odd. Now, if $p-1$ is divisible by 3 (and not by 9), and

$$t^{(p-1)/3} = \alpha^{x(p-1)/3} \bmod p$$

There are only three possible values of

$$x^{(p-1)/3} \bmod p-1$$

Exhaustive search will give you the (unique) solution

$$x \bmod 3$$

Calculating the discrete logarithm

The discrete logarithm $L_\alpha(t)$ is the solution to the equation $\alpha^x = t \bmod p$

Suddenly we have $x \bmod 2$ and $x \bmod 3$. This can be continued, but will only work for small primes (and powers of small primes, see the book)

If we can do this for all prime power factors of $p - 1$, we can use the Chinese remainder theorem to reconstruct x

The procedure is called the **Pohlig-Hellman algorithm** and works when $p - 1$ has only small factors

This works for the same reason that the Pollard $p - 1$ method can factor $n = pq$ if $p - 1$ has only small factors

Calculating the discrete logarithm

The discrete logarithm $L_\alpha(t)$ is the solution to the equation $\alpha^x = t \bmod p$

- **The Baby step, Giant step method**: choose $N^2 \geq p - 1$ and build two lists of N numbers α^j, and $t\alpha^{-Nk}$. Look for a match between the lists, use the match to form $x = j + Nk$ (works up to 20-digit p)
- **Index calculus** uses similar ideas as **Quadratic sieve factoring**: find a list of α^j mod p that are products of small primes. Match these against a similar list of $t\alpha^k$ mod p that also are products of small primes, and solve the resulting equations (choose 200-digit p to be safe)

ElGamal encryption

- Choose a large prime p, and a primitive root α mod p. Also, take a random integer a and calculate $\beta = \alpha^a$ mod p
- The public key is the values of p, α, and β, while the secret key is the value a
- Encryption uses a random integer k, and the ciphertext is the pair $(\alpha^k, \beta^k m)$
  - α^k is used to transmit the “one-time secret” k
  - β^k is the “one-time pad” for m
- Decryption is done with a, by calculating

$$(\alpha^k)^{-a}(\beta^k m) = (\alpha^{-ak})(\alpha^{ak} m) = m \bmod p$$

Security of ElGamal encryption

- The one-way function is (again) exponentiation mod p, so security depends on the difficulty of calculating discrete logarithms $L_\alpha(t)$, the solution to $\alpha^x = t \bmod p$
- If discrete logarithms are easy to calculate, Eve can do $L_\alpha(\beta) = a$ and decrypt using $(\alpha^k)^{-a}(\beta^k m) = (\alpha^{-ak})(\alpha^{ak} m) = m$
- ElGamal is slightly better off than vanilla RSA because of the random k used, so short messages are less of a problem. It rather compares with RSA-OAEP

Key length

Table 7.2: Key-size Equivalence.

Security (bits)	RSA	DLOG		EC
		field size	subfield	
48	480	480	96	96
56	640	640	112	112
64	816	816	128	128
80	1248	1248	160	160
112	2432	2432	224	224
128	3248	3248	256	256
160	5312	5312	320	320
192	7936	7936	384	384
256	15424	15424	512	512

Table 7.3: Effective Key-size of Commonly used RSA/DLOG Keys.

RSA/DLOG Key	Security (bits)
512	50
768	62
1024	73
1536	89
2048	103

From “ECRYPT II Yearly Report on Algorithms and Keysizes (2011-2012)”

Trapdoor one-way functions

- A trapdoor one-way function is a function that is easy to compute but computationally hard to reverse
  - Easy to calculate $f(x)$ from x
  - Hard to invert: to calculate x from $f(x)$
- A trapdoor one-way function has one more property, that with certain knowledge it *is* easy to invert, to calculate x from $f(x)$
- There is no proof that trapdoor one-way functions exist, or even real evidence that they can be constructed. Examples:
  - RSA (factoring)
  - Knapsack (NP-complete but insecure with trapdoor)
  - Diffie-Hellman + ElGamal (discrete log)