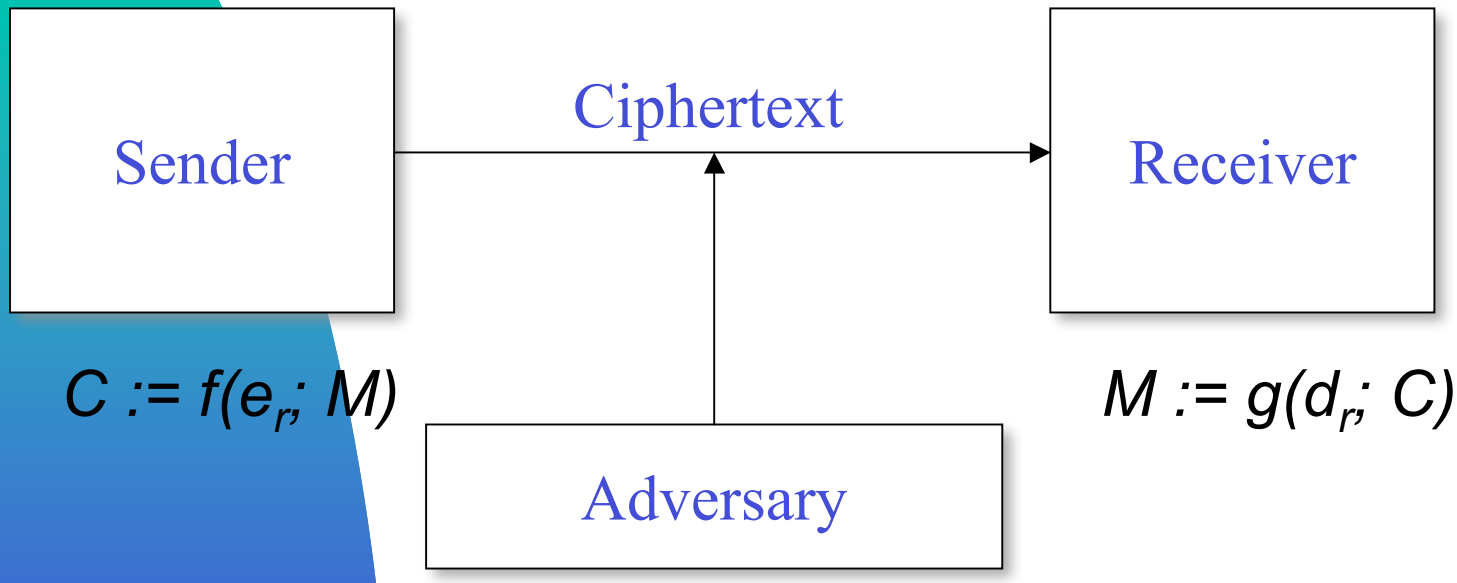
CINVESTAV

Outline

- Public-key cryptography
- A collection of *hard* problems
- Mathematical Background
- Trapdoor Knapsack
- Integer factorization Problem
- Discrete logarithm problem revisited
- Case of Study: The Sun NFS Cryptosystem
- The RSA Algorithm
- Elliptic curves and the discrete logarithm problem
- Available public-key technologies
- Comparison of Public-Key Cryptographic Systems
- Diffie-Hellman protocol
- Digital Signature

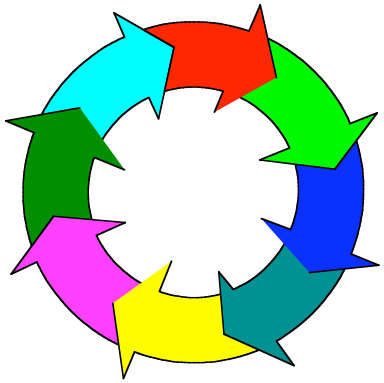
Public-key cryptography

User	Key
S	e_s
R	e_r



Public-key cryptography

- Functions $f(e_r, -)$ and $f(d_r, -)$ are inverse of one another, but $e_r \neq d_r$
- C := $f(e_r, M)$ and M := $f(d_r, C)$
- e_r is *public*; known to everyone
- d_r is *private*; known only by user R
- e_r is *easily* deduced from d_r
- d_r is NOT *easily* deduced from e_r

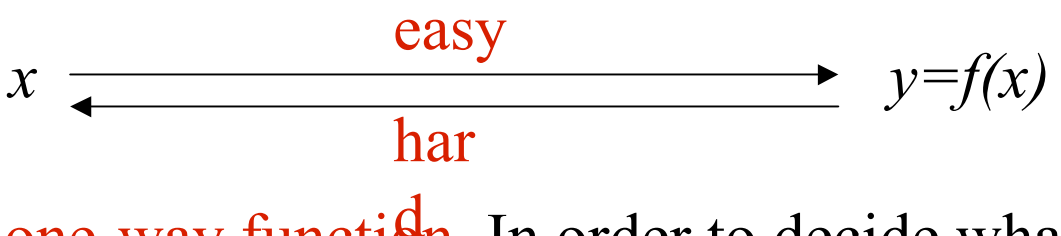


Public-key cryptography

A public-key cryptography system is based on a function $f(x)$ such that

Given x, computing $y=f(x)$ is easy

Given $y=f(x)$, computing s is hard



We call $f(x)$ a one-way function. In order to decide what is hard, we use the theory of complexity. Often, the test of time determines.

one-way function examples

- Discrete Logarithm
  - Given x, a and p, computing $y=x^a \ mod \ p$ is easy. However, given y, x and p, computing a is hard.
- Factoring
  - Given x and y, computing $n=xy$ is easy. However, given n, computing the factors x and y is hard.
- Discrete Square-root
  - Given x and n, computing $a=x^2 \ mod \ n$ is easy. However, given a and n, computing x is hard.

Discrete Logarithm example

- For $x=6$, $a=9$, $p=11$, we compute

$$Y=x^a = x((x^2)^2)^2 \bmod p$$

With 4 multiplications,

$$\begin{aligned} y &= 6((6^2)^2)^2 = 6((36)^2)^2 \\ &= 6((3)^2)^2 = 6(9)^2 \\ &= 6(81) = 6(4) = 24 = 2 \end{aligned}$$

However, finding an a such that $6^a = 2 \bmod 11$ is hard

Trapdoor Function

- One additional structure about the function $y=f(x)$ is needed to design a public-key cryptosystem.

Given y and some special information about $f(x)$, computing x is easy.

Given y without this special information, computing x is hard.

We call $f(x)$ a one-way trapdoor function. The special information is the trapdoor information.

A collection of *hard* problems

- Subset sum (knapsack)

  $x_i \in \{0,1\},\ 1 \le i \le n,$ such that $s = \sum_{i=1}^{n} a_i x_i$

- Integer factorization problem
  - Find prime divisors of an integer n
- Quadratic residue
  - $a = x^2 \ (mod\ n)$
- Discrete logarithm
  - $x = log_a b$ in group G (e.g. Z_p^*)

A collection of *hard* problems

- RSA problem
  - $c = m^e \pmod n$, $n=pq$, $gcd(e,(p-1)(q-1))=1$
  - $x^{\phi(n)} = 1 \pmod n$ with $gcd(x,n)=1$
- Elliptic curves
  - P,Q in $E(F_q)$, with $ord(P)=n$, $Q=mP$
  - Find integer m with $0 \leq m \leq n$

Math Background: Greatest Common Divisor

- The greatest common divisor *(a,b)* of *a* and *b* is the largest number that divides evenly into both *a* and *b*. Euclid's Algorithm is used to find the Greatest Common Divisor (GCD) of two numbers *a* and *n*, *a<n*

  Fact: if *a* and *b* have divisor *d* so does *a-b*, *a-2b*.

  GCD (a,n) is given by:

  let $g_0=n$; $g_1=a$;

  $g_{i+1} = g_{i-1} \bmod g_i$

  when $g_i=0$ *then* $(a,n) = g_{i-1}$

  Example: find *(56,98)*

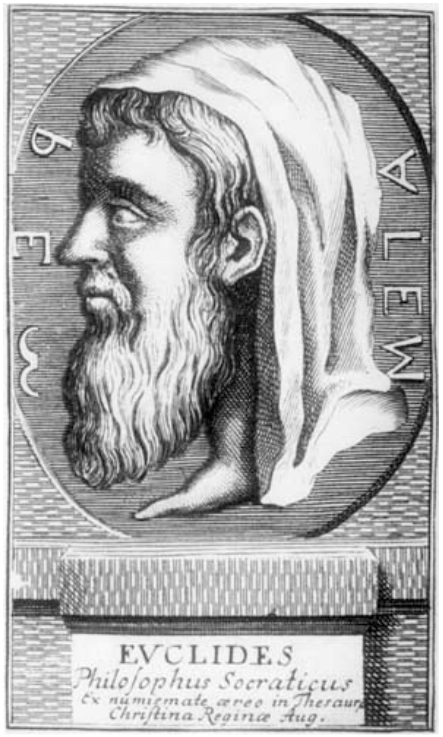
  g0 = 98; g1=56;

  g2 = 98 mod 56 = 42;

  g3 = 56 mod 42 = 14;

  g4 = 42 mod 14 = 0;

  hence *(56,98)=14.*



Math Background: Inverses and Extended Euclid's algorithm.

If $(a,n)=1$ then the number a in modular arithmetic has always a unique inverse a^{-1} in *mod n* given as,

$a.a^{-1} = 1 \bmod n$, where a, a^{-1} in $\{0,n-1\}$.

Example: $3*7 = 1 \bmod 10$

We can extend Euclid's Algorithm to find inverses by keeping track of $g_i = u_i n + v_i a$ as follows,

Inverse(a,n) is given by (only defined iff $(a,n)=1$):

$g_0=n;\ u_0=1;\ v_0=0;$

$g_1=a;\ u_1=0;\ v_1=1;$

let

$$y = g_{i-1} \ div \ g_i$$
$$g_{i+1} = g_{i-1} - yg_i = g_{i-1} \bmod g_i$$
$$u_{i+1} = u_{i-1} - yu_i$$
$$v_{i+1} = v_{i-1} - yv_i$$

when $g_i=0$ then $Inverse(a,n) = v_{i-1}$

Math Background: Euler Totient Function $\Phi(n)$

- if we consider arithmetic modulo n, then a reduced set of residues is a subset of the complete set of residues modulo n which are relatively prime to n
- Example for $n=10$,
- the complete set of residues is $\{0,1,2,3,4,5,6,7,8,9\}$
- the reduced set of residues is $\{1,3,7,9\}$
- the number of elements in the reduced set of residues is called the Euler Totient function $\Phi(n)$
- there is no single formula for [[Φ]](n) but for various cases count how many elements are excluded[4]:
- p (p prime) $\Phi(p)=p-1$
- $p.q$ (p,q prime) $\Phi(p)=(p-1)(q-1)$

Trapdoor Knapsack: definitions

- *Knapsack problem* Let $I=\{0,1,\ldots,n-1\}$. Given the integer vector $A=\{a_0,a_1,\ldots,a_{n-1}\}$ and another integer X, is there a $J \subseteq I$ such that

$$\sum_{i \in J} a_i = X$$

- *Easy Knapsack problem* If the numbers a_i have the **superincreasing property**, i.e.,

$$\sum_{i=0}^{j-1} a_i < a_j$$

then the knapsack problem is easy; however, without the superincreasing property, the Knapsack problem (in general) is hard.

Trapdoor Knapsack: Examples

- Easy Example

  $A=\{1,2,4,8,16\}$; superincreasing

  $1<2;\ 1+2<4;\ 1+2+4<8;\ 1+2+4+8<16$

  Let $X=23$. Solution is found by computing the binary expansion of $X=23=(10111)_2$ thus $1+2+4+16=23$.

- Hard Problem

  $A=\{3,4,5,12,13\}$; non-superincreasing

  Let $X=19$. We need to try all subsets of A to find out which one of these sums 19.

$3+4=7$	$3+5=8$	$3+12=15$
$3+13=16$	$4+5=9$	$4+12=16$
$4+13=17$	$5+12=17$	$5+13=18$
$12+13=25$	$3+4+5=12$	$3+4+12=19$

Trapdoor Knapsack: Design example

Let a knapsack vector be $A=\{1,3,7,13,26,65,119,267\}$. An easy Knapsack problem is a vector such that the sum of weights of its elements is a superincreasing sequence. A superincreasing sequence is a sequence in which every term is greater than the sum of all the previous terms. For the vector A defined above we have a superincreasing sequence, as next table shows

$A[i]$	1	3	7	13	26	65	119	267
Σ of the previus terms	0	1	4	11	24	50	115	234

In order to find t^{-1} such that $(t^{-1}t) \bmod p=1$, where both, t and p are given, we can use the extended Euclid's algorithm. It is easy to find that for, $t=467$, $p=523$ we obtain

$$t^{-1}=28.$$

Trapdoor Knapsack: Design example

- We can now create a hard knapsack vector by multiplying all the original values a_i in A by $(a_i\ t)\ mod\ p$ (p should be greater than the sum of all numbers in A; t and p should be relatively primes, and finally, $0<t<p-1$). The resulting hard knapsack vector is shown in next table

original Knapsack vector	1	3	7	13	26	65	119	267
hard Knapsack vector	467	355	131	318	113	21	135	215

- The new hard Knapsack vector is the public key. The original easy Knapsack vector along with t and t^{-1}, constitute the secret key.

Trapdoor Knapsack: Encryption

- Given the message,

01001011 11010110

- We can encrypt it by first breaking it up into blocks equal to the number of items in the Knapsack sequence (i.e., 8 bits). Then, if the i-esime bit is 1 we add the weight of the corresponding i-esime Knapsack item (*A[i]*), and if the i-esime bit is zero, we add nothing. For the given plaintext we have two blocks of eight bits each. The corresponding ciphertext is then,

818, 1296

Trapdoor Knapsack: Decryption

- To decrypt the ciphertext previously generated, we have to multiply each cipher-block by $t^{-1} \bmod p$:

  $818 * t^{-1} \bmod p = (818 \times 28) \bmod 523 = 415 = 267 + 119 + 26 + 3$,

  which correspond to

  01001011

  $1296 * t^{-1} \bmod p = (1296 * 28) \bmod 523 = 201 = 119 + 65 + 13 + 3 + 1$,

  which correspond to

  11010110

  These results coincide with the original plaintext.

Trapdoor Knapsack: Discussion

- Knapsack ciphers originally seemed to be excellent candidates for use in public key cryptosystems. However, Shamir has shown that they are not satisfactory for public-key cryptography as he broke the code using an efficient algorithm that needs only O(P(n)) bit operations, where P is a polynomial.
- There are several possibilities for altering this cipher system to avoid the weakness found by Shamir. Unfortunately, efficient algorithms have been also found for these cases.
- A comprehensive discussion of Knapsack ciphers can been found in the article “The rise and fall of Knapsack cryptosystems” by Odlyzko.

Integer factorization Problem

The integer factorization problem (IFP) is the following: Given a composite number n that is the product of two large prime numbers p and q, find p and q.

•While finding large prime numbers is a relatively easy task, the problem of factoring the product of two such numbers is considered computationally intractable if the primes are carefully selected. Based on the difficulty of this problem, Rivest, Shamir and Adleman developed the RSA public-key cryptosystem. Another public-key cryptosystem whose security lies on the intractability of IFP is due to Rabin and Williams.

•While the integer factorization problem has received some attention over the centuries from well-known mathematicians like Fermat and Gauss, it is only in the past 20 years that significant progress has been made towards its resolution.

Discrete logarithm problem revisited

- If p is a prime and g and x integers, the computation of y such that:

$$y = g^x \bmod p,\ 0 <= y <= p - 1$$

is referred to as a discrete exponentiation. Using the successive squaring method, it is very fast (polynomial in the number of bits of $|p| + |g| + |x|$).

On the other hand, the inverse problem, namely; given p, g, and y, to compute some x such that the above equation holds, which is referred to as the discrete logarithm problem (DLP), appears to be quite hard in general.

The discrete logarithm problem applies to groups (Galois field). There are algorithms that solve the problem to compute discrete logarithm in $GF(p)$ in time roughly $q^{1/2}$, where q is the largest prime dividing $p - 1$.

Discrete logarithm problem revisited

- Therefore, it is advisable to choose p such that $p-1$ is divisible by at least one large prime q, say $q > 10^{30}$.
- Also, it is advisable to choose g with its order divisible by a large prime (recall that g is called a generator or primitive element if it can generate all the elements in the group).
- If the above two precautions are observed, then the best published algorithm for computing DLP have running time:

  $exp((1 + O(1))(\log p)^{1/2}(\log\log p)^{1/2})$, as p goes to infinite.
- Currently large hard integers (hard here means that they are not of a special form and do not have small prime factors) with 100 to 110 decimal digits are factored in the equivalent of under a year on a 100 mips computer.

  Note: (a Pentium μprocessor @ 200MHz is about a 50-MIPS machine).

Discrete logarithm problem revisited

- The current record in factoring large RSA keys is the factorization of a 512 bit (155 digit) number achieved in August 1999 by running the number field sieve algorithm on hundreds of workstations for several months (http://www.rsasecurity.com/rsalabs/challenges/factoring/rsa155.html).
- The CPU-effort is estimated to be equivalent to approximately 8000 MIPS years; calendar time for the sieving was 3.7 months.
- Adi Shamir proposed in September 1999 the use of an opto-electronic device which, he claims, will be able to increase the size of factorable numbers by 100 to 200 bits, and in particular can make 512 bit RSA keys (which protect 95% of today's E-commerce on the Internet) very vulnerable.

Case of Study: The Sun NFS Cryptosystem.

- It has long been known that the Sun system is not very secure (as it was in 1992).
- The Sun security option in their NFS is built into the basic Sun Remote Procedure Call (RPC) and provides authentication of both users and machines using a combination of the Needham-Schroeder protocol which uses DES, and a public key cryptosystem that is a modification of DHC.
- In the Sun system, there is a prime p and an integer g that are the same for all users and all machines around the world that use this software.
- Each user or machine has a secret key m, and $g^m \ mod \ p$ is public. Authentication involves proving that one possesses the key m.
- p is a 192-bit prime, where both, p and $(p-1)/2$ are primes. The integer $g = 3$ is a primitive root modulo p.

The Sun challange

- The Challenge, provided by M. Shannon of Sun Microsystems, was to find m such that:

$$3^m = z \bmod p,$$

Where

$p =$
5213619424271520371687014113170182341777563603680354416779;

$z =$
308899365792522917304711040535452115103232581944049898356 5;

$g = 3$.

The Sun challange

- The general conclusion obtained by LaMacchia, Odlyyzko in 1990 is that computing discrete logarithms modulo a prime, is only a little harder than factoring integers of the same size (actually, about 10% more harder).
- The basic idea of using a modification of Diffie-Hellman scheme in the Sun NFS system is not necessarily bad. From LaMacchia's results, the immediate conclusion is that the number p has to be chosen much larger for proper security.
- A size of at least *512* bits for p was suggested at that time.
- There is even a better algorithm called number field Sieve proposed by Odlyyzko, which promises a much lower asymptotic running time. Nevertheless, this algorithm had several practical implementation problems at the time that LaMacchia published his results.

The RSA Algorithm

"It is not what it is, but what it seems to be"

W. Shakespeare

- The RSA algorithm was invented by Rivest, Shamir and Adleman in 1977.

  Let p and q be two distinct large random primes. The modulus n is the product of these two primes, $n=pq$

  Euler's totient function of n is given by

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

  Now, let us select a number $1<e<\Phi(n)$ such that

  $$gcd(e, \Phi(n))=1$$

  and compute d with

  $$d=e^{-1}(mod\ \Phi(n))$$

  using the extended Euclid's algorithm. Under this scheme, e is the public exponent and d is the private exponent. Usually, one selects a small public exponent (e.g., $e=2^{16}+1$).
- The modulus n and the public exponent e are published. The value of d and the prime numbers p and q are kept secret.

The RSA Algorithm

- Encryption Is performed by computing
  - $C=M^e(mod n)$

Where M is the plaintext such that $0< M<n$

- Decryption is performed by computing
  - $M=C^d(mod n)$

The RSA Algorithm

- The correctness of the RSA algorithm follows from Euler's theorem, which in turn is an extension of Fermat's little theorem,

Euler's Theorem: Let n and a be positive, relatively prime integers. Then

$$a^{\Phi(n)} = 1 \bmod n$$

- Proof: Since we have $ed = 1 \bmod \Phi(n)$, we can write $ed = 1 + K\,\Phi(n)$, for some integer K (why?). Hence,

$$C^d = \left(M^e\right)^d \pmod n = M^{ed} \pmod n = M^{1+K\Phi(n)} \pmod n =$$

$$M \cdot \left(M^{\Phi(n)}\right)^K \pmod n = M \cdot 1 \pmod n$$

provided that $gcd(M,n)=1$.

The RSA Algorithm: A simple example

- As an example, we construct a simple RSA cryptosystem as follows,

  Pick $p=11$ and $q=13$, and compute

  $$n=pq=11*13=143$$

  $$\Phi(n)=(p-1)(q-1)=10*12=120$$

  The public exponent e is selected such that $1<e<\Phi(n)$ and

  $$gcd(e, \Phi(n))= gcd(e, 120)=1$$

The RSA Algorithm: A simple example

For example, select *e=17*. The private key exponent *d* is computed by

$$d = e^{-1} (\mathrm{mod}\, \Phi(n)) = 17^{-1} (\mathrm{mod} 120) = 113$$

which is computed using the extended Euclid algorithm, or any other algorithm for computing the modular inverse. The user publishes the public exponent and the modulus pair,

$(e,n)=(13,143)$, Public key

and keeps the following as private,

$d=113$, $p=11$, $q=13$ Private key

The RSA Algorithm: A simple example

- A typical encryption/decryption process is executed as follows,

Plaintext	$M=50$
Encryption	$C:=M^e(modn)=50^{17}(mod143)=85$

ciphertext	$C=85$
Decryption	$M:=C^d(modn)=85^{113}(mod143)=50$

(How can I compute 85^{113}?)

The RSA Algorithm: Discussion

The security of the RSA algorithm relies in the difficulty of factorize efficiently. If one can factor quickly, then one can break the RSA algorithm. Let us assume that Anita's public keys are published: (e_a, n_a). Then,

- Factor n_a to get p_a and q_a.
- Compute $\phi(n_a) = \phi(p_a q_a) = (p_a - 1)(q_a - 1)$
- Compute $d_a = e_a^{-1} \pmod{\Phi(n_a)}$

Thus, we can now intercept and decrypt all messages sent to Anita.

Thus, factoring $\Rightarrow$ breaking RSA. However,

breaking RSA $\overset{?}{\Rightarrow}$ factoring

No proof exist that breaking RSA is equivalent to factoring.

CINVESTAV

Elliptic curves and the discrete logarithm problem

- As we have seen, at the foundation of every cryptosystem, there is a hard mathematical problem that is computationally infeasible to solve. The discrete logarithm problem is the basis for the security of many cryptosystems including the Elliptic Curve Cryptosystem. More specifically, the ECC relies upon the difficulty of the Elliptic Curve Discrete Logarithm Problem (ECDLP).
- Recall that we examined two geometrically defined operations over certain elliptic curve groups. These two operations were point addition and point doubling. By selecting a point in a elliptic curve group, one can double it to obtain the point $2P$. After that, one can add the point P to the point $2P$ to obtain the point $3P$. The determination of a point nP in this manner is referred to as Scalar Multiplication of a point. The ECDLP is based upon the intractability of scalar multiplication products.

EC Discrete logarithm computational difficulty

- Let us assume that a 1 MIPS machine can perform 4 x 104 elliptic curve additions per second. This assumption is optimistic since ASIC developed by Certicom for performing elliptic curve operations over the field $F2^{155}$ has a 40 MHz clock-rate and can perform roughly 40,000 elliptic additions per second. Also, software implementations on a SPARC IPC (rated at 25 MIPS) perform 2,000 elliptic curve additions per second. Then the number of elliptic curve additions that can be performed by a 1 MIPS machine in one year is about,

$$(4 \text{ X } 104) * (60 \text{ X } 60 \text{ X } 24 \text{ X } 365) \approx 240$$

EC Discrete logarithm computational difficulty

Field Size (in bits)	Size of n (in bits)	MIPS years
163	160	9.6×10^{11}
191	186	7.9×10^{15}
239	234	1.6×10^{23}
359	354	1.5×10^{41}
431	426	1.0×10^{52}

This table shows the computing power required to compute a single EC discrete logarithm using the Pollard rho-method for various values of n.

Available public-key technologies

	Based on Factoring	Based on discrete log in GF (p)	Based on elliptic curve log
Digital Signatures	RSA signatures	DSA	ECDSA
Encryption	RSA encryption	DH	DH over EC
Key Exchange	One-time RSA key pair	Classic DH	DH over EC

Comparison of Public-Key Cryptographic Systems

- **Key Size (bits)**

	System Parameter	*Public Key*	*Private Key*
RSA	N/A	1088	2048
DSA	2208	1024	160
ECC	481	161	160

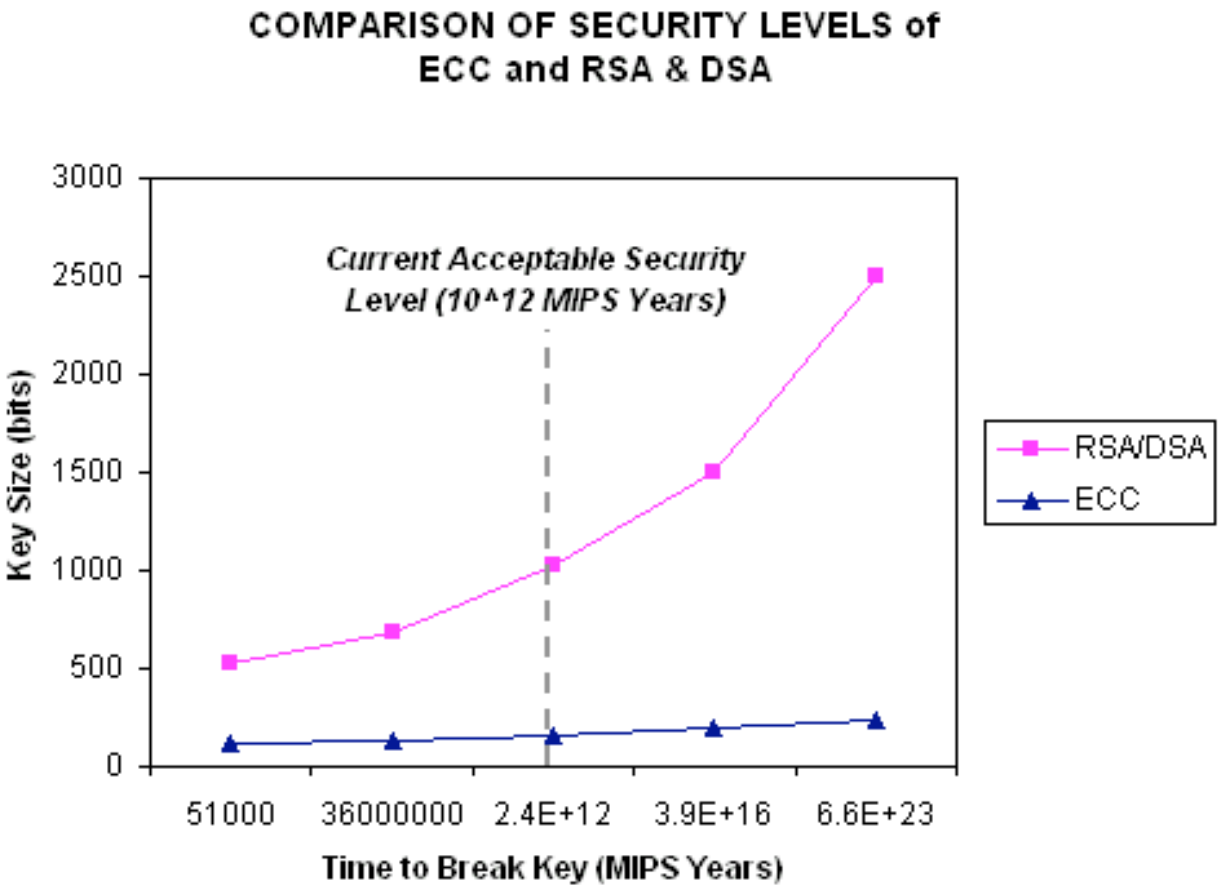
- **Bandwidth (bits)**

	Signature Size		*Encrypted message size*
RSA	1024	RSA	1024
DSA	320	ElGamal	2048
ECC	320	ECC	321

Comparison of Public-Key Cryptographic Systems

	RSA-1024 (e = 3)	DSA-1024	ECDSA-168 (over GF(p))
Parameter gen.	None	Slow	Research prob.
Key generation	150	1	1
Sign	6	1	1
Verify	1	45	30
Encipherment	1	45	30
Decipherment	6	1	1
Key Exchange	Slow	Fast	Fast

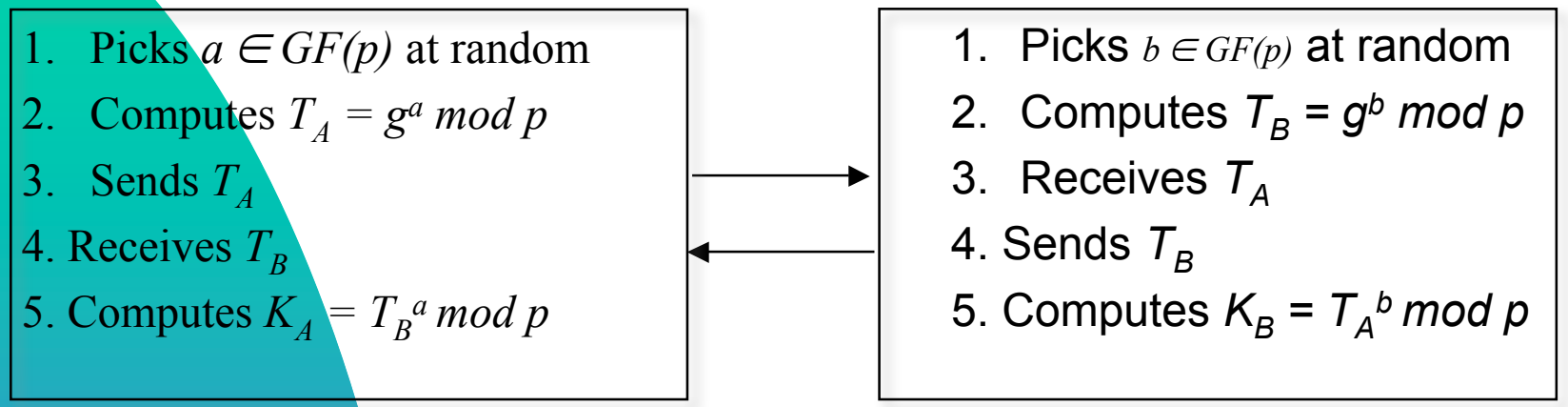
Comparison of Public-Key Cryptographic Systems



Diffie-Hellman protocol

- Diffie-Hellman Cryptosystem (DHC) is the oldest public key system still in use. It was published in 1976.
- DHC allows two individuals to agree on a shared-secret key, over an insecure medium without any prior secrets.
- DHC has two system public parameters p and g. Where:
  - *p* is a prime, and;
  - *g* (usually called primitive or generator) is an integer less than *p*, which is capable of generating every element from 1 to *p-1*, when multiplied by itself a certain number of times, modulo *p*.
  - The protocol depends on the discrete logarithm problem for its security. It assumes that it is computationally infeasible to calculate the shared secret key $k = g^{ab} \bmod p$, where $g, p, g^a \bmod p, g^b \bmod p$ are given; and when *p* is a prime sufficiently large.

Key exchange: Diffie-Hellman protocol



Machine A

Machine B

Where $K = K_A = K_B$, Because:

$$T_B^a = (g^b)^a = g^{ba} = g^{ab} = (g^a)^b = T_A^b \bmod p$$

Mensaje para Anita en La Jornada

Querida Anita de mi corazón:

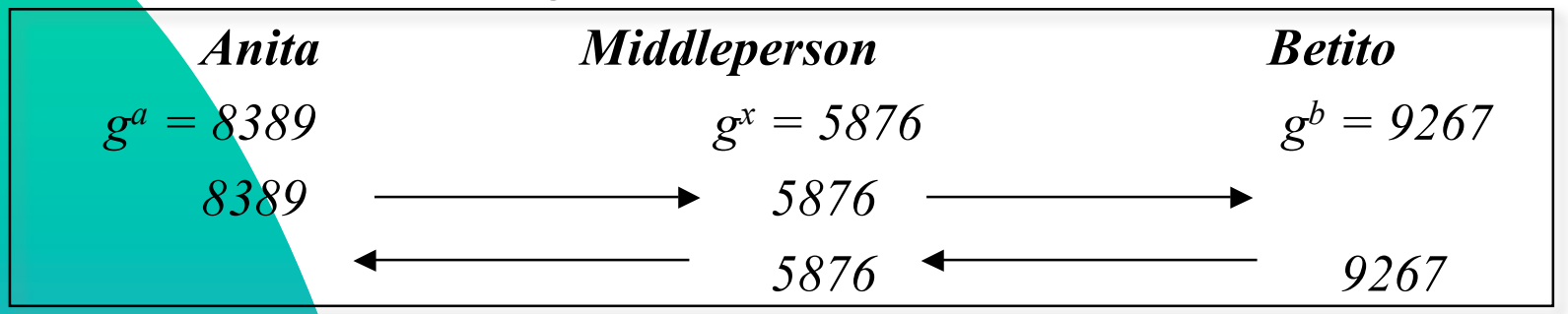
Quisiera pedirte que nuestro número primo sea *128903289023* y nuestra *g 23489*.

Te quiere

Betito.

Middle-person attack.

- Consider the following scenario:



Shared key K_{AX}:

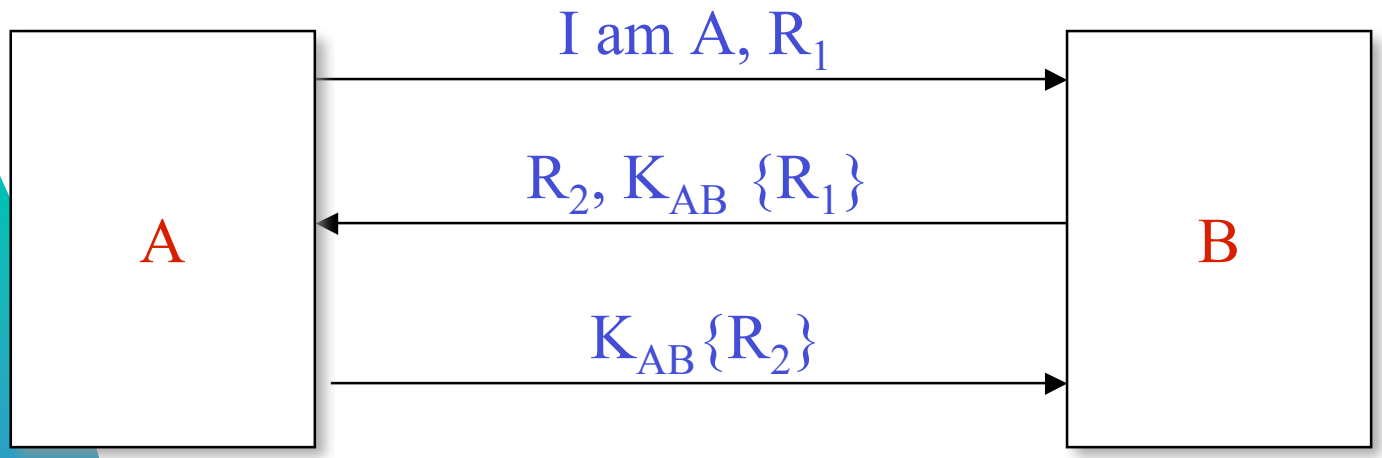
$5876^a = 8389^x$

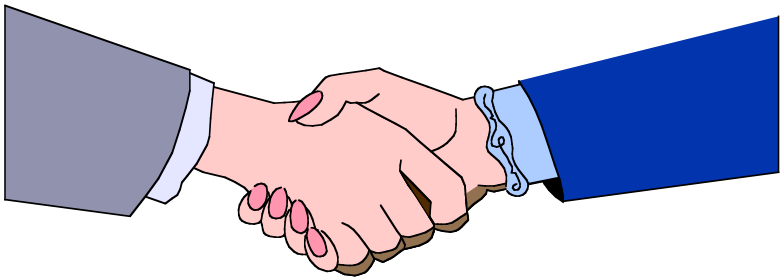
Shared key K_{BX}

$9267^x = 5876^b$

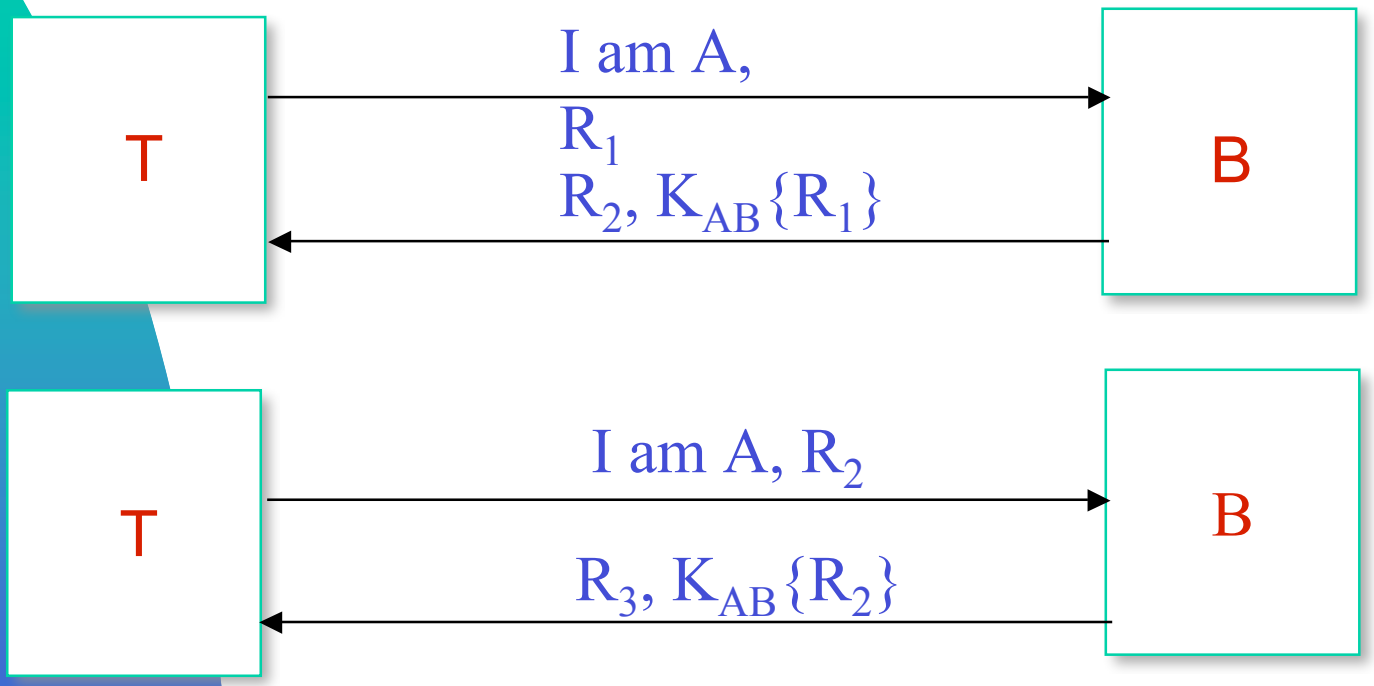
- After this exchange, the middle-person attacker simply decrypts any messages sent out by A or B, and then reads any possibly modifies them before re-encrypting with the appropriate key and transmitting them to the correct party.
- Middle-person attack is possible due to the fact that DHC does not authenticate the participants. Possible solutions are digital signatures and other protocol variants.

Solution: Mutual authentication





Reflection attack



Digital Signatures

- A data string associating a message with an originating entity
  - Signature generation algorithm
  - Signature verification algorithm
  - Signature scheme
- Used for authentication, integrity, and nonrepudiation
- Public key certification is one of the most significant applications

Message Digest

- A message digest, also known as a one-way hash function, is a fixed length computionally unique identifier corresponding to a set of data. That is, each unit of data (a file, a buffer, etc.) will map to a particular short block, called a message digest. It is not random: digesting the same unit of data with the same digest algorithm will always produce the same short block.
- A good message digest algorithm possesses the following qualities
  - The algorithm accepts any input data length.
  - The algorithm produces a fixed length output for any input data.
  - The digest does not reveal anything about the input that was used to generate it.
  - It is computationally infeasible to produce data that has a specific digest.
  - It is computationally infeasible to produce two different unit of data that produce the same digest.