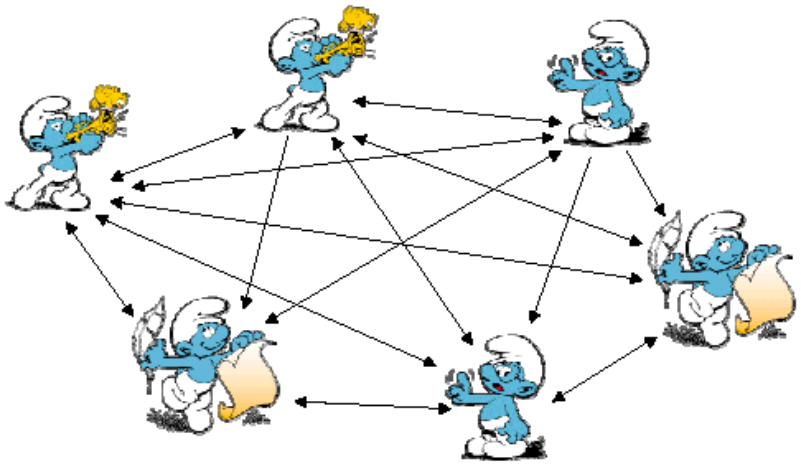
Public Key Cryptography

Motivation for Public Key Cryptography

- In symmetric key or secret key cryptosystems, the communication parties must have some pre-share secret, i.e. the master key
- Distribution of such keys in a secure and scalable manner is a major problem
- The introduction of a trust third party, namely the Key Distribution Center (KDC) solves the problem partially by reducing the number of master keys from $O(N^2)$ to $O(N)$ but still inconvenient and KDC can become the single point of failure and/or performance bottleneck (more details later)
- Symmetric key system also cannot provide *non-repudiation*

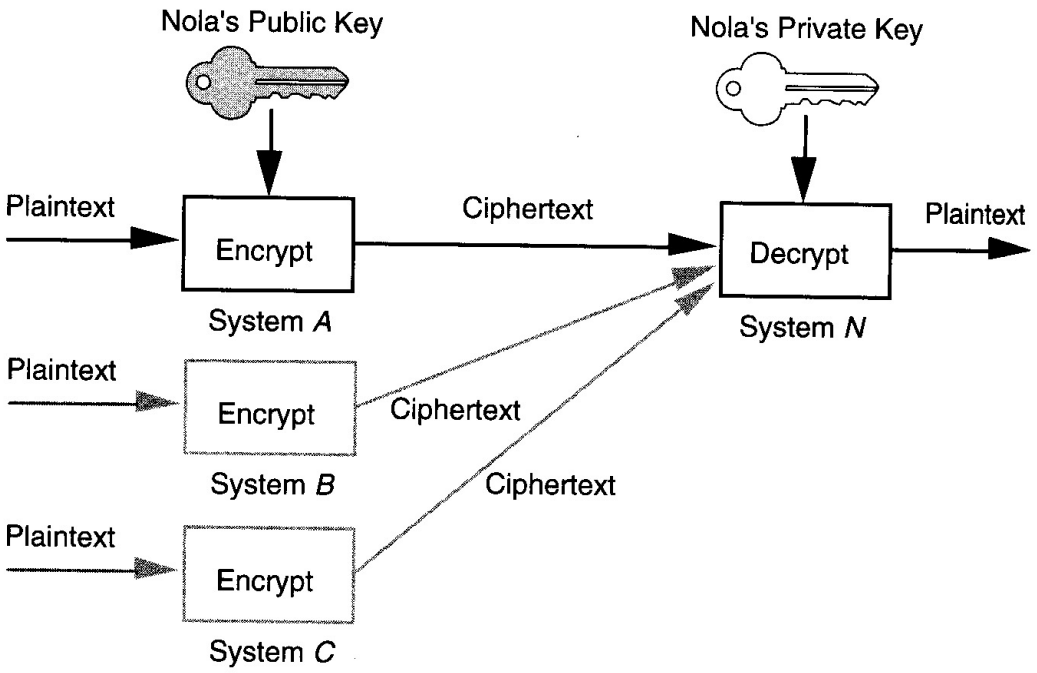


N - Users ⇨ $\frac{N \cdot (N-1)}{2}$ Keys

Users	Keys
100	5,000
1000	500,000

The Concept of Public Key Cryptography

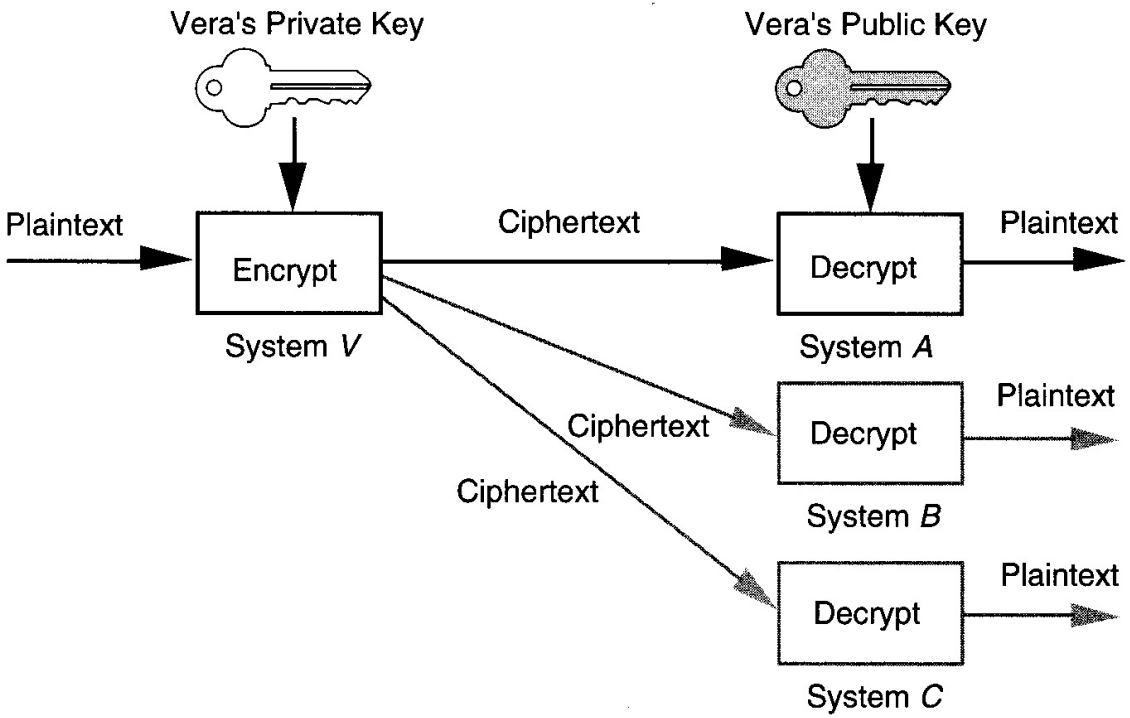
- Every participant has a pair of keys: the Public Key and Private Key
- The Public key is published or sent to everyone else in the community openly
- The Private key is kept secret by its owner
- Plaintext encrypted by A's public key can only be decrypted by A's private key
- Some Ciphertext can be decrypted by A's public key if and only if it has been encrypted by A's private key



To send Nola a secret message, any sender first finds Nola's Public Key, e.g. from a public directory, and uses it for encrypting the message.
Only the person who has Nola's private key (presumably Nola's herself) can decrypt the message successfully

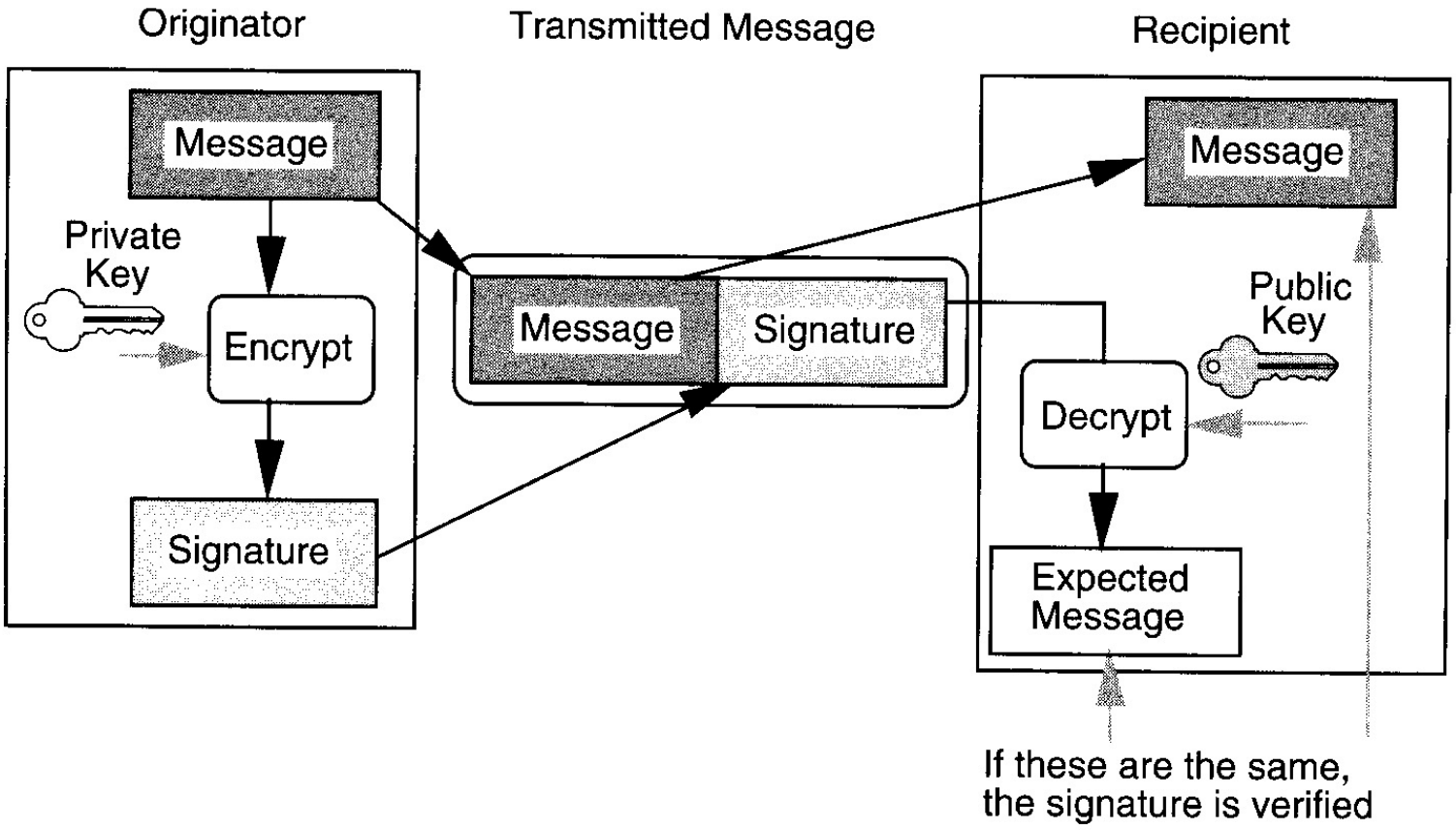
•Note: No need for secure distribution of pre-shared secret key anymore

The Concept of Public Key Cryptography (cont'd)

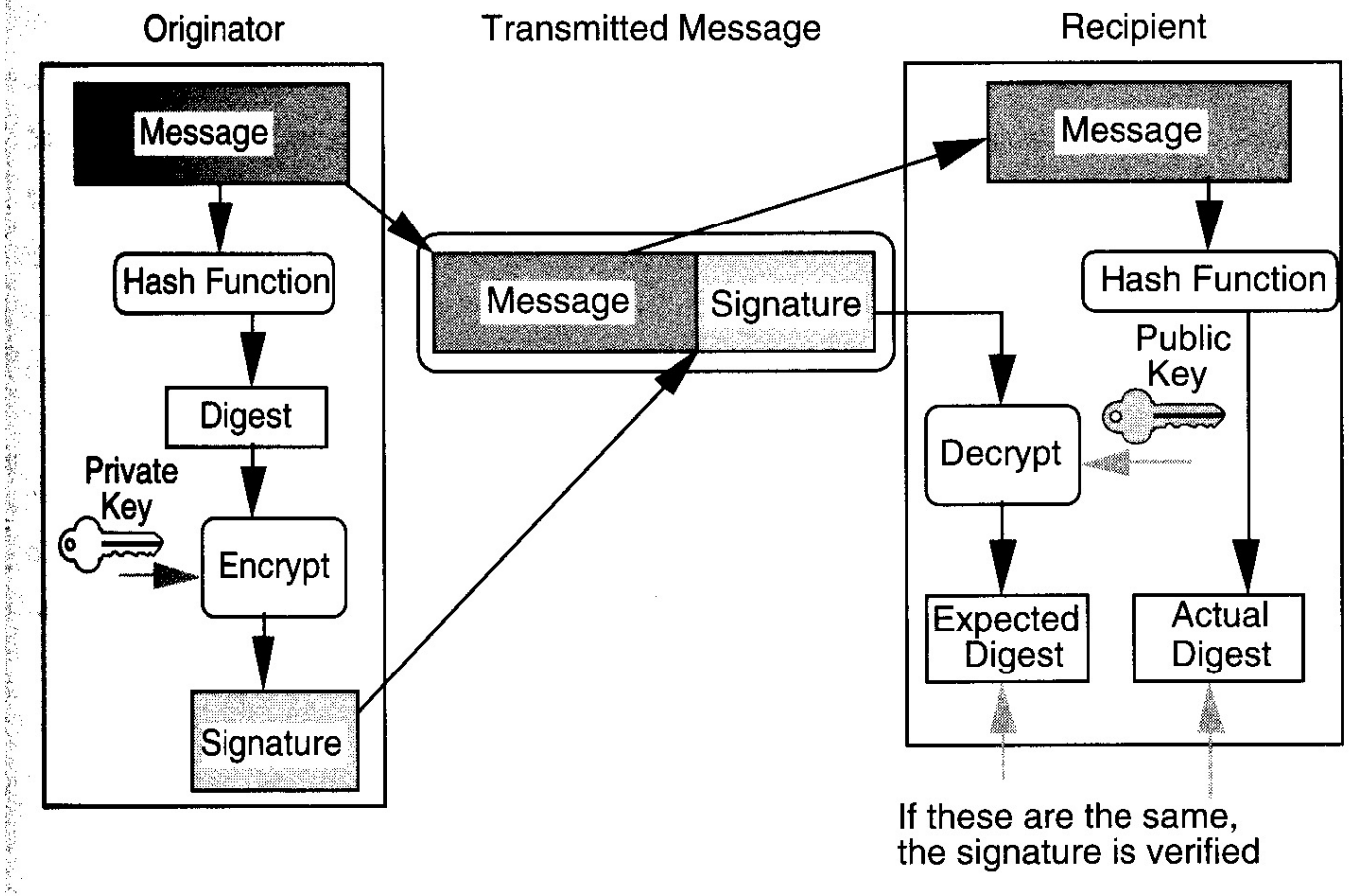


Hmm…if I can decrypt successfully an incoming message with Vera's public key, the message must have been encrypted with Vera's private key.
Since Vera is required (e.g. by law) to keep her private key secret to herself, no one but Vera could have encrypted (and sent) the message
=> This provides the notion of digital signature and thus non-repudiation service

Digital Signature using Public Key Cryptography

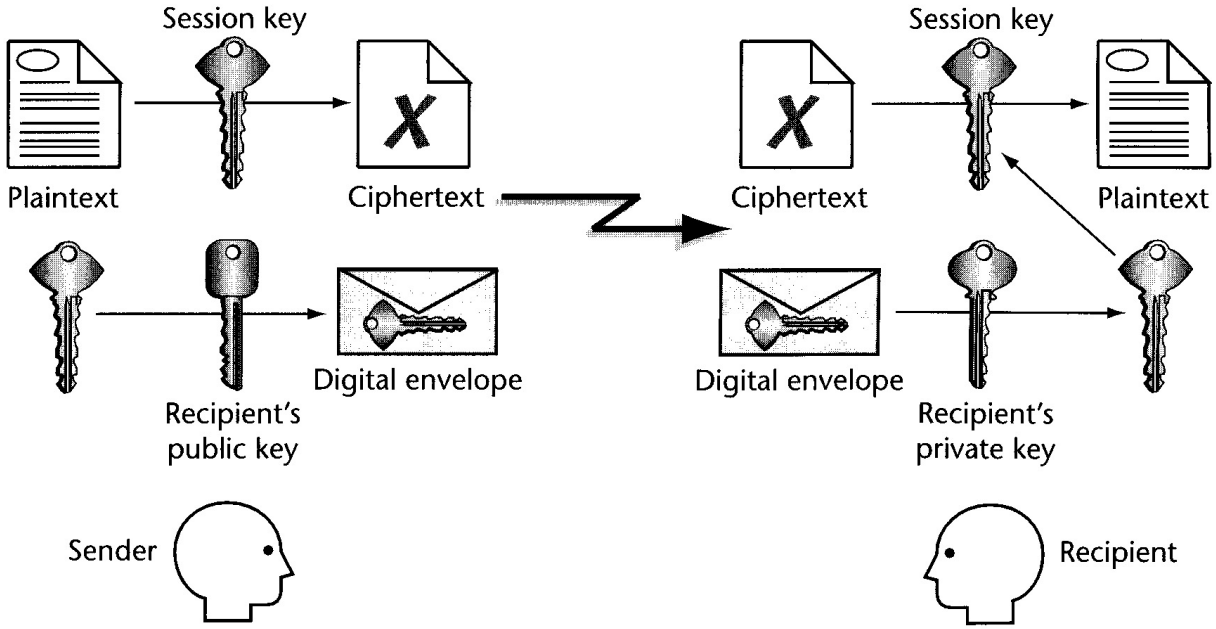


Digital Signature (cont'd)



- Instead of signing the entire message, one can sign the digest of the message to improve performance because public key algorithms are much slower than secret key ones. One should avoid using public key algorithms to encrypt large amount of data (long messages)

Use Public-key encryption to “seal” a digital envelope



- The sender picks a “secret” Session Key to encrypt the long message using a secret key algorithm, e.g. AES.
- By encrypting the session key with the Recipient’s public key, the session key can be delivered securely to the recipient without any pre-shared secret between the 2 parties
- Conversely, we can consider this as doing a secure session-key exchange using public key encryption

History of Public Key Cryptography in "public" world

- Diffie was a graduate student in Stanford, working with Prof. Hellman on solving the "key distribution problem".
- They proposed the concept of a "Public-Key Cryptosystem" (PKC). (This remarkable idea developed jointly with Merkle.) which can:
  - solve the key distribution problem of a symmetric key system and
  - Even more amazingly, introduce the notion of digital signature
- However, they were unable to find the necessary functions to realize such a system, namely, to find a pair of functions D( ) and E( ) such that:

  $D(E(m)) = E(D(m)) = m$ and

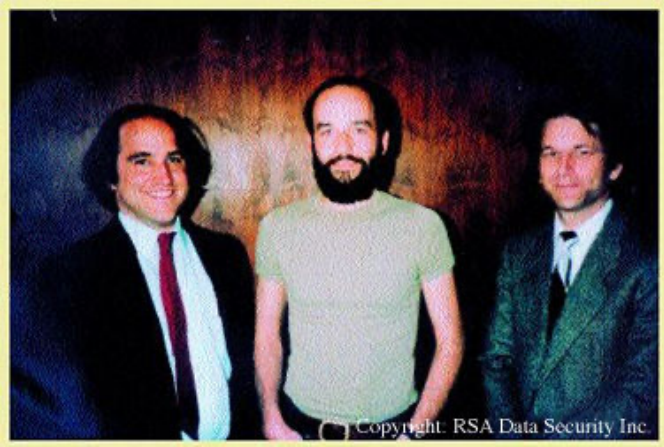
  $D( )$ can be kept secret while $E( )$ is known to the public

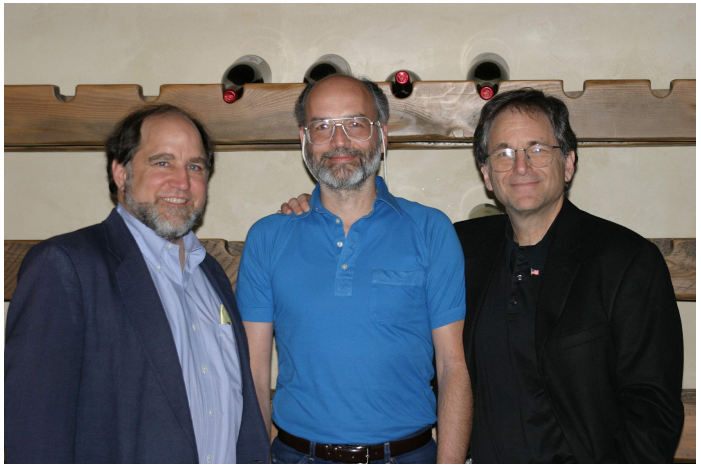
  i.e. it is computationally infeasible to derive $D( )$ by knowing $E( )$
- Instead, they were able to find a way for communication parties to establish a shared secret via open communications only

  => This is the Diffie-Hellman Key exchange algorithm

The Beginning of Public Key Cryptography in "public" world (cont'd)

- Diffie and Hellman published their ideas and findings in "New Directions in Cryptography" Nov '76, together with the open problem of realizing PKC
- Ron Rivest saw Diffie and Hellman's paper and was intrigued by it. He enlisted the help of Shamir and Adleman, all from MIT, to work on the open problem and came up with the solution in 1977 --- this is the RSA algorithm
- Diffie, Hellman, Merkle, Rivest, Shamir, Adleman were commonly recognized as the founders of Public Key Cryptography.





RSA Algorithm

- Ron Rivest, Adi Shamir, Len Adleman – found the functions and published the results in 1978:
  - *D[E[m]] = m = E[D[m]]*
- Most widely accepted and implemented approach to public key encryption
- Block cipher where m = plaintext ; and c =ciphertext are integers, between $0 <= m , c <= n-1$ for some n
- Following form:

This is the E[ ]

This is the D[ ]

$$c = m^e \bmod n$$

$$m = c^d \bmod n$$

Public key is *(n,e)*. *Private* key is *(n,d)*.

RSA: Choosing keys

1. Choose two large prime numbers *p, q.* (e.g., 1024 bits each)

2. Compute *n = pq, z = (p-1)(q-1)*

3. Choose *e (with e<n)* that has no common factors with z. (*e, z* are "relatively prime").

4. Choose *d* such that *ed-1* is exactly divisible by *z*. (in other words: *ed* –1 = *K* z* for some integer *K* , i.e. , *ed = K * z + 1,* in other words: *If ed is divided by z, the remainder is equal to 1, i.e., ed* mod *z = 1*).

5. *Public* key is *(n,e).* *Private* key is *(n,d).*

K^{+}_{pub} K^{-}_{priv}

RSA: Encryption, decryption

0. Given (n,e) and (n,d) as computed above

1. To encrypt bit pattern, m, compute
   $c = m^e \bmod n$ (i.e., remainder when m^e is divided by n)

2. To decrypt received bit pattern, c, compute
   $m = c^d \bmod n$ (i.e., remainder when c^d is divided by n)

Magic happens!

$$m = (\underbrace{m^e \bmod n}_{c})^d \bmod n$$

RSA example:

Bob chooses *p=5, q=7.* Then *n=5*7 =35,*
z=(5-1)(7-1) =24*

e=5 (so *e, z* relatively prime).
d=29 (so *ed-1 = 5*29 – 1 =144 which is* exactly divisible by z.)

encrypt:

letter	m	m^e	$c = m^e \bmod n$
D	4	$1024 = 29 * 35 + 9$	9

decrypt:

c	c^d	$m = c^d \bmod n$	letter
9	9^{29}= 4710128697246……	4	D

Use the following property to compute : $9^{29} \bmod 35$
$(a * b) \bmod n = [(a \bmod n) * (b \bmod n)] \bmod n$, i.e.
$9^{29} \bmod 35 = 9^{10+10+9} \bmod 35$
$= [(9^{10} \bmod 35) * (9^{10} \bmod 35) * (9^9 \bmod 35)] \bmod 35$
$= (16 * 16 * 29) \bmod 35 = 4$

RSA: Why is that $m = (m^e \bmod n)^d \bmod n$

Useful number theory result: If p,q prime and $n = pq$, then:

$$x^y \bmod n = x^{y \bmod (p-1)(q-1)} \bmod n$$

$$(m^e \bmod n)^d \bmod n = m^{ed} \bmod n$$

$$= m^{ed \bmod (p-1)(q-1)} \bmod n$$

(using number theory result above)

$$= m^1 \bmod n$$

(since we chose ed to be divisible by $(p-1)(q-1)$ with remainder 1)

$= m$ (since $m < n$, thus $m \bmod n = m$)

RSA: Important properties

- It is infeasible to determine d given e and n

and

$$K_{priv}(K_{pub}(m)) = m = K_{pub}(K_{priv}(m))$$

use public key first, followed by private key

use private key first, followed by public key

Result is the same!

- Ciphertext block can be as big as the key-length

=> digital signature can be as big as the key-length

How secure is RSA ?

- Brute force attack: try all possible keys – the larger the value of d the more secure
- The larger the key, the slower the system ;

Alternatively, one can break RSA by finding p and q, and thus d by knowing n and e

- However, for large n with large prime factors, factoring is a hard problem
- Cracked in 1994 a 428 bit key; $100
- **Used to be Bounty offers by RSA (but RSA ended the Bounty in 2007):**

  https://en.wikipedia.org/wiki/RSA_Factoring_Challenge

  - Bounty for RSA-576 and RSA-640 were cracked in 2003 and 2005.
  - RSA-704 cracked in 2012 ; RSA-768 in 2009 [both after RSA ended its Bounty programme]

$100 RSA Scientific American Challenge

- Martin Gardner publishes Scientific American column about RSA in August '77, including the RSA $100 challenge (129 digit , or about 430-bit n) and the infamous "40 quadrillion = $40*10^{15}$ years" estimate required to factor
- RSA-129 = 114,381,625,757,888,867,669,235,779,976,146,612,010,218,296, 721,242,362,562,561,842,935,706,935,245,733,897,830,597,123, 563,958,705,058,989,075,147,599,290,026,879,543,541

(129 digits)

- or to decode encrypted message.
- RSA-129 was factored in 1994, using thousands of computers on Internet, using 5000 MIPS-years (1GHz Pentium PC ~= 250 MIPS)

  "The magic words are squeamish ossifrage."
- Cheapest purchase of computing time ever!
- Gives credibility to difficulty of factoring, and helps establish key sizes needed for security.

Other Factoring milestones

- ’ 84: 69D (D = “decimal digits”) (Sandia; Time magazine)
- ’91: 100D = 332 bits (using Quadratic Sieve techniques)
- ’94: 129D = 428 bits ($100 challenge number) (Distributed QS, 8 months, 5000MIPS-year) ; [Ref: 1GHz Pentium PC ~= 250 MIPS]
- ’99: 155D = 512 bits; (Generalized Number Field Sieve techniques, 2 months and 10 days, 8000-MIPS-year)
  - 512-bit RSA Backdoor in Quicken files for recovery service by Intuit ; Elcomsoft is able to offer a competitive service => cracked !
- ’01: 15 = 3 * 5 (4 bits; IBM quantum computer!)
- Dec 2003: 576-bit cracked
- Nov. 2005: 640-bit cracked
- Dec. 2009: 768-bit cracked
- Dec 2019: 795-bit cracked
- Feb 2020: 829-bit cracked

Recommended Key Sizes for RSA

- Old Standard:
  - Short term protection: 1024-bits (308 decimal digits)
  - Long term protection: 2048-bits (616 decimal digits)

Ref: No. of operations required to crack 512-bit RSA with best known attack = 1/50 * N_{DES}

where N_{DES} is the no. of operations required to crack 56-bit DES by brute-force key-enumeration

- Starting 2024, 2048-bit RSA keys are no longer VS-NfD compliant. The Federal Office for Information Security (BSI) recommended the use of at least 3000-bit RSA keys since 2023.
  - The use of RSA keys with length of 2048-bit was permitted for a “transition period until end of 2023”.

Implementation Aspects of RSA

- How to find the big primes p and q ?
  - Generate random numbers and test for their primality using known testing algorithms
  - How many times (numbers) one need to try before finding a prime no. ?
    - For a randomly chosen no. N, the probability of it being prime ~= 1/ ln N ; => need to try ln N times on average
    - For a 100-digit number, one 1 in 230 chance.
- e can be fixed to some constant value without decreasing security ;
- e is commonly set to 3 or 65537 = $2^{16}+1$ in practice to speed up encryption: $m^e \bmod n$; one can compute m^{65537} quickly as well
- Once e is fixed, d can be found using the Euclid's Algorithm
- **Not so Recent News: (Feb 15, 2012): Implementation Flaws in RSA random key generations**
  http://www.nytimes.com/2012/02/15/technology/researchers-find-flaw-in-an-online-encryption-method.html?_r=1&hp

Some arcane Attacks on RSA

- Guessing plaintext attack: if the attackers know the candidate set of plaintexts to be sent (with exact wordings), the attacker can encrypt each of the possible choice using the recipient's public key and compare them to the actual ciphertext sent ;
- Chosen ciphertext attack: don't sign arbitrary messages sent by others because signing is equivalent to decrypt the message with your private key.
  - Assume you are using the a single pair of public and private key, (Kpub,Kpriv) for both encryption/decryption and signing/verification.

    Eve, the attacker, records an encrypted letter sent to you by someone else, and ask you to sign this recorded message (and of course, return the signed result to her). If you follow Eve's request and sign on what Eve gives you, you are actually decrypting your own secret letter for Eve.

  => It's better to use different public/private key-pairs for different purposes, e.g. one key-pair **(Kpub1, Kpriv1)** for letting people to send secret to you by encrypting with **Kpub1** and you can decrypt using **Kpriv1**; use a different pair **(Kpub2,Kpriv2)** for digital-signature/verification, i.e. you use **Kpriv2** to sign outgoing messages and your intended receiptant can use **Kpub2** to verify your signature.
- Cube-root attack for $e = 3$: if $m^3 < n$ because the "mod" operation becomes null , i.e. $m^3 \bmod n = m^3 = C$ and the attacker can obtain m by performing $m = \sqrt[3]{C}$
- With $e = 3$, sending exactly the same secret message to 3 or more people (using 3 or more public keys) would reveal the secret message ;
- See https://crypto.stanford.edu/~dabo/papers/RSA-survey.pdf and http://members.tripod.com/irish_ronan/rsa/attacks.html

Public Key Cryptography Standard (PKCS)

- A list of Standards (PKCS#1 to PKCS#15) on how to use RSA in practice, regarding message formatting, information encoding scheme, choice of parameters etc
- Protected against the following "improper use" or attacks on RSA including:
  - Plaintext guessing
  - Chosen ciphertext attack
  - $m^3 < n$
  - Sending the same message to multiple people ;
- This is done by pre-pending some fixed number of constant and random bytes to the message to be encrypted/ decrypted
- WARNING: WITHOUT proper preprocessing of the plaintext, Textbook RSA (as well as El Gamal) Encryption are Insecure !!

Reference:

- Dan Boneh et al, "Why Textbook ElGamal and RSA Encryption are Insecure," AsiaCrypt 2000.
- Dan Boneh, "Twenty Years of Attacks on RSA Cryptosystem," Notices of American Math. Society, 1999, https://crypto.stanford.edu/~dabo/papers/RSA-survey.pdf

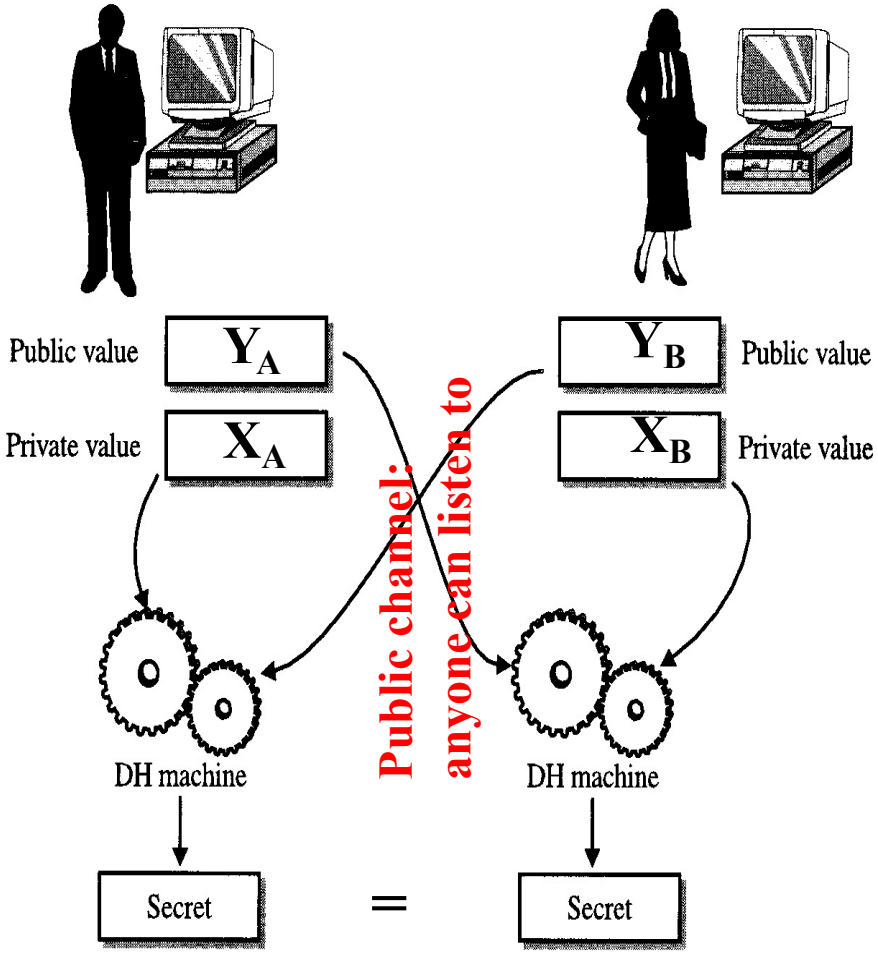
Performance of RSA

- For hardware implementation, RSA is about 1000 times slower than DES ; for software implementation, RSA is about 100 times slower ;
- Time to do RSA decryption on a 1 MIPS VAX was around 30 seconds (VERY SLOW…) when it was invented in late 70's
- The inventors needed to work on efficient special-purpose implementation (e.g. special circuit board, and then the "RSA chip", which did RSA in 0.4 seconds) to prove practicality of RSA.
- IBM PC debuts in 1981 and Moore's Law to the rescue---software now runs 2000x faster…

  also, software and the Web rule…now ;
- Speed differs on types of operations, (i.e. encryption, decryption, digital signing and signature verification), as well as relatively size of e and d ;
  - e.g. with $e = 3$, encryption and signature verification are typically *much (5-10 times) faster* than decryption and digital signing respectively ; Why not make $d = 3$ instead ?

Diffie-Hellman Key Exchange

- Diffie-Hellman key-exchange enables two users to establish a shared secret key securely using an open/ public communications channel.



Global Public Elements	
q	prime number
α	$\alpha < q$ and α a primitive root of q

User A Key Generation	
Select private X_A	$X_A < q$
Calculate public Y_A	$Y_A = \alpha^{X_A} \bmod q$

User B Key Generation	
Select private X_B	$X_B < q$
Calculate public Y_B	$Y_B = \alpha^{X_B} \bmod q$

- $(Y_B)^{X_A} \bmod q = \alpha^{X_B X_A} \bmod q = Secret = \alpha^{X_A X_B} \bmod q = (Y_A)^{X_B} \bmod q$

Diffie-Hellman Key Exchange

- enables two users to establish a shared secret key via an open/ public communications channel.

- Choose a prime number q, and α ($<$ q and is a primitive root of *q*); both made public

- Alice randomly chooses X_A in {2, …, q-1} as her secret; send Bob: $Y_A = \alpha^{X_A} \pmod q$
- Bob randomly chooses X_B in {2, …, q-1} as his secret; send Alice: $Y_B = \alpha^{X_B} \pmod q$
- Shared key $K_{AB} = (\alpha^{X_A})^{X_B} = (\alpha^{X_B})^{X_A}$

Diffie-Hellman Example

- users Alice & Bob who wish to swap keys:
- agree on prime $q=353$ and $\alpha=3$
- select random secret keys:
  - A chooses $x_A=97$, B chooses $x_B=233$
- compute respective public keys:
  - $y_A=3^{97} \mod 353 = 40$ (Alice)
  - $y_B=3^{233} \mod 353 = 248$ (Bob)
- compute shared session key as:
  - $K_{AB}= y_B^{x_A} \mod 353 = 248^{97} \mod 353 = 160$ (Alice)
  - $K_{AB}= y_A^{x_B} \mod 353 = 40^{233} \mod 353 = 160$ (Bob)

How secure is Diffie-Hellman Key Exchange ?

- It relies on the fact that "Discrete Logarithm" is a computationally difficult problem, i.e.:

Knowing that $Y_A = a^{X_A} \bmod q$ and the values of a, q and Y_A

It is still computationally difficult to find X_A

- But still subject to Man-in-the-Middle Attack !! Because Alice does not know for sure if it's actually Bob who is sending her the Y_B
  - Remedy: Published those public numbers, i.e. *a*, q and Y_A , Y_B in a "Trusted, publicly accessible directory for each person"
    - This also allows Alice to send Bob an encrypted message even when he is currently offline.
  - But how can you be sure that you are looking at the directory hosted by the "true trusted directory server" ?

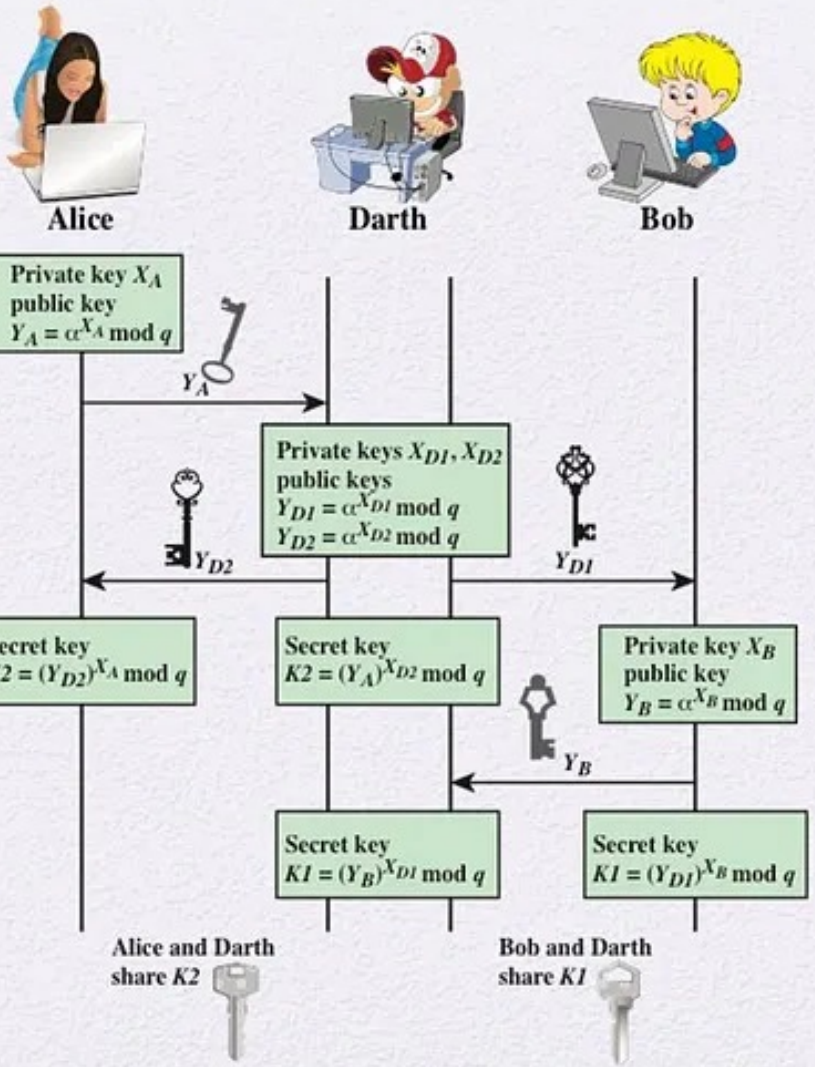
Man-in-the-middle (MITM) Attack

- DH protocol:

  1. Alice -> Bob: $\alpha^x \pmod q$
  2. Bob -> Alice: $\alpha^y \pmod q$

- Attack scenario ?

Man-in-the-middle (MITM) Attack



Source:
https://medium.com/@dillihangrae/understanding-diffie-hellman-key-exchange-and-man-in-the-middle-attacks-f9b08abe2c20

- What is the Root cause of this Vulnerability ? Lack of what?

Other Public Key Algorithms

- 1978: Merkle/Hellman (Knapsack), subsequently found to be insecure
- 1985: El Gamal (Discrete logarithm Problem)
- 1985: Miller/Koblitz (Elliptic curves)
- 1991: Digital Signature Standard (DSS) (Discrete logarithm Problem)

And many others, too

El Gamal

- El Gamal can be considered to be a generalization of Diffie-Hellman key-exchange algorithm => still relies on the difficulty of doing discrete logarithm:

$$y = \alpha^x \bmod q$$

  - q is prime ;
  - α and x are +ve integers < q and α is a primitive root of q and $0 < x < q-1$
  - Public key = (y, α, q) ; Private key = x

- Encryption of plaintext message M (< q):
  - Select k: $1 \le k \le q-2$
  - $C1 = \alpha^k \bmod q$
  - $C2 = (y^k M) \bmod q$
  - Ciphertext = (C1,C2)
- Decryption:
  - $M = [C2 * (C1^x)^{-1}] \bmod q$

where

$b^{-1} \pmod q$ is the "multiplicative inverse" of b (mod q), i.e.

$$[b * b^{-1}] \bmod q = 1 \bmod q ;$$

El Gamal

- Encryption of plaintext message M (< q):
  - Select k: 0< k < q, relatively prime to (q-1)
  - C1 = α^k mod q
  - C2 = (y^kM) mod q
  - Ciphertext = (C1,C2)
- Decryption:
  - M = [C2 * $(C1^x)^{-1}$] mod q

Proof: [C2 * $(C1^x)^{-1}$]mod q = [y^k M * $(C1^x)^{-1}$]mod q

= [α^{kx} M * $(C1^x)^{-1}$]mod q = [$C1^x$ * M * $(C1^x)^{-1}$] mod q = M mod q = M

because y = α^x mod q

and y^k mod q = α^{kx} mod q = $C1^x$

where

b^{-1} (mod q) is the "multiplicative inverse" of b (mod q), i.e.

[b*b^{-1}] mod q = 1 mod q ;

e.g.

8^{-1} (mod 17) = 15 (mod 17) because (8 * 15) mod 17 = (17*7+1) mod 17 = 1

We can use Fermat's little theorem to find b^{-1} mod q :

If q is prime and q does not divide b, then b^{-1} mod q = b^{q-2} mod q

El Gamal - an example

$q = 11$, $\alpha = 2$, $x = 3$ => $y = 2^3 \bmod 11 = 8$;

⇒ Public Key of recipient = (y, α , q) = (8, 2, 11)

- Encryption of plaintext message M= 7 (< q):
  - Select k = 4 : $1 \le k \le q-2$
  - $C1 = \alpha^k \bmod q = 2^4 \bmod 11 = 5$
  - $C2 = (y^k M) \bmod q = [8^4 (7)] \bmod 11 = (4096 * 7) \bmod 11 = 6$
  - Ciphertext = (5,6)

- Decryption:
  - $M = [C2 * (C1^x)^{-1}] \bmod q = [6 * (5^3)^{-1}] \bmod 11 = (6 * 3) \bmod 11 = 7$

because

$(5^3)^{-1} \bmod 11 = (5^3)^{11-2} \bmod 11 = (125 \bmod 11)^9 \bmod 11 = [(4^3 \bmod 11)^3] \bmod 11 = 9^3 \bmod 11$

$= 729 \bmod 11 = 3 \bmod 11$

El Gamal

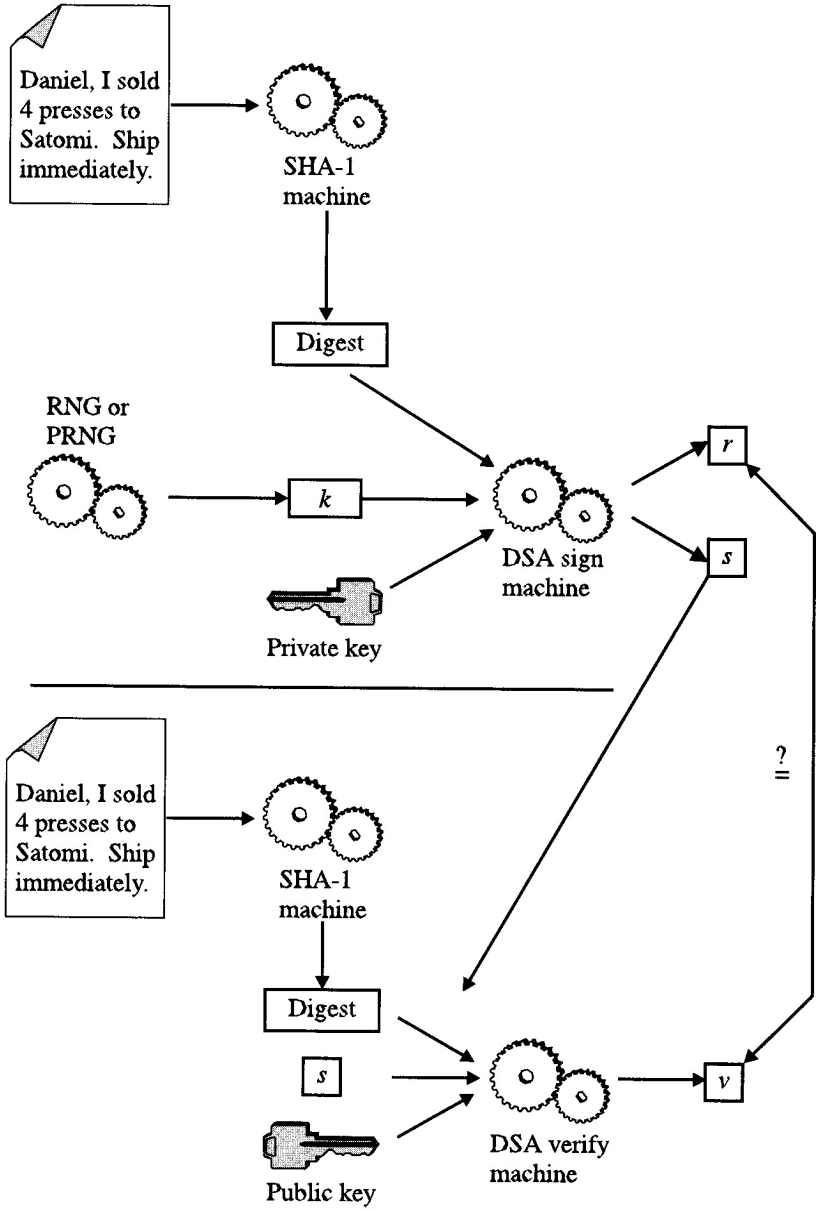
- El Gamal can be considered to be a generalization of Diffie-Hellman key-exchange algorithm => relies on the difficulty of doing discrete logarithm:

$$y = \alpha^x \bmod q$$

- Advantages:
  - support both encryption and digital signature
  - Not patented (but someone claims it is covered by the DH patent)
- Drawbacks:
  - The ciphertext (or digital signature) is about twice as big as the plaintext (or message digest to be signed on)
- The scheme was never popular in practice
- The Digital Signature Algorithm (DSA) used in the US Digital Signature Standards (DSS) was a variant/ or based on the El Gamal's scheme ;
- The core ideas of the EL Gamal scheme can be generalized for the design of encryption and digital signing algorithm for ECC public key crypto systems.
- The inventor, Taher El Gamal, also from Stanford was Netscape's Director of Security at one point ; aka "Father of SSL" ; PhD graduate of Prof. Hellman
  - https://en.wikipedia.org/wiki/Taher_Elgamal

Digital Signature Standard (DSS)

- In 1991, NIST in US standardized Digital Signature Standard (DSS). SHA-1 is used to first compute the message digest which is then signed by the Digital Signature Algorithm (DSA).
- DSA is based on a variant of El Gamal digital signature, thus also inherits it's "size-doubling" property => SHA-1 digest is 160-bit long, the DSA signature is 320 bits long: signature = (r,s).
- Since DSA does not support encryption by design, it avoids US technology-export concerns.



Elliptic Curve Cryptosystems (ECC)

- Independent proposed by Koblitz (U. of Washington) and Miller (IBM) in 1985
- Depends on the difficulty of the elliptic curve logarithm problem
  - fastest method is "Pollard rho method"
  - Best attacks for discrete logarithm problem do NOT apply to elliptic curve logarithm problem
- The first true alternative for RSA
- ECC is beginning to challenge RSA in practical deployment in selected areas: embedded, wireless/mobile systems
- It is a family of cryptosystems instead of a single one:
  - ECC replaces Modulo Exponentiation by Elliptic Curve Multiplication

  and
  - ECC replaces Modulo multiplication by Elliptic Curve Addition

  Can then be applied directly to Diffie-Hellman, El Gamal and DSA to yield ECC Diffie-Hellman (ECDH), ECC-ElGamal and ECC-DSA algorithms to support key exchange, encryption and digital signature respectively
- Certicom (http://www.certicom.com, a canadian-based company, is one of the leading companies for ECC commercialization

ECC Vs. RSA

ECC

- Shorter keys (equivalent key sizes: ~150bits Vs. 1024bits of RSA) and thus, shorter signature as well.
- Fast and compact implementations, especially in hardware

=> Advantageous in environments with limited bandwidth and storage, e.g. wireless applications, smartcards, embedded systems

- Shorter history of cryptanalysis (since early 90's)
- Complex mathematical description
- No patents for the cryptosystems themselves, but many on the implementation optimization
- Shorter signature generation time
- Shorter key generation time
- Larger no. of operations for attacks against the algorithm

RSA

- Proven technology,
- Widely deployed and used in a large set of general applications
- Efficient software implementation
- Longer history of cryptanalysis (since late 70's)
- Patent expired in 2000
- Shorter signature verification time
- Larger Memory requirements for attacks against the algorithm

ECC Vs. RSA (cont'd): Equivalent Key-size to support same level of Security

Elliptic Curve PKC	
Key Size	**MIPS-Years to Crack**
150	3.8×10^{10}
205	7.1×10^{18}
234	1.6×10^{28}

RSA PKC	
Key Size	**MIPS-Years to Crack**
512	3×10^{4}
768	2×10^{8}
1024	3×10^{11}
1280	1×10^{14}
1536	3×10^{16}
2048	3×10^{20}

Example:

Equivalent key-sizes given current acceptable security level of 4.12×10^{12} MIPS-year:

RSA : ECC : Symmetric cipher, (e.g. AES) = 1024:163:79

[Ref: 1GHz Pentium PC ~= 250 MIPS]

Relative Performance: ECC Vs. RSA (cont'd)

Estimated Relative Time units of Encryption/Decryption and/or Key-exchange (source: RSA)				
	RSA	DH	ECC	ECC with acceleration
Initiate contact (Public Key)	1	32	18	N/A
Receive message (Private Key)	13	16	6	2

Estimated Relative Time units of Digital signing and verification (source: RSA)				
	RSA	DSA	ECC	ECC with acceleration
Sign (Private Key)	13	17	7	2
Verify (Public Key)	1	33	19	N/A

Some Cryptographic predictions by the S. of RSA:

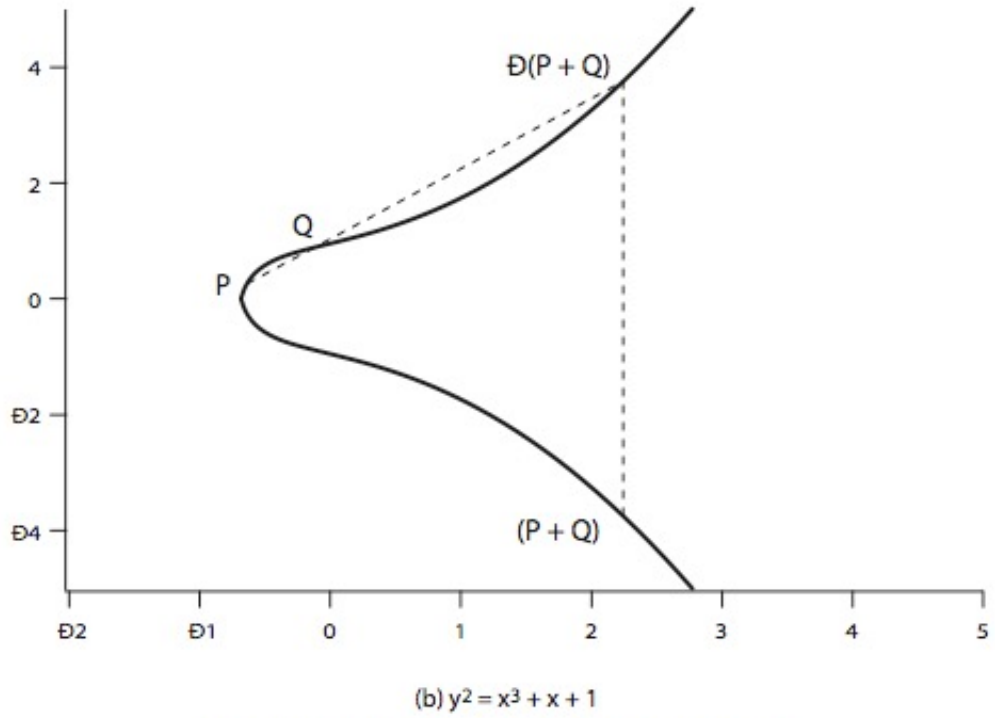
- AES will remain secure for the forseeable future
- Some PK schemes and key sizes will be successfully attacked in the next few years
- Crypto will be invisibly everywhere
- Vulnerabilities will be visibly everywhere
- Crypto research will remain vigorous, but only its simplest ideas will become practically useful
- Non-crypto security will remain a mess

Backup Slides

Real Elliptic Curves

- an elliptic curve is defined by an equation in two variables x & y, with coefficients
- consider a cubic elliptic curve of form
  - $y^2 = x^3 + ax + b$
  - where x,y,a,b are all real numbers
  - also define zero point O
- have addition operation for elliptic curve
  - geometrically sum of P+Q is reflection of intersection R

Real Elliptic Curve Example



(b) $y^2 = x^3 + x + 1$

Finite Elliptic Curves

- Elliptic curve cryptography uses curves whose variables & coefficients are finite
- have two families commonly used:
  - prime curves $E_p(a,b)$ defined over Z_p
    - use integers modulo a prime
    - best in software
  - binary curves $E_{2^m}(a,b)$ defined over GF(2^n)
    - use polynomials with binary coefficients
    - best in hardware

Elliptic Curve Cryptography

- ECC addition is analog of modulo multiply
- ECC repeated addition is analog of modulo exponentiation
- need “hard” problem equiv to discrete log
  - `Q=kP`, where Q,P belong to a prime curve
  - is “easy” to compute Q given k,P
  - but “hard” to find k given Q,P
  - known as the elliptic curve logarithm problem
- Certicom example: $E_{23}(9,17)$

Recall Diffie-Hellman Key Exchange

- enables two users to establish a shared secret key via an open/ public communications channel.

- Choose a prime number q, and α (< q and is a primitive root of *q*); both made public

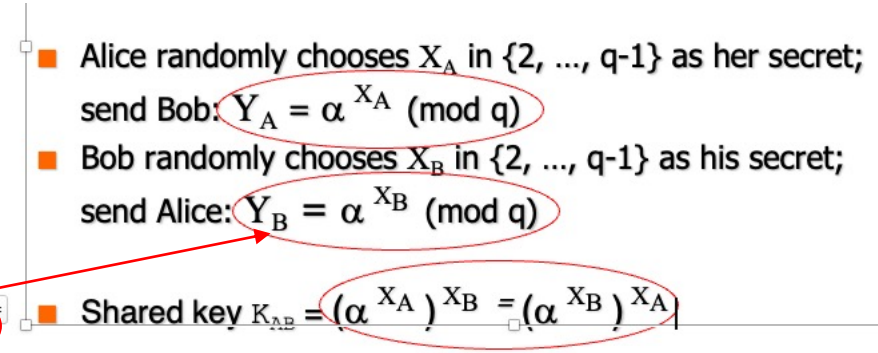
- Alice randomly chooses X_A in {2, …, q-1} as her secret; send Bob: $Y_A = \alpha^{X_A} \pmod q$
- Bob randomly chooses X_B in {2, …, q-1} as his secret; send Alice: $Y_B = \alpha^{X_B} \pmod q$

- Shared key $K_{AB} = (\alpha^{X_A})^{X_B} = (\alpha^{X_B})^{X_A}$

ECC Diffie-Hellman

- Readily can support Key Exchange via analogy to D-H:
  - ECC Multiplication = ECC repeated addition is analog of modulo exponentiation

- users select a suitable curve $E_p(a,b)$
- select base point $G=(x_1, y_1)$
  - with large order n s.t. $nG=O$
- A & B select private keys $X_A<n$, $X_B<n$
- compute public keys: $Y_A=X_AG$, $Y_B=X_BG$
- compute shared key: $K=X_AY_B$, $K=X_BY_A$
  - same since $K=X_AX_BG$



ECC Encryption/Decryption

- ECC addition is analog of Modulo Multiply
- ECC repeated addition is analog of Modulo Exponentiation

- several alternatives, will consider simplest:
- must first encode any message M as a point on the elliptic curve P_m
- select suitable curve & point G as in D-H
- each user chooses private key $X_A<n$
- and computes public key $Y_A=X_AG$
- When encrypting a message M, we get 2 pieces of ciphertext: $\{C1=kG, C2=M+kY_A\}$, where k is some random number.
- To retrieve M, we decrypt C1 and C2 together by computing:

$$M+kY_A-X_A(kG) = M+k(X_AG)-X_A(kG) = M$$

- El Gamal still relies on the difficulty of Discrete Logarithm, the i.e. one-way nature of the following function :
  - $y = \alpha^x \bmod q$
  - q is prime ;
  - α and x are +ve integers < q and
  - α is a primitive root of q and $0 < x < q-1$
  - Public key = (y, α, q) ; Private key = x
- Encryption of plaintext message M (< q):
  - Select k: $1 \le k \le q-2$
  - $C1 = \alpha^k \bmod q$
  - $C2 = (y^k M) \bmod q$
  - Ciphertext = (C1,C2)
- Decryption:
  - $M = [C2 * (C1^x)^{-1}] \bmod q$