Introduction into Cyber Security

Chapter 4: Asymmetric Cryptography

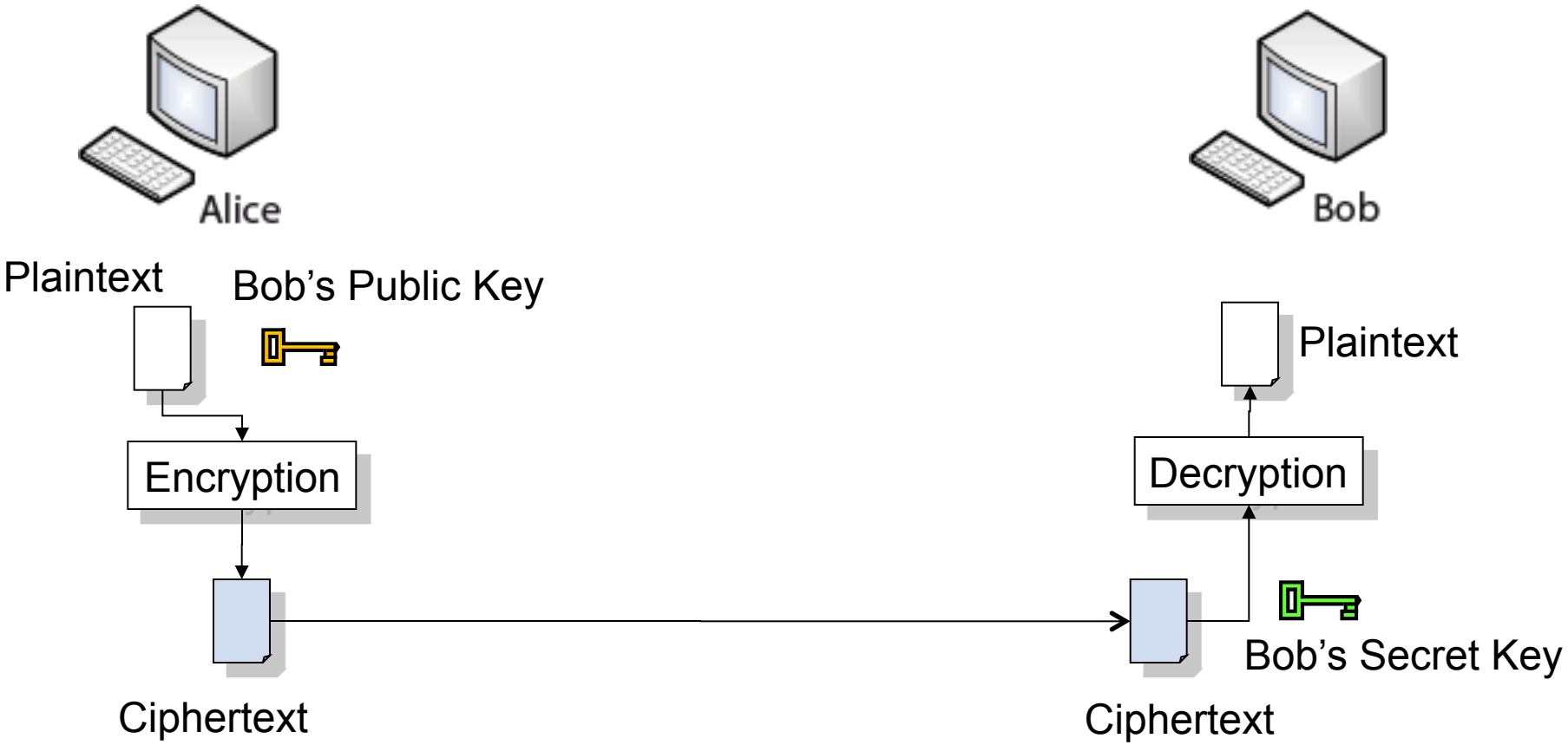
WiSe 18/19

Chair of IT Security

Chapter Overview

- Asymmetric Encryption
  - General Idea of Asymmetric Encryption
  - Modular Arithmetic
  - RSA
- Digital Signatures
  - General Idea of Digital Signatures
  - RSA-based signatures
  - Digital Signature Standard
- Diffie-Hellman Key-Agreement

General Idea of Asymmetric encryption



Note: There needs to be a mechanisms that ensures that Alice gets into possession of Bob's public key

Modular Arithmetic (1)

- Let n be a positive integer ≠ 0, then for any integer k
  - k mod n is defined as the remainder of k divided by n
    - Example: 10 mod 7 = 3
- We define addition and multiplication of the numbers {0,…, n-1} by
  - a + b := (a+b) mod n
  - ab := (ab) mod n
- Then
  - For all a, b, c in {0,…, n-1}: (a+b) + c = a + (b + c)
  - For all a, b, c in {0,…, n-1}: a + b = b + a
  - For all a in {0,…,n-1}: a + 0 = a
  - For all a in {0,…, n-1} there is an element x in {0,…, n -1} with a + x = 0. This element x is called the additive inverse of a mod n and is denoted as “-a”

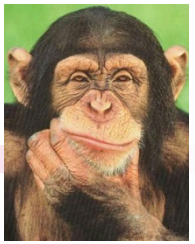
Addition and multiplication of mod n: Formula:
x is called additive inverse of a mod n, if x+a=0, then x=-a,

if we get ax=1 mod n, so, x is called inverse of a,then x=a^-1 mod n

Modular Arithmetic (2)

- And
  - For all a, b in {1,…,n-1}: ab = ba
  - For all a, b, c in {1,…,n-1}: (ab)c = a(bc)
  - For all a in {1,…,n-1}: a1 = a
- If for an a in {1,…,n-1} there is an x in {1,…,n-1} with
  - ax = 1 mod n
- Then a is called invertible mod n and x is called the inverse of a and x is denoted as a^{-1} Important Slides

Example: Addition and Multiplication mod 6

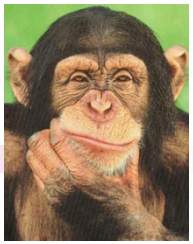


+	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4

*	1	2	3	4	5
1	1	2	3	4	5
2	2	4	0	2	4
3	3	0	3	0	3
4	4	2	0	4	2
5	5	4	3	2	1

1 and 5 are invertible mod 6
2, 3, and 4 are not

Example: Addition and Multiplication mod 5



+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

*	1	2	3	4
1	1	2	3	4
2	2	4	1	3
3	3	1	4	2
4	4	3	2	1

$2^{-1} = 3$, $3^{-1} = 2$,
$1^{-1} = 1$, $4^{-1} = 4$

Modular Exponentiation

- For all a in {0,…,n-1} a^k mod n is defined as

$$a^k \bmod n = \underbrace{a \cdots a}_{k \text{ times}} \bmod n$$

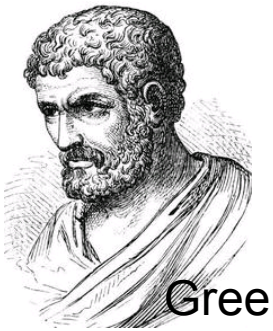
Important Slides

Multiplicative Inverses in General

- An integer k has a multiplicative inverse mod n iff k and n are relatively prime
  - E.g. in mod 10: 1, 3, 7, and 9 are invertible while 2, 4, 5, 6, and 8 are not
- Relatively prime means that k and n do not share any prime divisors
- The number of integers relatively prime to n is called $\varphi(n)$
- If n is prime, then all integers $\neq 0$ are invertible mod n
  - $\varphi(n) = n-1$
- If n is the product of two different prime numbers p, q ($n = pq$)
  - Then: $\varphi(n) = (p-1)(q-1)$
- The set of integers that are invertible mod n are denoted Z^*_n
  - E.g. $Z^*_5 = \{1, 2, 3, 4\}$, $Z^*_4 = \{1, 3\}$, $Z^*_{10} = \{1, 3, 7, 9\}$

What Euclid's Algorithm Does and How

- Allows to compute the greatest common divisor (gcd) of two integers
- Two integers are relatively prime iff their gcd = 1 Important Slides
- With Euclid's algorithm one can therefore check if an integer k has an inverse mod n by checking if gcd(k,n) = 1 Important Slides
  - Example how Euclid's algorithm works for 408 and 595:
    - 595:408 = 1 remainder 187
    - 408:187 = 2 remainder 34
    - 187:34 = 5 remainder 17
    - 34:17 = 2 remainder 0
  - ➢ gcd(408,595) = 17



Greek mathematician
360 BC to 280 BC

Why does Euclid's Algorithm Work?

- gcd(k,n) = gcd(k-n,n) Important Slides
  - If d divides k and n, then k = jd, n = ld for some j, l, therefore k - n = (j - l)d
  - Vice versa if d divides n and k - n, then n = yd and k - n = xd for some y, x, therefore k = k - n + n = (x + y)d
- Repeating this several times gives us
  - gcd(k,n) = gcd(k mod n, n) Important Slides
- Repeating this with k mod n instead of n and n instead of k now gives us
  - gcd(k mod n, n) = gcd(k mod n, n mod (k mod n)) Important Slides
- We continue doing this until the remainder is 0
- The remainder before the 0 is then the gcd Important Slides

Euclid's Algorithm in Formulas

- Goal: compute gcd(k,n)
- Initial Setup
  - Let $r_0 = n$, $r_1 = k$, $i = 0$
- Step i
  - If $r_{i+1} = 0$, then $\gcd(k,n) = r_i$
  - Else, divide r_i by r_{i+1} to get quotient q_{i+1} and remainder r_{i+2}
  - Set i = i+1 and repeat
- Note: if we additionally set
  - $u_0 = 1$, $u_1 = 0$, $v_0 = 0$, $v_1 = 1$ in the initial step
  - and then $u_{i+1} = u_{i-1} - q_i u_i$ and $v_{i+1} = v_{i-1} - q_i v_i$ for $i > 0$
- Then: if $r_{l+1} = 0$, then $r_l = \gcd(k,n)$ and
  - $\gcd(k,n) = u_l n + v_l k$ Important Slides

Extended Euclidean Algorithm

Important Slides

- gcd(81, 57) = 3.
- 81 = 1(57) + 24
  57 = 2(24) + 9
  24 = 2(9) + 6
  9 = 1(6) + **3**
  6 = 2(3) + 0.
- **3** = 9 -1(6)
  = 9 - 1(24 - 2(9)) = 3(9) - 1(24)=
  3(57 - 2(24)) - 1(24) = 3(57) - 7(24)=
  3(57) - 7(81 - 1(57)) = 10(57) -7(81)

Compute the Inverse with Euclid

- If k and n are relatively prime, then Euclid's algorithm gives us u, v with
  - 1 = uk + vn
- As a consequence: u is the inverse of k mod n

Important Slides

Key Generation

- The key problem of the development of asymmetric encryption methods is to find functions for the generation of the key pairs which ensure that the private key cannot be calculated efficiently from the knowledge of the public key.
- → Mathematical class of one-way functions
- One-way function
  - An injective function f: X → Y is called a one-way function, if
  - (1) the function value y = f(x) is efficiently computable for all x ∈ X and
  - (2) there is no efficient method of calculating x from y = f(x).
- Analogy: breaking a plate

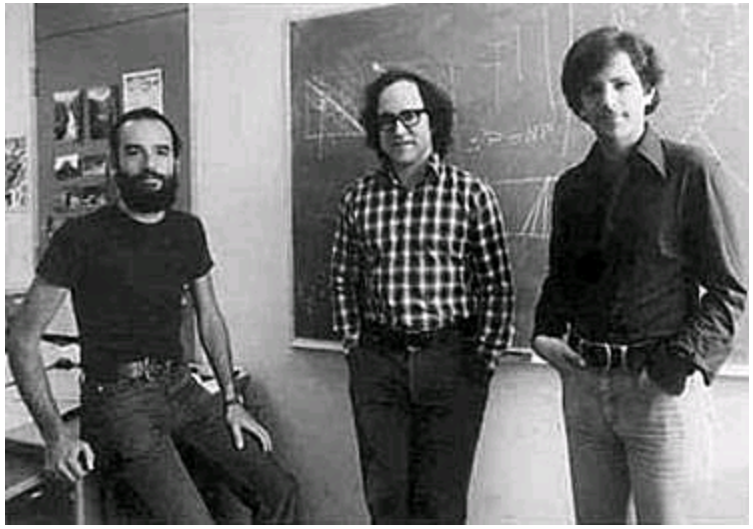
Key Generation (2)

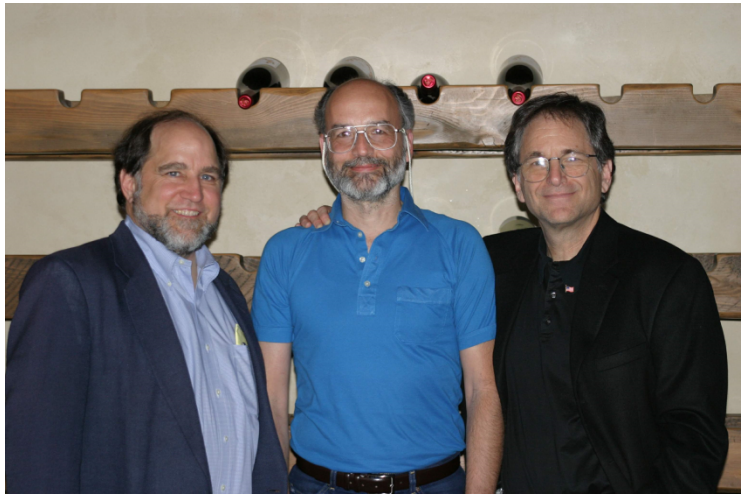
- Examples
  - Factorization
    - Breaking a large number into prime numbers
  - Calculation of discrete logarithms over a finite body
- ☞ Asymmetric encryption methods take their security from the use of mathematical problems that can only be solved with great effort (i.e., not in polynomial time)

Methods for Asymmetric Encryption

- The best known methods of asymmetric encryption are:
  - RSA
    - Basis: Factorization problem
  - ElGamal
    - Basis: Calculation of discrete logarithms over a finite body
  - Elliptic curves
    - Basis: Calculation of discrete logarithms in algebraic groups

RSA





- Invented by Rivest, Shamir, and Adleman
- In 1977 at MIT
- Was patented from 1983 to 2000
- First published public key cryptosystem
- Original idea goes back to Diffie and Hellman
  - Theory of how it could work
  - No mathematical function

RSA Cryptosystem: Some Facts

- Is an asymmetric encryption / decryption mechanism
- Can be defined for different key lengths
  - E.g., 1024 bit, many applications use 2048 bit modulus as default
- Variable block size but smaller or equal to the key
- Cipher blocks are always as long as the key
- Rarely used for encryption of longer messages
- Often used to
  - Encrypt/decrypt symmetric keys when they are distributed
  - Encrypt authentication challenges
  - Construct digital signatures

Cipher blocks always as long as the key.

Use may be in :
1. Symmetric keys distribution,
2. Encrypt authentication challenges,
3. Construct digital signatures

RSA Key Generation

- Public key:
  - Choose two prime numbers p, q
  - Compute n = pq
  - Choose e invertible mod φ (n)
  - Public key: (n,e)
- Private key:
  - Find d with ed = 1 mod φ (n)
  - Select d as the private key
  - Note: $de = 1 + \varphi(n)k$ for some integer $k \neq 0$

Check if e is invertible with the help of Euclid's Algorithm

Find d with the help of extended Euclid's Algorithm

RSA Encryption / Decryption

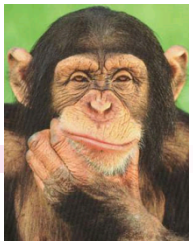
- Encryption
  - For each plaintext m in Z_n: $c = m^e \bmod n$
- Decryption
  - For each cipher-text c in Z_n: $m = c^d \bmod n$
- Why does this work?
  - For any c in Z_n: $c^d \bmod n = (m^e)^d \bmod n = (m^{ed}) \bmod n$

$= m^{\varphi(n)k+1} = m \bmod n$

Euler's Theorem for n=pq

$a^k = \underbrace{a.....a}_{k} \bmod n$

Trivial RSA Example



Setup

Let $p = 3$, $q = 5$, then $n = pq = 15$

$\varphi(n) = (p - 1)(q - 1) = 2 \cdot 4 = 8$

Choose $e = 3$ (rel. prime to $\varphi(n)$)

Then $d = 3$ ($ed = 1 \bmod 8$)

Encryption of m = 7:

$m^e \bmod n = 7^3 \bmod 15 = 343 \bmod 15 = 13$

Decryption of c = 13:

$c^d \bmod n = 13^3 \bmod 15 = 2197 \bmod 15 = 7$

Encryption Speed of RSA

- Asymmetric encryptions are generally slower than symmetric.
  - Key sizes: 512 - 2048 bits
- Hardware implementations
  - Fastest hardware implementation: 64 Kbit/s
  - 512-bit blocks
  - ☞100 times slower than AES
- Software implementations
  - ☞1000 times slower than AES

Cryptanalytic Attacks on RSA

- Cryptanalysis has not yet broken RSA
- Approaches
  - Brute Force attacks
    - Countermeasure: longer keys
  - Factorization
    - Largest success: 768-bit number in two years (equivalent to 2000 years of computing on a single-core 2.2GHz AMD Opteron)
    - Not proved whether factorization is necessary at all!
  - Guessing the value $(p - 1) \times (q - 1)$
    - just as complex as factorization
  - Side-Channel attacks
    - E.g., processing time, power consumption
  - Quantum computing?
    - Not a big problem for symmetric crypto and hash functions

Euler's Theorem

Euler's Theorem: Important Slides

For any a in Z^*_n: $a^{\varphi(n)} = 1 \bmod n$

Reformulation: Important Slides

For any a in Z^*_n, and any integer l: $a^{\varphi(n)l+1} = a \bmod n$

Proof:

Note: If a, b in Z^*_n, then ab in Z^*_n

Also note: Multiplying all elements of Z^*_n with some a in Z^*_n just reorders them.

Assume x is the product of all different $x_1,\ldots,x_{\varphi(n)}$ in Z^*_n. Then, for any a in Z^*_n: $ax_1ax_2\cdots ax_{\varphi(n)} = a^{\varphi(n)}x = x$ (otherwise $ax_i = ax_j$ for some $i \neq j$)

Multiplying this with x^{-1} yields $a^{\varphi(n)} = 1 \bmod n$.

Generalization for n=pq

Generalization in case n=pq: Important Slides

In case n = pq, $a^{\varphi(n)+1} = a \bmod n$ holds for all a in Z_n and not only for the ones relatively prime to n.

Proof:

For a = 0 the equation clearly holds. For a invertible mod n see proof on last slide. If a is not invertible, then a is not relatively prime to n=pq. Therefore a is divisible by either p or q (if by both, then a = 0). So assume a is divisible by q but not by p, then $a^{p-1} = 1 \bmod p$, and $a^{q-1} = 0 \bmod q$. As a consequence $a^{\varphi(n)+1} = a^{(p-1)(q-1)+1} = a \bmod p$ and $a^{\varphi(n)+1} = a^{(p-1)(q-1)+1} = a \bmod q$. So there are integers k,s with $a^{\varphi(n)+1} = a+kp$ and $a^{\varphi(n)+1} = a+sq$. So sq = kp, such that q divides k. So there is an integer l with k = lq and we have $a^{\varphi(n)+1} = a + kp = a + lqp = a \bmod n$

How Secure is RSA?

Theorem: Let p, q prime numbers and n = pq. Then n can efficiently be factorized iff φ(n) can be efficiently computed.

Proof:

=>“: If n can be efficiently factorized then p and q can efficiently be computed from n and therefore φ(n) = (p-1)(q-1) is efficiently computable

⇐“: If φ(n) is known, then one can compute p and q from the two equations n = pq und φ (n) = (p − 1)(q − 1)

So factorizing n is equivalent to computing φ(n)

Important Slides

How Secure is RSA? (2)

Theorem: Let p, q prime, n = pq and (e,n) a public RSA key and d the corresponding private RSA key. Then d can be efficiently computed from (n,e) iff n can be factorized efficiently.

Proof Outline: "<=" clear! "=>" Let s: = max{t in N, 2^t | ed-1} and k: = ed-1 / 2^s. Choose an integer a in Z_n randomly and uniformly distributed. Compute gcd(a,n). If gcd(a,n) ≠ 1 then p or q is found. If gcd(a,n) = 1, then compute gcd($a^{2^t k}$, n) for t = s-1,.., 0. If gcd($a^{2^t k}$, n) ≠ 1 for t = s-1,..,0, then choose another a in Z_n. Each time an a in Z_n is chosen, the probability to find p or q is >1/2. The probability of success after r iterations is therefore > 1- $½^r$

So, computing d is equivalent to factorizing

Important Slides

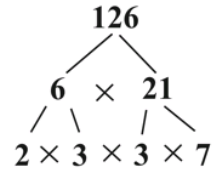
How Secure is RSA? (cont'd)

Important Slides

- Putting it together we have
  - Being able to compute a private RSA key d from (e, n) is equivalent to being able to factorize n
  - Which in turn is equivalent to being able to efficiently compute φ(n)

- However: it is unclear if there is a way to decrypt RSA-encrypted messages without knowledge of the private key.

How Hard is Factorization?



- There is currently no efficient algorithm for factorization
- There are algorithms with sub-exponential run time. E.g.
  - Pollard's Rho Method
  - Quadratic Sieve
  - Number Sieve
- Currently, RSA modules of 1024 bit still often in use but many applications moved to 2048 bit as default already
- The prime numbers p and q are required to be of 512 bit or 1024 bit length respectively

Modular Exponentiation

- RSA Encryption and Decryption are based on modular exponentiation: $x^k \bmod n$
- "Naïve" modular exponentiation requires k modular multiplications
- Problem: the size of the exponent is of the same order as the size of the modulus n Important Slides
  - E.g. 1024 bit or 2048 bit

➢ "Naïve" modular exponentiation is not efficient

(too ready to believe someone or something, or to trust that someone's intentions are good, esp. because of a lack of experience: It was naive of her to think that she would ever get her money back.)

RSA Efficiency

- RSA uses large numbers => need for some tricks to speed up computations
- Example: n=678, d=54, x=123
- For encryption compute $123^{54} \bmod 678$
  - Naïve approach:
    - compute $r=123^{54}$ (about 100 digits)
    - compute r mod 678
  - Very time consuming
- => A reduction of the number of computation steps is necessary

Efficiency of RSA (2)

- Reduction of the number of steps:
- 1. Reduce result after each multiplication step
  - $123^2 = 123*123 = 15129 = 213 \bmod 678$
  - $123^3 = 123^2*123 = 123*213 = 26199 = 435 \bmod 678$
  - …
- 2. Using computation rule for n=m+m: $x^n = x^{m+m} = x^m * x^m$
  - $123^4 = 123^2*123^2 = 213*213 = 54369 = 621 \bmod 678$
  - $123^8 = 123^4*123^4 = 621*621 = 385641 = 537 \bmod 678$
  - …

Efficiency of RSA (3)

- Computing exponents that are not a power of 2
  - Need for intermediate steps: compute x^{2y+1} from x^{2y}
  - When to perform such an additional step?
    - Consider 54 == 110110
    - Consists of sequences 1, 11, 110, 1101, 11011, 110110
  - Each time a sequence element is twice the preceding element or one more
    - If 'twice the preceding element' double the exponent (and divide by mod 678)
    - If 'one more': double the exponent and multiply by x (and divide)
    - $123^{54}=(((((123)^2 * 123)^2)^2 * 123)^2 * 123)^2 = 87 \bmod 678$
  - 8 multiplications and 8 divisions necessary
  - Increases linearly with length of exponent in bit
  - Efficient method

Efficient Modular Exponentiation Formalization

- Idea: Use the binary representation of the exponent

$$k = \sum_{i=0}^{r-1} k_i 2^i = k_0 + 2(k_1 + 2(k_2 + \cdots) \cdots),$$

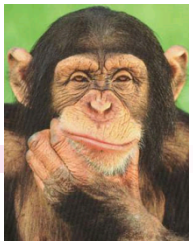
where k_0, k_1, k_2 in $\{0,1\}$

- Then we get:

$$x^k = \prod_{i=0}^{r-1} x^{k_i 2^i}$$

- So all we have to do is multiply by x or square

Another Example

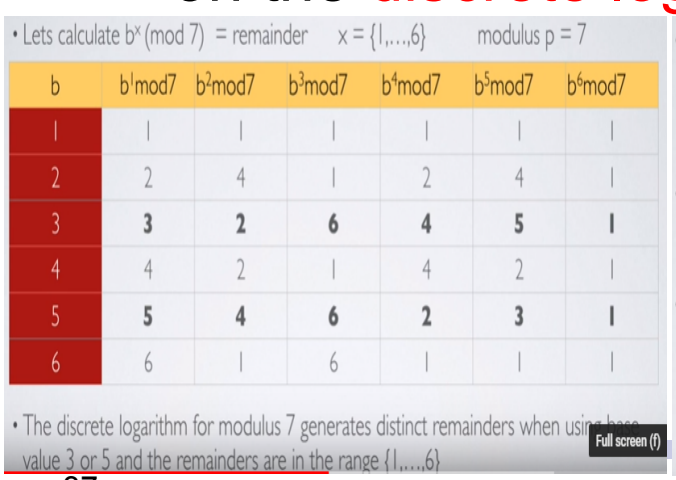


$$37 = 1 * 2^0 + 0 * 2^1 + 1 * 2^2 + 0 * 2^3 + 0 * 2^4 + 1 * 2^5$$

$$X^{37} = x * x^{2^2} * x^{2^5} = (((((x^2)^2)^2)x)^2)^2 x$$

Diffie-Hellman Key Exchange(1)

- Oldest public key mechanism
  - Invented in 1976
- Is a key establishment protocol by which two parties can
  - Establish a secret key K
  - Based on publicly exchanged values
- The security of the key exchange is based on the discrete logarithm problem

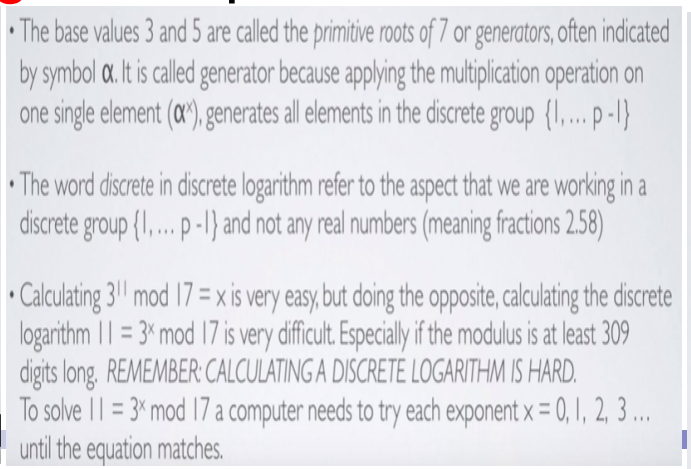


• Lets calculate $b^x \pmod 7$ = remainder x = {1,...,6} modulus p = 7

b	$b^1 \bmod 7$	$b^2 \bmod 7$	$b^3 \bmod 7$	$b^4 \bmod 7$	$b^5 \bmod 7$	$b^6 \bmod 7$
1	1	1	1	1	1	1
2	2	4	1	2	4	1
3	**3**	**2**	**6**	**4**	**5**	**1**
4	4	2	1	4	2	1
5	**5**	**4**	**6**	**2**	**3**	**1**
6	6	1	6	1	1	1

• The discrete logarithm for modulus 7 generates distinct remainders when using base value 3 or 5 and the remainders are in the range {1,...,6}

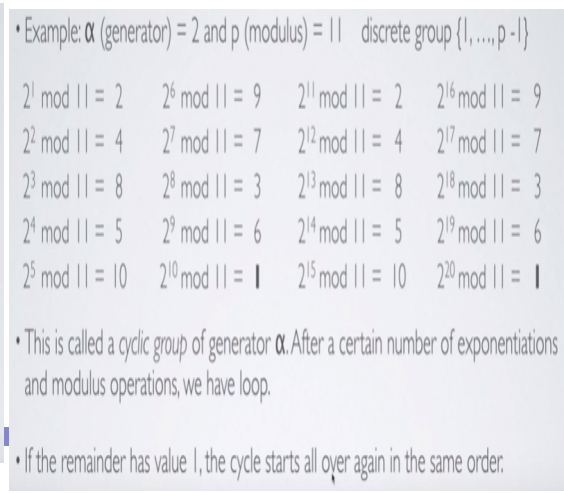
Full screen (f)



• The base values 3 and 5 are called the *primitive roots of 7* or *generators*, often indicated by symbol α. It is called generator because applying the multiplication operation on one single element (α^x), generates all elements in the discrete group {1, ... p -1}

• The word *discrete* in discrete logarithm refer to the aspect that we are working in a discrete group {1, ... p -1} and not any real numbers (meaning fractions 2.58)

• Calculating $3^{11} \bmod 17 = x$ is very easy, but doing the opposite, calculating the discrete logarithm $11 = 3^x \bmod 17$ is very difficult. Especially if the modulus is at least 309 digits long. *REMEMBER: CALCULATING A DISCRETE LOGARITHM IS HARD.*
To solve $11 = 3^x \bmod 17$ a computer needs to try each exponent x = 0, 1, 2, 3 ... until the equation matches.



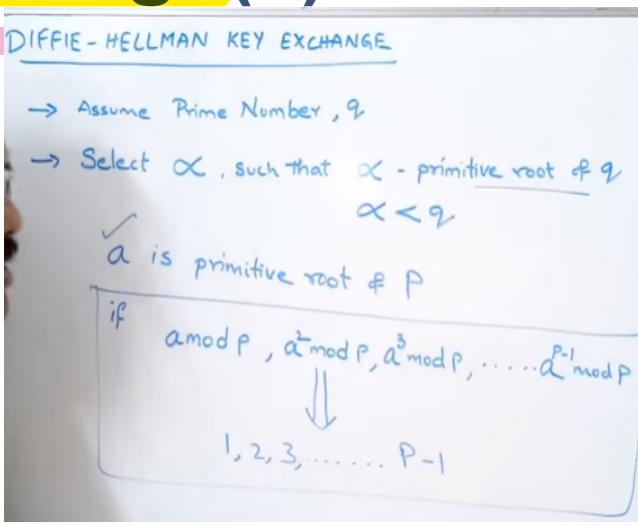
• Example: α (generator) = 2 and p (modulus) = 11 discrete group {1, ..., p -1}

$2^1 \bmod 11 = 2$	$2^6 \bmod 11 = 9$	$2^{11} \bmod 11 = 2$	$2^{16} \bmod 11 = 9$
$2^2 \bmod 11 = 4$	$2^7 \bmod 11 = 7$	$2^{12} \bmod 11 = 4$	$2^{17} \bmod 11 = 7$
$2^3 \bmod 11 = 8$	$2^8 \bmod 11 = 3$	$2^{13} \bmod 11 = 8$	$2^{18} \bmod 11 = 3$
$2^4 \bmod 11 = 5$	$2^9 \bmod 11 = 6$	$2^{14} \bmod 11 = 5$	$2^{19} \bmod 11 = 6$
$2^5 \bmod 11 = 10$	$2^{10} \bmod 11 = 1$	$2^{15} \bmod 11 = 10$	$2^{20} \bmod 11 = 1$

• This is called a *cyclic group* of generator α. After a certain number of exponentiations and modulus operations, we have loop.

• If the remainder has value 1, the cycle starts all over again in the same order.

Diffie-Hellman Key Exchange (2)



Public information known to both parties

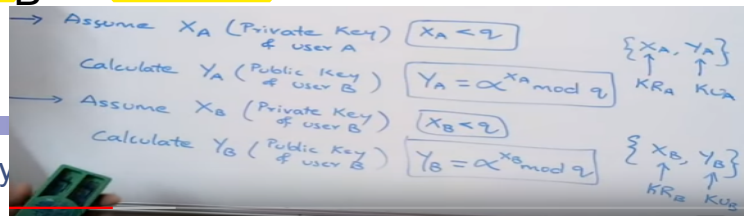
- Prime number p and a generator g of Z^*_p

Private / Public Diffie-Hellman values

- Alice
  - Selects a private DH value a in {1,…,p-2} randomly
  - Computes the corresponding public DH value $h_A = g^a \bmod p$
- Bob
  - Selects a private DH value b in {1,…,p-2} randomly
  - Computes its public DH value $h_B = g^b \bmod p$

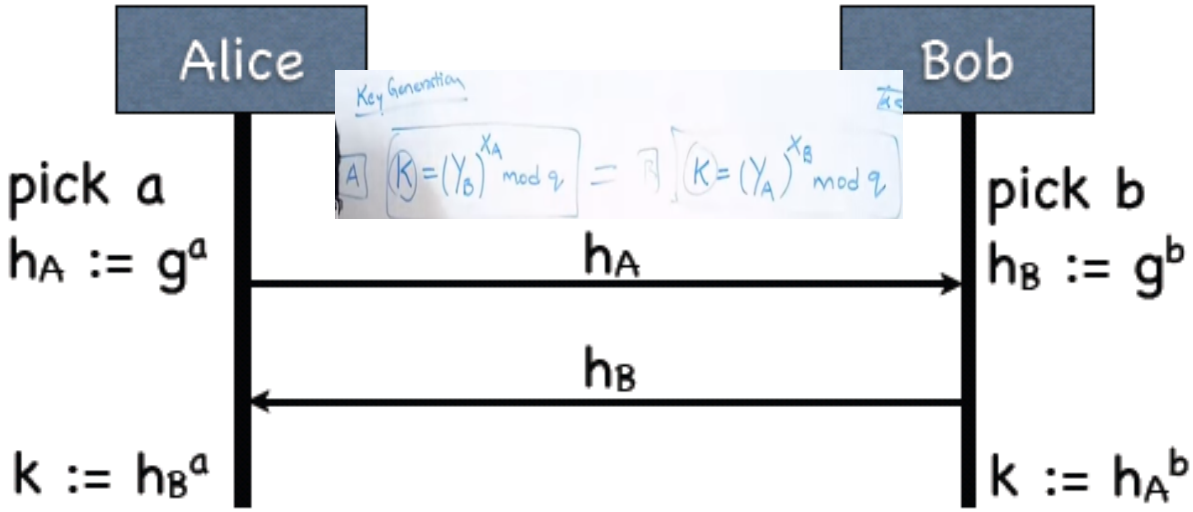
Key exchange

- Alice sends h_A to Bob, Bob sends h_B to Alice

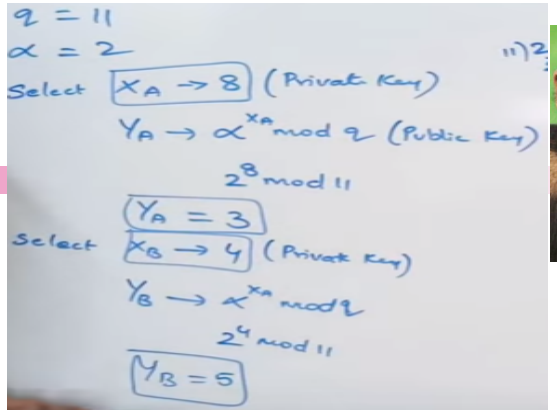


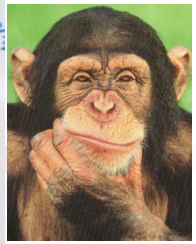
Diffie-Hellman Key Exchange(2)

- With Alice's public value and his private value Bob can compute $h_A^b = g^{ab}$
- With Bob's public value and his private value Alice can compute $h_B^a = g^{ba}$

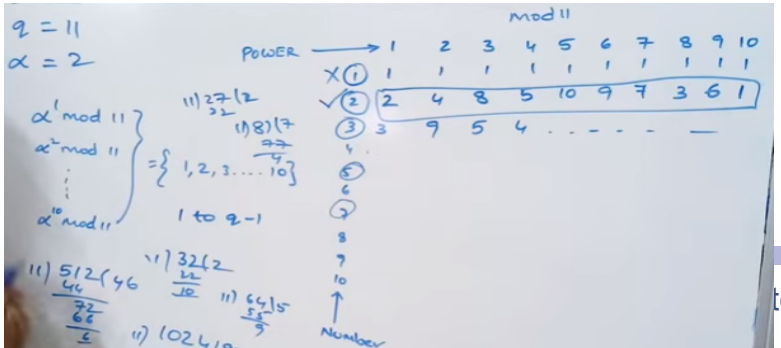


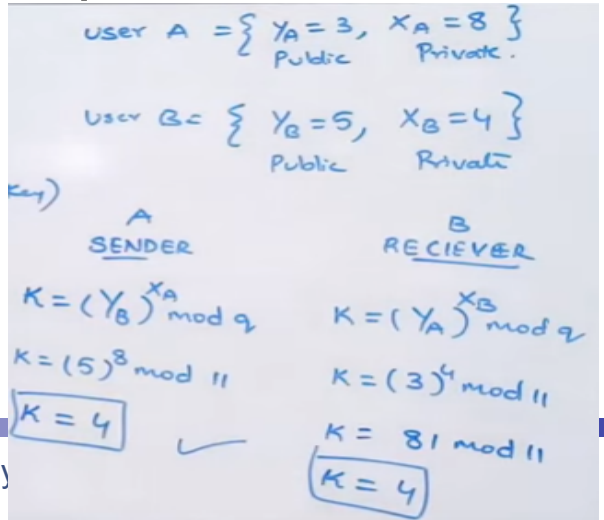
Example



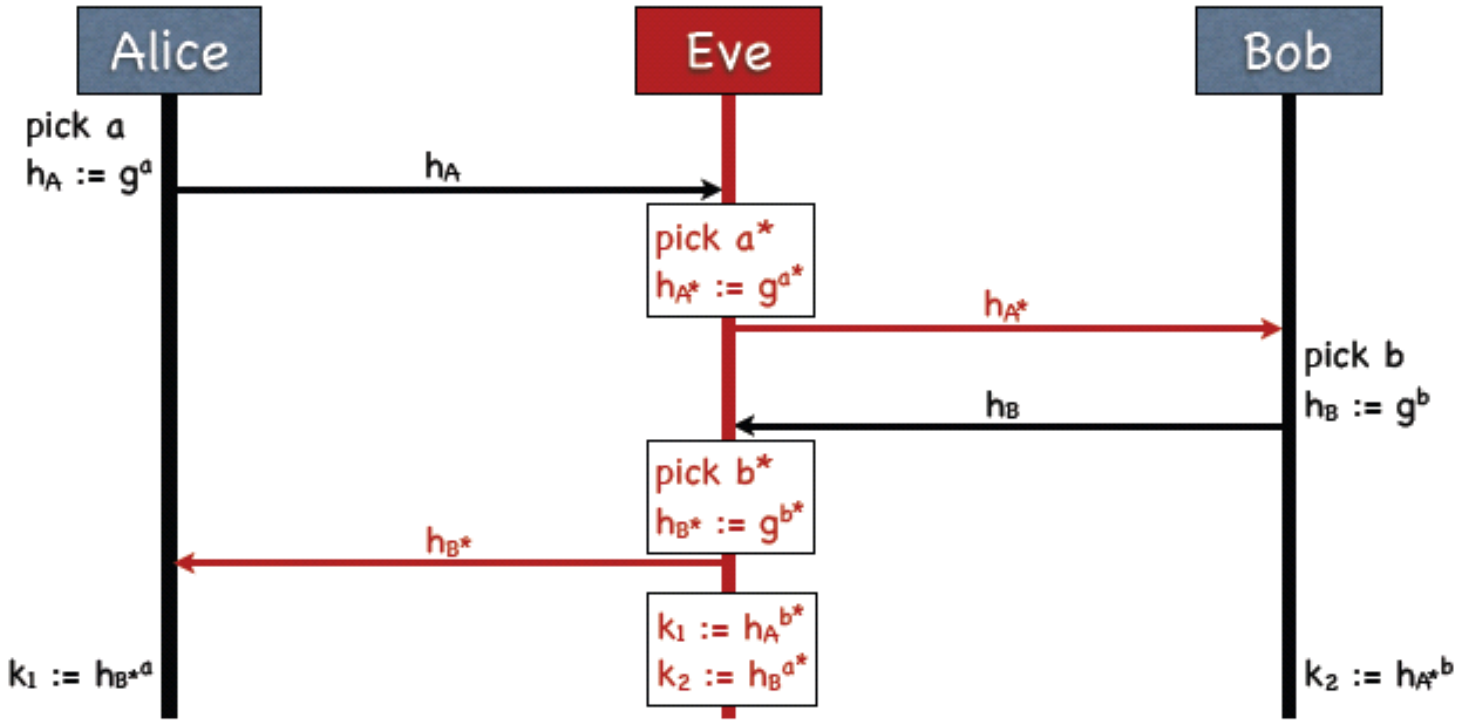


- Let p = 17, g = 3
- Assume Alice chooses a = 7, then A = 3^7 = 11 mod 17
- Assume Bob chooses b = 4, then B = 3^4 = 13 mod 17
- Alice receives B = 13 from bob and computes K = 13^7 mod 17 = 4
- Bob receives A = 11 from Alice and computes K = 11^4 mod 17 = 4
- Alice and Bob share the key K = 4





Man-in-the-Middle Attack



- Eave has a key with each Alice and Bob
- Alice and Bob do not share a key
- Works because Bob and Alice have no proof that the public values are authentic -> see lecture on certificates

ElGamal Algorithm

- *Public Key* algorithm
  - Deployment for asymmetric decryption and digital signatures
  - *Developer*: T. ElGamal
  - Slower than RSA

- Principle
  - Security is due to the difficulty of calculating discrete logarithms over a finite body
  - ↳ Specific variant of the Diffie-Hellmann key exchange method combined with symmetric encryption

Key Generation

- Choose a prime p and two random numbers g and x with g < p, x < p
- Calculate $y = g^x \bmod p$

- Public key: {y, g, p}
- Private key: x

Encryption and Decryption

Encryption

- Choose randomly a value r that is smaller than p

- Calculate
  - $a = g^r \bmod p$
  - $k = y^r \bmod p$ (→ one-time key)
  - Encrypt plain text M with k : $C = e_k(M) = M * k = M * y^r \bmod p$

- ☞ ***a* and *C* are transmitted**
  - → r remains secret

Decryption

- Calculate symmetric key $k = a^x \bmod p$
- $M = d_k(C) = C * k^{-1}$

Encrypting longer messages ElGamal

- M<p
- If M is longer, it needs to be broken up into blocks
- If k is used for more than one block, knowledge of one block allows to compute other blocks
  - $C_1=e_k(M_1) = M_1*k = M_1*y^r \bmod p$
  - $C_2=e_k(M_2) = M_2*k = M_2*y^r \bmod p$
  - $C_1 / C_2=e_k(M_1) = M_1*y^r \bmod p \,/\, M_2*y^r \bmod p$
  - If M_1 is known then M_2 can be easily computed: $M_2=(C_1)^{-1}\, C_2\, M_1\, y^r \bmod p$
- => *unique k should be used for each block!*

ECC – Elliptic Curve Cryptography

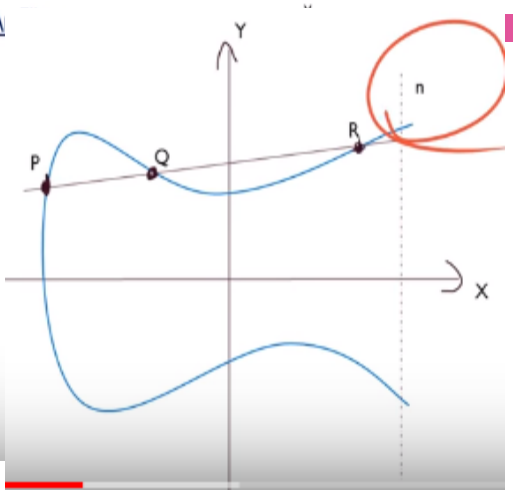
- What are Elliptic Curves?
  - Curve with standard form $y^2 = x^3 + ax + b$ a, b ϵ $\mathbb{R}$

- Asymmetric / Public key cryptosystem.
- ECC provides equal security with smaller key size.
- ECC makes use of Elliptic curves.
- Elliptic curves are defined by mathematical functions - Cubic functions.
- eg:- $y^2 = x^3 + ax + b$

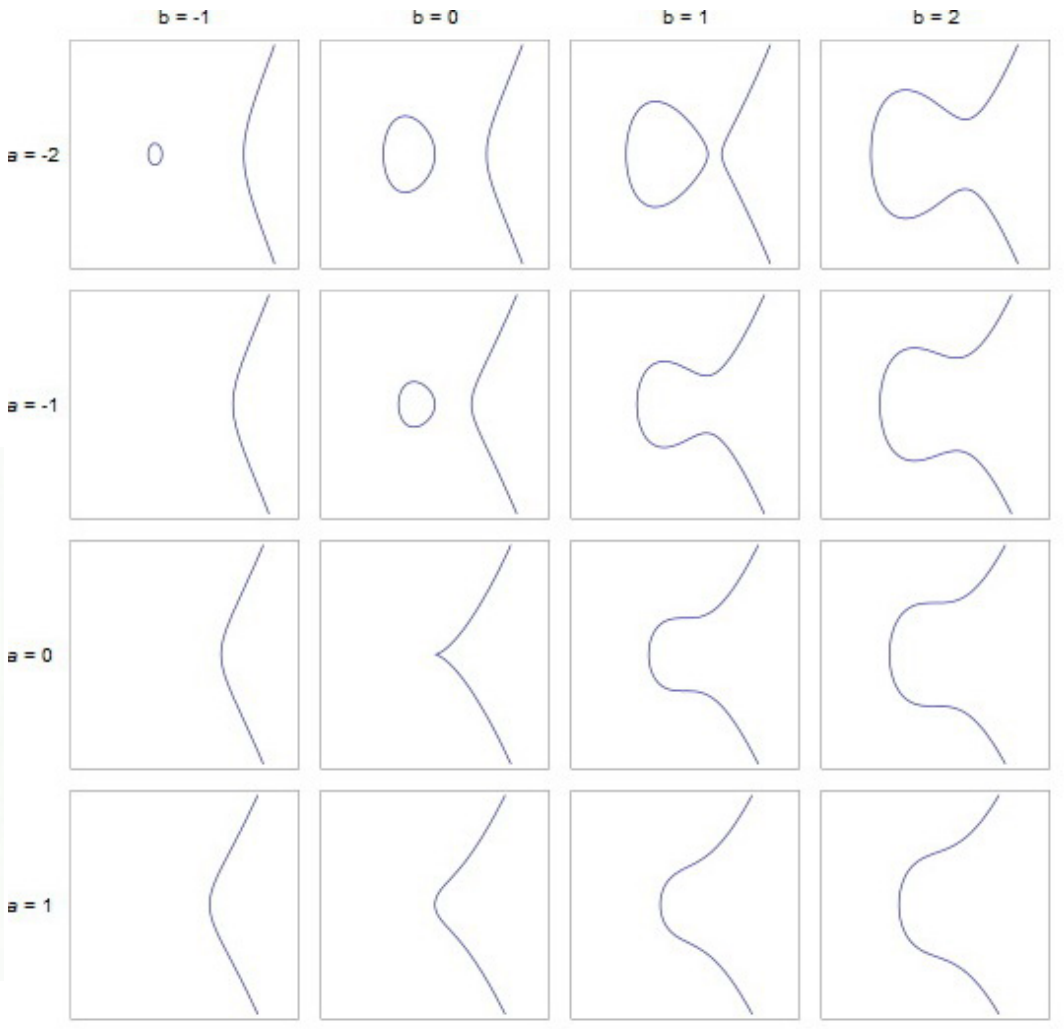
- Characteristics of Elliptic Curve
  - Forms an abelian group
    - Commutative (result of applying the group operation to two group elements does not depend on the order in which they are written)
    - Exists an inverse element
  - Symmetric about the x-axis
  - *Point at Infinity (aka zero point)* acting as the identity element
    - Leaves other elements unchanged when combined with them

(based on slides of Chandrashekar and Maheshwary)

Examples of elliptic curves



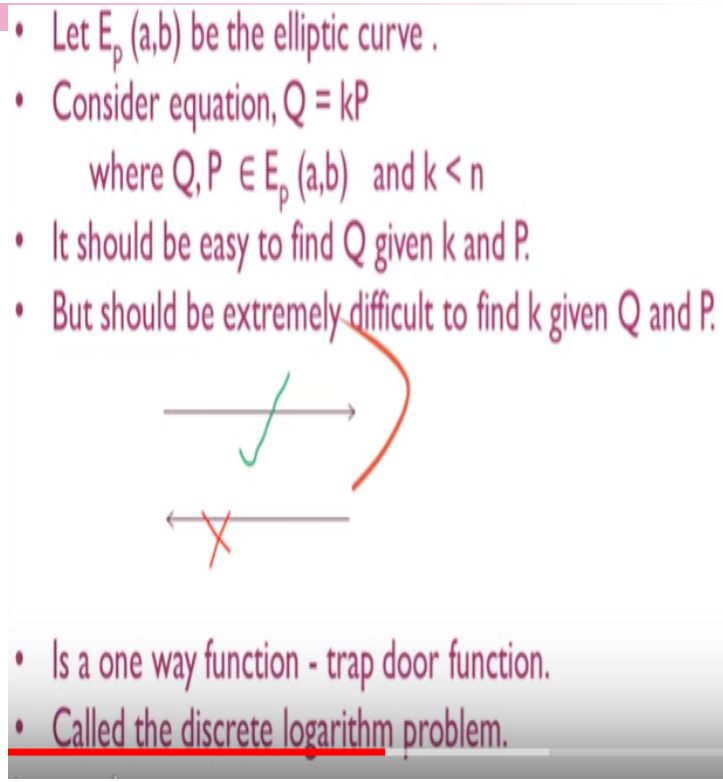
- **Closure:** For all a, b in A, the result of the operation $a \bullet b$ is also in A.
- **Associativity:** For all a, b and c in A, the equation $(a \bullet b) \bullet c = a \bullet (b \bullet c)$ holds.
- **Identity element:** There exists an element e in A, such that for all elements a in A, the equation $e \bullet a = a \bullet e = a$ holds.
- **Inverse element:** For each a in A, there exists an element b in A such that $a \bullet b = b \bullet a = e$, where e is the identity element.
- **Commutativity:** For all a, b in A, $a \bullet b = b \bullet a$.
- A group in which the group operation is not commutative is called a "non-abelian group" or "non-commutative group".



Why Elliptic Curve Cryptography?

Pros:

- Shorter Key Length
- Lesser Computational Complexity
- Low Power Requirement
- More Secure



Cons:

- Relatively new (possibly not all attack vectors have been discovered already)

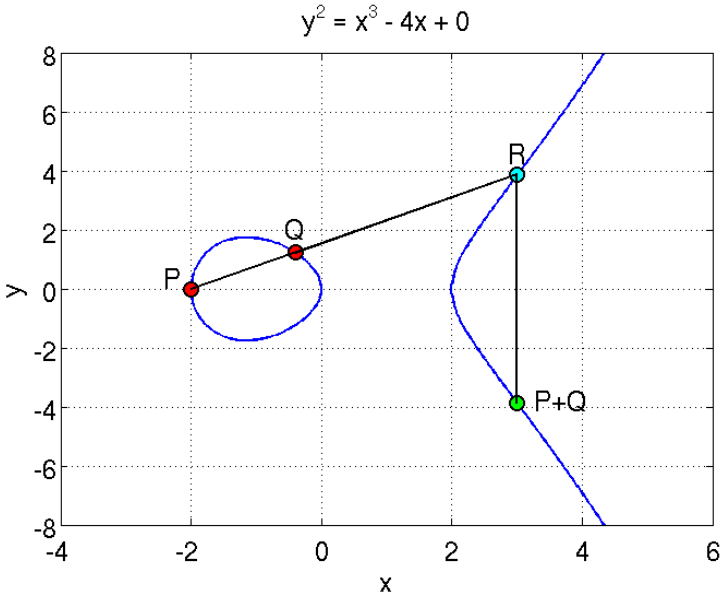
Comparable Key Sizes for Equivalent Security

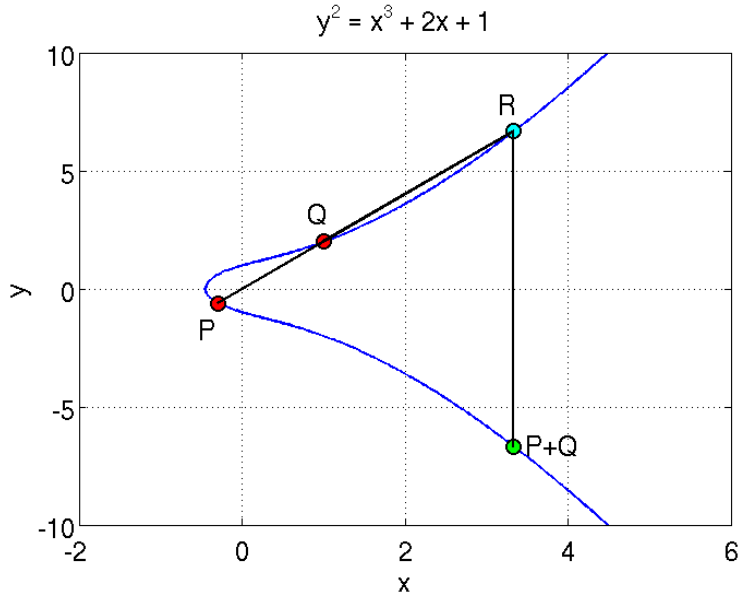
Symmetric Encryption (Key Size in bits)	RSA and Diffie-Hellman (modulus size in bits)	ECC Key Size in bits
56	512	112
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	512

What is Elliptic Curve Cryptography?

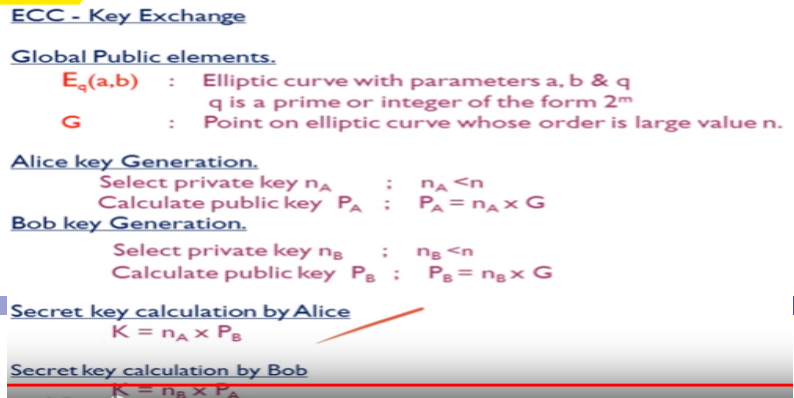
- Implementing Group Operations
  - Main operations - point addition and point multiplication
  - Adding two points that lie on an Elliptic Curve – results in a third point on the curve
  - Point multiplication is a repeated addition
  - If P is a known point on the curve (aka Base point; part of domain parameters) and it is multiplied by a scalar k, Q=kP is the operation of adding P + P + P + P… +P (k times)
  - Q is the resulting public key and k is the private key in the public-private key pair
  - Discrete logarithm problem: given P and Q find k
    - If k is very large, it becomes computationally infeasible

What is Elliptic Curve Cryptography?

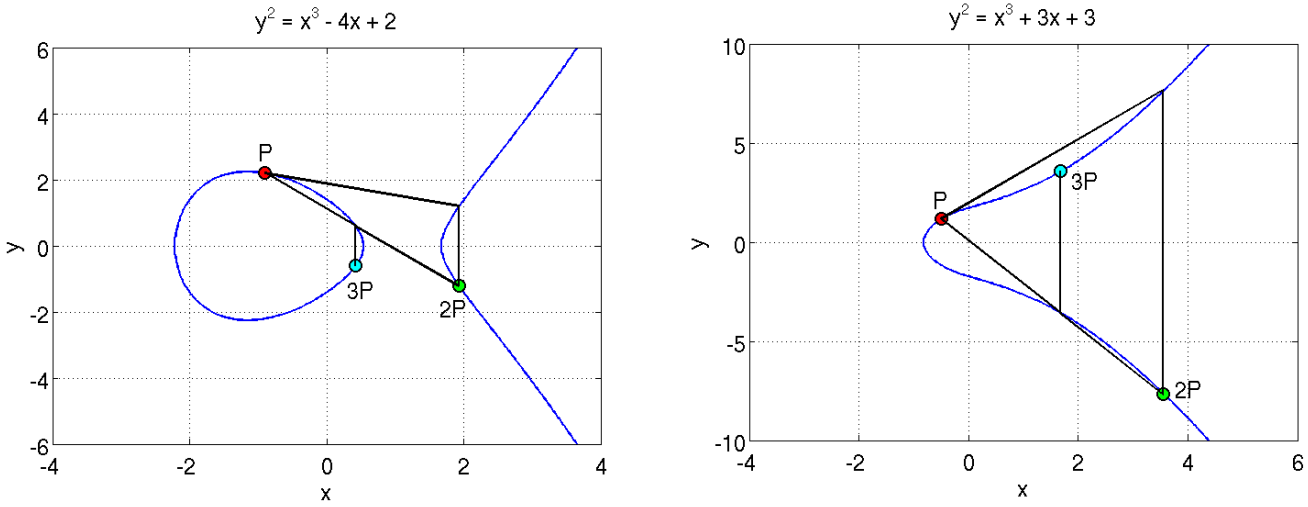




- P and Q are added to obtain P+Q which is a reflection of R along the x axis



What is Elliptic Curve Cryptography?



- A tangent at P is extended to cut the curve at a point; its reflection is 2P
- Adding P and 2P gives 3P
- Similarly, such operations can be performed as many times as desired to obtain Q = kP

Finite Fields

- The Elliptic curve operations shown were on real numbers
  - Issue: operations are slow and inaccurate due to round-off errors
- To make operations more efficient and accurate, the curve is defined over finite fields
- The field is chosen with finitely large number of points suited for cryptographic operations
- Galois Field
  - $GF(p^n)$ = a set of integers {0, 1, 2, …, p^n -1) where p is a prime, n is a positive integer

What is Elliptic Curve Cryptography?

- Discrete Log Problem
  - The security of ECC is due the intractability or difficulty of solving the inverse operation of finding k given Q and P
  - This is termed as the discrete log problem
  - Methods to solve include brute force and Pollard's Rho attack both of which are computationally expensive or infeasible
  - The version applicable in ECC is called the Elliptic Curve Discrete Log Problem
  - Exponential running time

- Let the message be M.
- First encode the message M into a point on the elliptic curve.
- Let this point be P_m.
- Now this point is encrypted.
- For encrypting choose a random positive integer k.
- Then $C_m = \{kG, P_m + kP_B\}$ where G is the base point.

- For decryption, multiply first point in the pair with receiver's secret key.
  i.e, $kG \times n_B$
- Then subtract it from second point in the pair.
  i.e, $P_m + kP_B - (kG \times n_B) = P_m + kP_B - (kP_B) = P_m$

ECC in Windows DRM v2.0

A Practical Example :

Finite field chosen

p = 7859631023794288223766947894468973962074985689 51

Gx = 771507216262649826170648268565579889907769254176

Gy = 390157510246556628525279459266514995562533196655

$y^2 = x^3 + 3176890812513255034763174764138276932727469559 27x +$
$790528966078787587181205720257185354321100651934$

Gx and Gy constitute the agreed upon base point (P) and the numbers in the above equation are values for the parameters a and b

Elliptic Curve Diffie-Hellman

- Elliptic curve variant of the key exchange Diffie-Hellman protocol
- Decide on domain parameters and come up with a Public/Private key pair
- To obtain the private key, the attacker needs to solve the discrete log problem

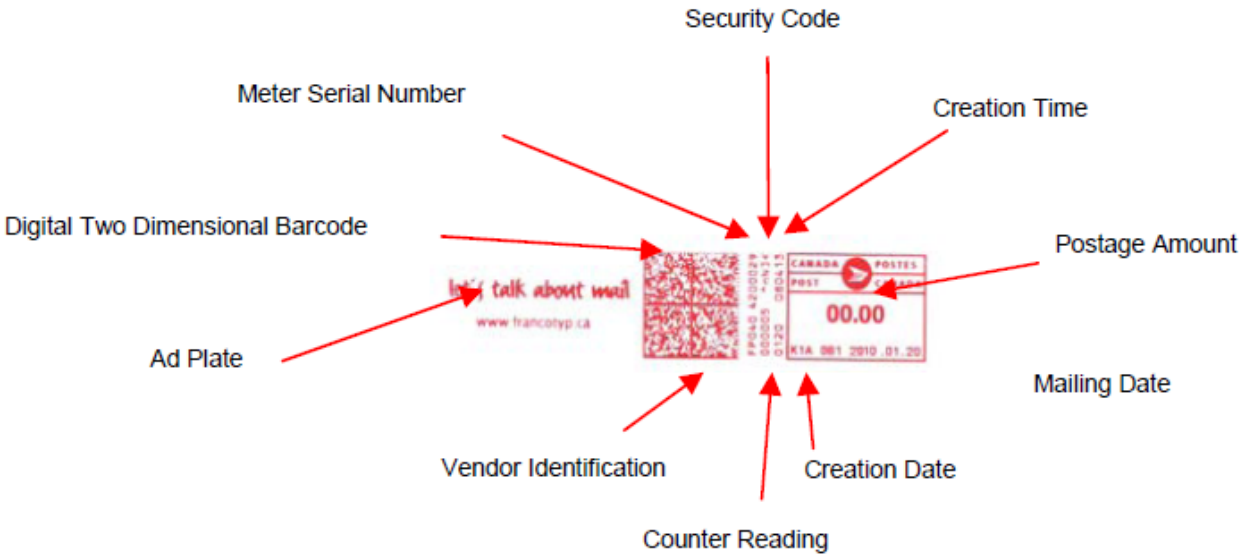
Elliptic Curve Diffie-Hellman

- How the key exchange takes place:

  1. Alice and Bob publicly agree on an elliptic curve E over a large finite field F and a point P on that curve.
  2. Alice and Bob each privately choose large random integers, denoted a and b
  3. Using elliptic curve point-addition, Alice computes aP on E and sends it to Bob. Bob computes bP on E and sends it to Alice.
  4. Both Alice and Bob can now compute the point abP Alice by multiplying the received value of bP by her secret number a and Bob vice-versa.
  5. Alice and Bob agree that the x coordinate of this point will be their shared secret value.

Elliptic Curve Digital Signature Algorithm (ECDSA)

- Elliptic curve variant of Digital Signature Algorithm



Canadian postage stamp that uses ECDSA

ECDSA – Signature Generation

For signing a message m by sender A, using A's private key d_A and public key $Q_A = d_A * G$

1. Calculate $e = HASH(m)$, where HASH is a cryptographic hash function, such as SHA-3
2. Select a random integer k from $[1, n-1]$
3. Calculate $r = x_1 \pmod n$, where $(x_1, y_1) = k * G$. If $r = 0$, go to step 2
4. Calculate $s = k^{-1}(e + d_A r) \pmod n$. If $s = 0$, go to step 2
5. The signature is the pair (r, s)

ECDSA – Signature Verification

For B to verify A's signature, B must have A's public key Q_A

1. Verify that r and s are integers in [1,n − 1]. If not, the signature is invalid
2. Calculate e = HASH (m), where HASH is the same function used in the signature generation
3. Calculate $w = s^{-1} \pmod n$
4. Calculate $u_1 = ew \pmod n$ and $u_2 = rw \pmod n$
5. Calculate $(x_1, y_1) = u_1G + u_2Q_A$
6. The signature is valid if $x_1 = r \pmod n$, invalid otherwise

ECDSA – Why it works

This is how we know that the verification works the way we want it to:

We have, $s = k^{-1}(e + dr) \bmod n$ which we can rearrange to obtain, $k = s^{-1}(e + dr)$ which is

$$s^{-1}e + s^{-1}rd$$

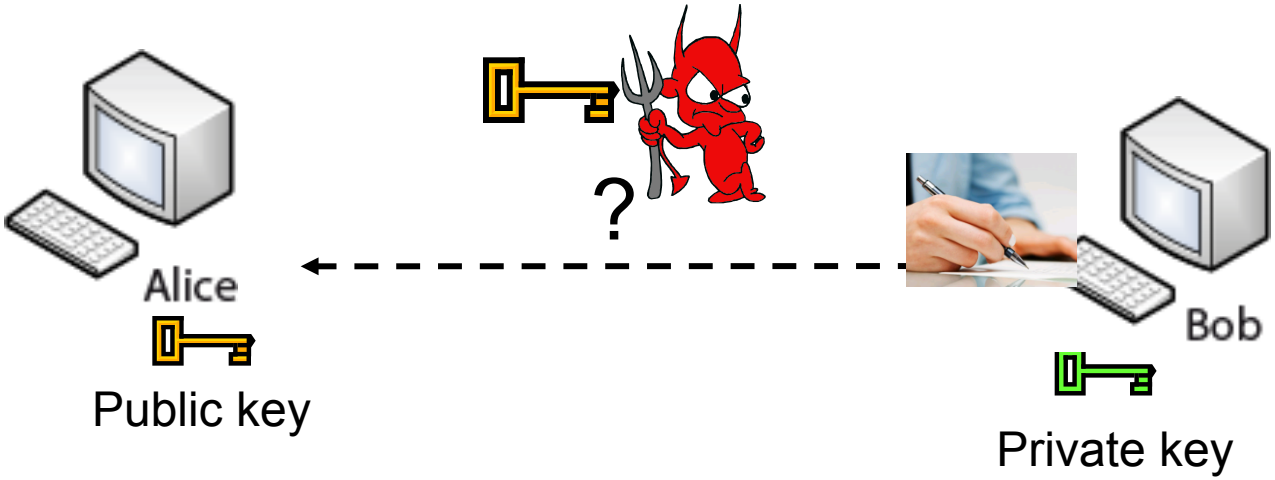
This is nothing but $we + wrd = (u_1 + u_2 d_A) \pmod n$

We have $u_1 G + u_2 Q_A = (u_1 + u_2 d_A)G = kG$ which translates to $x_1 = r$.

Symmetric vs. Asymmetric Encryption

- Advantage of asymmetric encryption schemes
  - Do not require a secret channel for key exchange
  - Less keys required to be kept secret
- Advantages of symmetric encryption schemes
  - Can be implemented more efficiently
- Goal: Leverage on the advantages of both
  - Use hybrid schemes in which the secret key for symmetric encryption is exchanged using asymmetric primitives

Basic Idea of Digital Signatures



- Given:
  - Everybody knows Bob's public key
  - Only Bob knows the corresponding private key
- Goal: Bob can digitally sign a message such that anyone can verify his signature
  - To compute a signature, one must know the private key
  - To verify a signature it is enough to know the public key

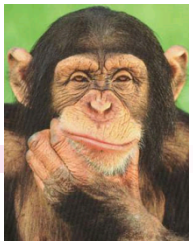
Definition of a Digital Signature Scheme

- A digital signatures scheme consists of a
  - Key generation algorithm
  - A signature generation algorithm
  - A signature verification algorithm
- Key generation algorithm specifies
  - Generation of the public key for signature verification
  - Generation of the private key for signature generation
- Signature generation algorithm
  - Takes a message M and the private key as input and outputs the signature
- Signature verification algorithm
  - Takes the message, the public key, and the signature as input and returns success or failure of signature verification

Example: RSA Signatures

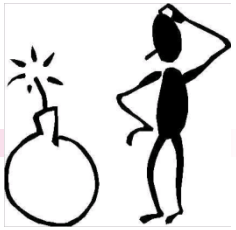
- Key generation (as in RSA Encryption)
  - Choose two large primes p,q randomly
  - Choose an exponent e in $Z^*_{\varphi(n)}$
  - Compute n = pq and d in $Z^*_{\varphi(n)}$ with ed = 1 mod φ (n)
  - The public key is then (n,e), the private key is d
- Signing a message m: uses private key d
  - Signature s on m is $s = m^d \bmod n$
- Signature verification: uses public key e
  - On receipt of (m,s) the verifier computes $m' = s^e \bmod n$ and verifies that m = m'

Example for RSA Signature



- Assume Alice chooses p = 11, q = 23, e = 147
- Then n = 253, d = 3
- The public key is (147, 253), the private key is 3
- If Alice wants to sign the message m = 111, she computes
  - $s = 111^3 \bmod 253 = 166$
- On receipt of (m,s) Bob verifies the signature by computing
  - $m' = 166^{147} \bmod 253 = 111$
- As m' = m, Bob knows that Alice has generated the signature s

Problems



- Bob needs to be in possession of the authentic public key of Alice
- "Existential Forgery":
  - An attacker can choose s in Z_n randomly and claim that s is a signature generated by Alice on the message $m:=s^e \bmod n$
  - I.e. it is easy to generate pairs of (m,s) such that s is a valid signature on m (Note: it is not easy to generate a valid s for a given meaningful m)
- RSA is multiplicative homomorphic:
  - If m_1, m_2 are two messages and s_1, s_2 the corresponding signatures of m_1 and m_2, then
  - $s:=s_1 s_2 = m_1^d m_2^d = (m_1 m_2)^d$
  - I.e. s is a valid signature on $m_1 m_2$

Solutions

- Authenticity of Alice's public key can be guaranteed with the help of public key certificates (more below)
- RSA scheme can be protected against existential forgery and the multiplicatively problem with the help of a hash function:
  - Instead of signing m, Alice signs h(m) for some publicly known cryptographic hash function h, i.e. $s = h(m)^d$
  - Existential forgery would now require Eve to chose some s and find a message m such that $s^e = h(m)$. If h is pre-image resistant, this is hard
  - The fact that RSA is multiplicative is not a problem any more either: it is hard to find a message m such that $h(m) = h(m_1)h(m_2)$

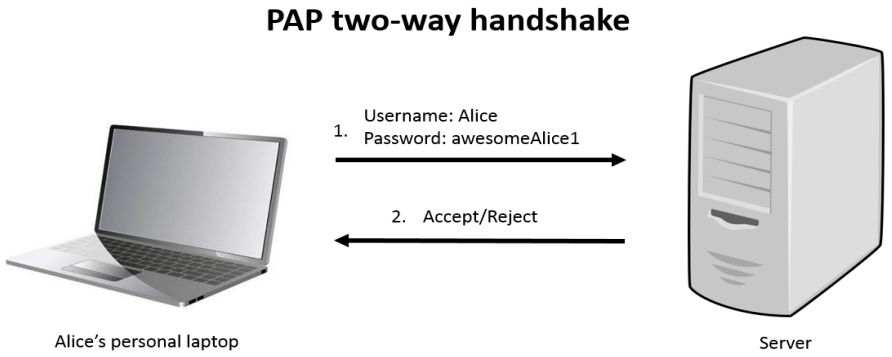
Signing the Hash Instead of the Message

- Instead of signing a message m of arbitrary length a hash function h: {0,1}* ➔ $\{0,1\}^n$ is used and h(m) is signed
- The advantages are
  - Signatures are computed on smaller values
  - Security advantages for some signature schemes
    - See e.g. above for RSA
  - Larger messages do not have to be split in blocks small enough for the signature scheme
    - If messages are split receiver of the signed blocks is not able to recognize if all the blocks are present and in the appropriate order

Use of Digital Signatures

- Message authentication
  - Can also be provided by a MAC
- Message integrity
  - Can also be provided by a MAC
- Non-repudiation
  - Can not be provided by a MAC: a MAC can be generated by the "signer" and the "verifier"
  - A signature can only be computed by the entity that is in possession of the private key. The signer can therefore not repudiate having signed the message
- Authentication Protocols

An authentication protocol is a type of computer communications protocol or cryptographic protocol specifically designed for transfer of authentication data between two entities.



Types of Attacks on Digital Signatures

- Key-Only Attack: The attacker tries to generate valid signatures while only in possession of the public verification key
- Known-Message Attack: The attacker knows some message/signature pairs and tries to generate another valid signature on some message
- Chosen-Message Attack: The attacker can choose messages, can make the signer sign them and tries to generate another valid signature on some other message
- Power of attacker highest in chosen-message attack

Hierarchy of Attack Results

- Total break: attack results in recovery of the signature key
- Universal forgery: attack results in the ability to forge signatures for any message
- Selective forgery: attack results in a signature on a message of the adversary's choice
- Existential forgery: merely results in some valid message/signature pair not already known to the adversary (one's opponent in a contest, conflict, or dispute.)
- The strongest notion of security for a signature scheme is to be secure against existential forgery in a chosen message attack

PKCS#1v.2.2 (2012)

- PKCS = Public Key Cryptography Standard
  - By RSA Laboratories
- Defines two encoding methods for RSA Encryption
  - RSAES-PKCS1-v1_5
  - RSAES-OAEP
- For new applications RSAES-OAEP should be used
  - RSAES-PKCS1-v1_5 is vulnerable to chosen ciphertext attacks
- Both methods describe how to pad bit string messages and how to convert them to large integers
- OEAP = Optimal Asymmetric Encryption Padding
- RSAES = RSA Encryption Scheme

RSAES-OAEP Encryption

RSAES-OAEP-ENCRYPT ((*n*, *e*), *M, L*)

Options:

- Hash function (hLen: length of hash in octets)
- MGF: mask generation function
  - May be based on the hash function
  - Main difference: can generate output of arbitrary length

Input:

- (n, e) recipient's RSA public key (k: length of n in octets)
- M message to be encrypted, an octet string of length mLen, where mLen <= k – 2hLen – 2
- L optional label to be associated with the message; the default value for L, if L is not provided, is the empty string

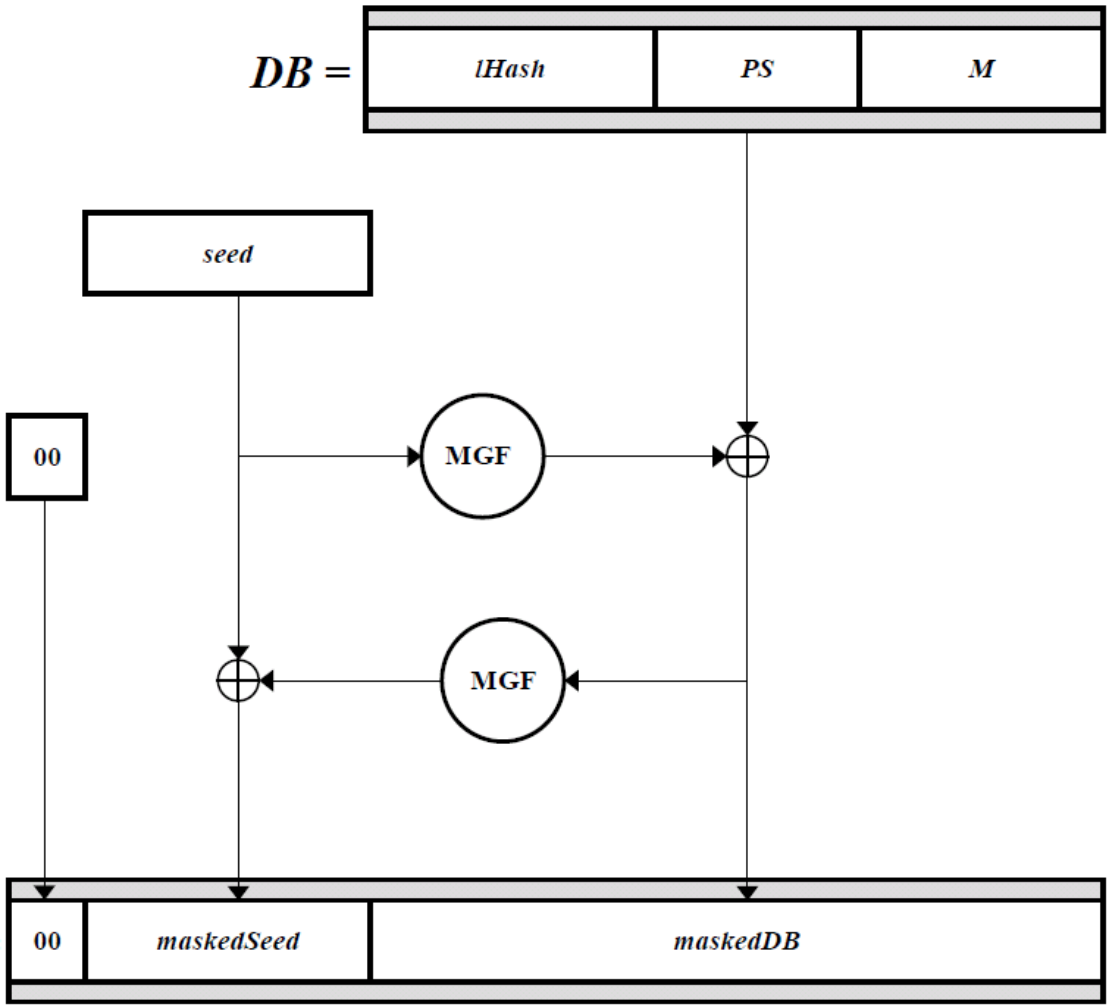
Output:

- C ciphertext, an octet string of length k

RSAES-OAEP Overview

- Octet string message M is padded to an encoded message EM of k octets
- EM is converted to an integer
- The integer representation of EM is encrypted with (n,e) to an integer c
- The integer c is converted to the k-octet string ciphertext C

Encoding of M to EM



- lHash = Hash(L)
- PS: $k - mLen - 2hLen - 2$ zeros (padding string)
- M message
- Seed = random octet string of length hLen
- The leading octet of zeros in EM is used to detect decryption errors
- How does decoding work?

Intuitively: Why PKCS#1?

- The encoding methods in PKCS1 help to detect if a message M is a valid plaintext
  - "Recognizable" plaintext
- In particular the encodings protect against someone exchanging a cipher text $c = m^e$ with a cipher text cs^e
  - Without the encoding: cs^e is the cipher text of ms
  - With the encoding, ms will not have the correct structure

This property is called multiplicativ homomorphic

PKCS#1v.2.2

- Defines two encoding schemes for RSA signature schemes
  - RSASSA-PSS
    - Recommended for new applications
  - RSASSA-PKCS1-v1_5

- Both encoding schemes specify how to
  - Hash an octet string message
  - Encode the hash to a padded octet string
  - Convert the encoded hash to an integer
  - Generate the signature

DSS

- The Digital Signature Standard (DSS) was adopted by NIST in 1994
- Is standardized in FIPS 186
  - Updated in FIPS 186-2, 186-3, 186-4 (2013)
- Uses the Digital Signature Algorithm (DSA)
- Uses SHA-x (160 bit) as hash function h
- Is based on the discrete logarithm problem
  - Given a prime number p and g, x in Z^*_p, and $X = g^x \bmod p$ then it is hard to compute the "discrete logarithm" x of X from X, g, and p
- Is closely related to a Signature Scheme by Schnorr

Key Generation for DSS

- Signer generates two prime numbers p, q with
  - p of 512 bit
  - q of 160 bit
  - q|(p-1) [ie. this means, (p-1) / q]
- Signer chooses x in Z^*_p such that $x^{(p-1)/q} \bmod p \neq 1$ and sets $g := x^{(p-1)/q} \bmod p$
  - Then the smallest integer i for which $g^i = 1 \bmod p$ is $i = q$
- Signer chooses a in {1,…,q-1} and computes
  - $A = g^a \bmod p$
- The public key of the signer is then (p,q,g,A) and the private key is a

Signature Generation in DSS

Signature Generation on message m:

- Signer randomly chooses k in {1,…,q-1} and computes
  - $r = (g^k \bmod p) \bmod q$
  - $s = (k^{-1}(h(m) + ar)) \bmod q$, where k^{-1} is the inverse of k mod q
- The signature on m then is the pair (r,s)

- Note: all exponents used in the signature generation are 160 bit max -> efficiency
- Note: a different k has to be chosen for each message

Signature Verification in DSS

- Upon receipt of m, s, r the verifier
  - Checks if r in {1,…,q-1} and s in {1,…,q-1}
  - Computes $u_1 = h(m)s^{-1} \bmod q$, $u_2 = rs^{-1} \bmod q$
  - Computes $v = (g^{u1}A^{u2} \bmod p) \bmod q$
  - Accepts the message if v = r and rejects the message otherwise

Why the Verification Works

- If m, s, and r were received correctly by the verifier, then:

$$v = (g^{u1} A^{u2} \bmod p) \bmod q$$

$$= (g^{h(m)s^{-1}} g^{ars^{-1}} \bmod p) \bmod q$$

$$= (g^{(h(m)+ar)s^{-1}} \bmod p) \bmod q$$

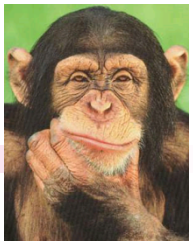
$$= (g^{kss^{-1}} \bmod p) \bmod q$$

$$= (g^{k} \bmod p) \bmod q$$

$$= r$$

Remember: $g^q = 1 \bmod p$ such that we can compute all exponents mod q without changing the result mod p: $g^k \bmod p = g^{k \bmod q} \bmod p$

Trivial Example for DSS



- Let p = 11, q = 5, (5 divides 10)
- Choose e.g. x = 2 and therefore g = 4 g= x^(p-1)/q mod p
- Choose a = 3 and A = g^a mod p = 4^3 = 9 mod 11
- Signing h(m) = 4 works as follows:
  - Choose k in {1,..4}, e.g. k = 3
  - Compute r = g^3 mod p mod q = 4^3 mod 11 mod 5 = 9 mod 5 = 4
  - Compute s = k^{-1} (h(m) + ar) mod q = 2 (4 + 3*4) mod 5 = 2
- I.e. the signature on h(m) = 4 is (4,2)

Note

- Note that if r = 0 mod p mod q or if s = 0 mod q the signature verification will not work
  - Intuitively this is due to the fact that the private key is not used in the generation of s any more if r = 0
- The DSS standard states on this problem
  - "As an option, one may wish to check if r=0 or s=0, a new value of k should be generated and the signature should be recalculated"

Security of DSS

- The same k should not be used twice
- If no hash function is used or the conditions for the length of the r and s are not checked by the signer, existential forgery is possible
- If the discrete logarithm problem is efficiently solvable, then an attacker can totally break DSS signatures (as he can compute the secret key **a** from the public key **A**)

References and Further Reading

- Kaufmann et al., *Network Security*
  - Chapter 6 and 7
- PKCS#1, v.2.2
  - https://tools.ietf.org/html/rfc8017
- DSS specification: FIPS 186-4
  - https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf