CS458: Introduction to Information Security

Notes 4: Public-Key Cryptography

Yousef M. Elmehdwi

Department of Computer Science

Illinois Institute of Technology

yelmehdwi@iit.edu

September 14th 2022

Slides: Modified from Computer Security: Principles and Practice, 4th Edition, Christof Paar and Jan Pelzl, Stephen R. Tate UNC Greensboro & Ewa Syta

Padding

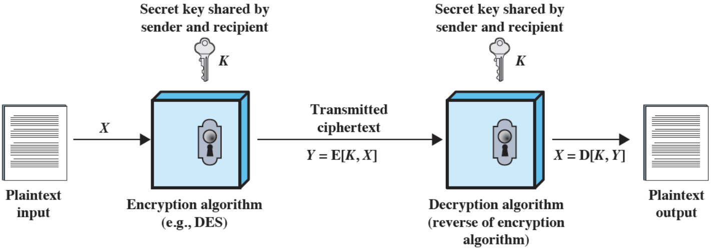
- How to encrypt data that is less than one block using block cipher
- ECB and CBC modes must encrypt full blocks of plaintext!
- What if you have `192` bits of plaintext with AES/CBC? `128+64` bits
- Padding: used in a block cipher where we fill up the blocks with padding bytes.
  - AES uses `128-bit` (`16` bytes), and DES uses `64-bit` blocks (`8` bytes).
- Technique: byte count padding or `PKCS#7 / PKCS#5`[1]:
  - Count how many bytes of padding needed (at least 1), say c
  - Add c bytes each with value `c`
  - Ex (`32-bit` blocks, `hex`):
    - *`42 1a 49 c3 | 21`* becomes *`42 1a 49 c3 | 21 03 03 03`*
  - Only works for padding full bytes!
  - `PKCS#5` padding is identical to `PKCS#7` padding, except that it has only been defined for block ciphers that use a `64-bit` (8-byte) block size, `PKCS#7` padding would work with any block size from 1 to 255 bytes.
    - In practice the two can be used interchangeably.

[1] *Public Key Cryptography Standards*

Outline

- Principles of Asymmetric Cryptography
- Practical Aspects of Public-Key Cryptography
- Important Public-Key Algorithms

Symmetric Cryptography revisited


Secret key shared by sender and recipient
K
Secret key shared by sender and recipient
K
X
Transmitted ciphertext
Y = E[K, X]
X = D[K, Y]
Plaintext input
Encryption algorithm (e.g., DES)
Decryption algorithm (reverse of encryption algorithm)
Plaintext output

- Two properties of symmetric (secret-key) crypto-systems:
  - The same ***secret key K*** is used for encryption and decryption
  - Encryption and Decryption are very similar (or even identical) functions

Symmetric Cryptography: Shortcomings

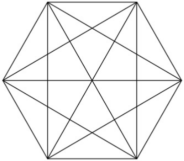
- Symmetric algorithms, e.g., AES or 3DES, are very secure, fast & widespread **but**:

1. Key distribution problem: The secret key must be transported securely
   - i.e., when Alice and Bob communicate using a symmetric system, they need to securely exchange their shared key k_{ab}
2. Key management: In a network, each pair of users requires an individual key

   → n users in the network require $\frac{n \times (n-1)}{2}$ keys, each user stores $(n-1)$ keys
   - If Alice wants to talk to Bob, Carol and Dave, she needs to exchange and maintain k_{ab}, k_{ac}, and k_{ad}

- Example: 6 users (nodes)

  $\frac{6 \times 5}{2} = 15$ keys (edges)



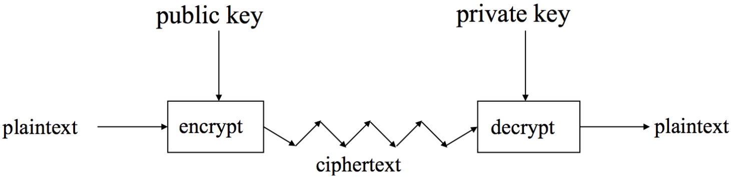
Symmetric Cryptography: Shortcomings

3. No Protection Against Cheating by Alice or Bob: Alice or Bob can **cheat each other**, because they have identical keys.
   - Who is the author of a message encrypted with $\boldsymbol{k_{ab}}$, a key Alice and Bob share?
     - Example: Alice can claim that she never ordered a TV on-line from Bob (he could have fabricated her order). To prevent this: “non-repudiation”

Public-Key Encryption Structure

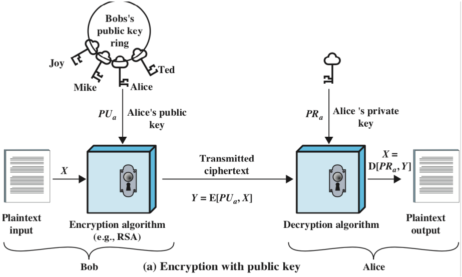
- Publicly proposed by Diffie and Hellman in 1976
- Based on mathematical functions rather than on simple operations on bit patterns
- Asymmetric
  - Uses two separate keys
  - Public key and private key
  - Public key is made public for others to use

Public Key Crypto



- Two keys:
  - Private key known only to owner
  - Public key available to anyone
  - One key pair per person
    - `O(N)` keys

Public Key Crypto



(a) Encryption with public key

- Plaintext: Readable message or data that is fed into the algorithm as input
- Encryption algorithm: Performs transformations on the plaintext
- Public and private key: Pair of keys, one for encryption, one for decryption
- Ciphertext: Scrambled message produced as output
- Decryption algorithm: Produces the original plaintext

Public Key Crypto

- In public key cryptography, an encryption key (which could be the public or private key) is used to encrypt a plaintext message and convert it into an encoded format known as ciphertext.
- Then the other key is used as a decryption key to decrypt this ciphertext so that the recipient can read the original message.
- In short, the main difference between a public key vs private key is that one encrypts while the other decrypts.

Uses of Public Key Crypto

- Q: What is the Usage of Public Key Cryptography?
  - One of the unique advantages of asymmetric encryption using public key pairs is the ability to ensure both the security of encrypted messages and the identity of the sender.
- Encryption
  - Suppose we encrypt m with Bob's public key.
  - Bob's private key can decrypt c to recover m.
  - Q: Why public key for encryption?
- But the order of using the key pairs can flip to ensure the identity of the sender
- Digital Signatures
  - Bob signs by "encrypting" with his private key.
  - Anyone can use Bob's public key to verify the signature (by decrypting with public key).
  - Like a handwritten signature, but way better...
  - Q: Why private key for digital signatures?

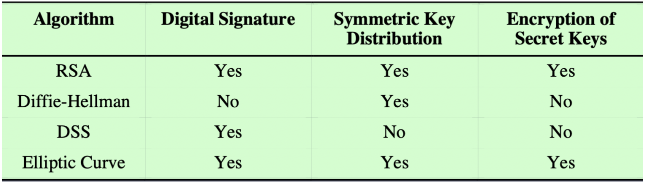
Security of the keys

- Two keys
  - Private key known only to individual
  - Public key available to anyone
  - Given that one key is public, the other one cannot be (easily) computable.
- Based on "trapdoor one-way function"
  - "One-way" means easy to compute in one direction, but hard to compute in other direction (reverse)
    - Easy to calculate $f(x)$ from x
    - Hard to invert: to calculate x from $f(x)$
    - Example:
    - Given p and q, product $N = pq$ easy to compute, but hard to find p and q from N.
  - A trapdoor one-way function has one more property, that with certain knowledge it is easy to invert, to calculate x from $f(x)$
    - i.e., "Trapdoor" is used when creating key pairs. If you have it, you can reverse the process
    - The analogy to a "trapdoor" is something like this: It's easy to fall through a trapdoor, but it's very hard to climb back out and get to where you started unless you have a ladder.

Applications for Public-Key Cryptosystems

- We can classify the use of public-key cryptosystems into three categories:
  - Symmetric Key Distribution (e.g., Diffie-Hellman key exchange, RSA) without a pre-shared secret (key)
  - Nonrepudiation and Digital Signatures[2] (e.g., RSA, DSA or ECDSA) to provide message integrity
    - i.e., Integrity/authentication[3]: encipher using private key, decipher using public one
  - Encryption (e.g., RSA / Elgamal)
    - Confidentiality: encipher using public key, decipher using private key
    - Disadvantage: Computationally very intensive (1000 times slower than symmetric Algorithms!)

Major Public Key Algorithms



Algorithm	Digital Signature	Symmetric Key Distribution	Encryption of Secret Keys
RSA	Yes	Yes	Yes
Diffie-Hellman	No	Yes	No
DSS	Yes	No	No
Elliptic Curve	Yes	Yes	Yes

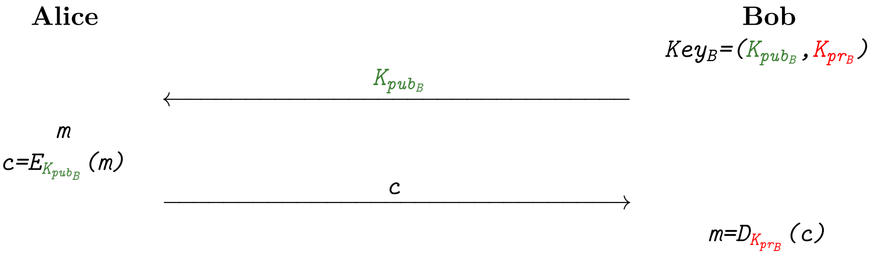
[2] *Digital signatures are used to detect unauthorized modifications to data and to authenticate the identity of the signatory.*

[3] *Message integrity means that a message has not been tampered with or altered.*
The concept of Authentication serves to provide proof that the other side of a communication is indeed who they claim to be, and who you intend for them to be.

Requirements for Public-Key Cryptosystems

- Computationally easy
  - for a party to generate a key pair
  - for sender knowing public key to encrypt messages
  - for receiver knowing private key to decrypt ciphertext
- Computationally infeasible
  - for an opponent knowing only the public key to determine the private key
  - for an opponent knowing the public key and a ciphertext to recover the original message
- Useful if either key can be used for each role (not necessary for all public key applications)

Basic Protocol for Public-Key Encryption



Alice

Bob

$Key_B = (K_{pub_B}, K_{pr_B})$

$\xleftarrow{K_{pub_B}}$

m

$c = E_{K_{pub_B}}(m)$

$\xrightarrow{c}$

$m = D_{K_{pr_B}}(c)$

- Key Distribution Problem solved (at least for now; public keys need to be authenticated)

Basic Key Transport Protocol

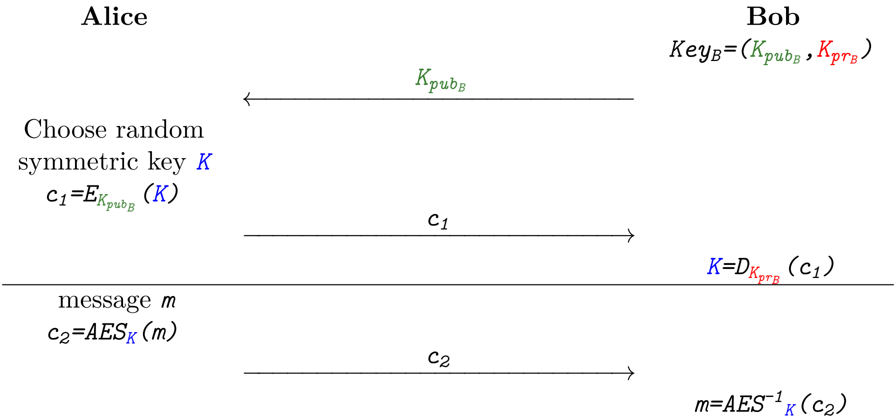
- In practice: Hybrid systems, incorporating asymmetric and symmetric algorithms
  - Examples: SSL/TLS protocol[4] for secure Web connections, or IPsec, the security part of the Internet communication protocol.

1. Key exchange (for symmetric schemes) and digital signatures are performed with (slow) asymmetric algorithms
2. Encryption of data is done using (fast) symmetric ciphers, e.g., block ciphers or stream ciphers

[4] *SSL/TLS (Secure Sockets Layer /Transport Layer Security) is a cryptographic protocol designed to provide communications security over a computer network. The protocol is widely used in applications such as email, instant messaging, and voice over IP, but its use as the Security layer in HTTPS remains the most publicly visible.*

Basic Key Transport Protocol

- Example: Hybrid protocol with AES as the symmetric cipher



Alice

Bob

$Key_B = (K_{pub_B}, K_{pr_B})$

$\xleftarrow{\quad K_{pub_B} \quad}$

Choose random symmetric key K

$c_1 = E_{K_{pub_B}}(K)$

$\xrightarrow{\quad c_1 \quad}$

$K = D_{K_{pr_B}}(c_1)$

message m

$c_2 = AES_K(m)$

$\xrightarrow{\quad c_2 \quad}$

$m = AES^{-1}{}_K(c_2)$

How to build Public-Key Algorithms

- Asymmetric schemes are based on a “one-way function” $f()$:
  - Computing $y = f(x)$ is computationally easy
  - Computing $x = f^{-1}(y)$ is computationally infeasible
- One way functions are based on **mathematically hard problems**. Three main families:
  - Integer-Factorization Schemes:
    - Several public-key schemes are based on the fact that it is difficult to factor large integers, e.g., RSA
    - Given a composite integer n, find its prime factors
    - (Multiply two primes: easy)
  - Discrete Logarithm Schemes
    - Several algorithms, such as Diffie-Hellman key exchange, Elgamal, Digital Signature Algorithm (DSA)
    - Given a, y and n, find x such that $a^x \equiv y \bmod n$
    - (Exponentiation a^x: easy)
  - Elliptic Curves (EC) Schemes
    - Generalization of discrete logarithm
    - e.g., Elliptic Curve Diffie-Hellman key exchange (ECDH) and the Elliptic Curve Digital Signature Algorithm (ECDSA)

Key Lengths and Security Levels

- In cryptography, key size, key length, or key space is the number of bits in a key used by a cryptographic algorithm.
- An algorithm is said to have a "security level of $n\ bit$" if the best known attack requires 2^n steps
- Symmetric algorithms with security level of n have key of length n bit.
- The relationship between cryptographic strength and security is not as straightforward in the asymmetric case
  - The effectiveness of public key cryptosystems depends on the intractability (computational and theoretical) of certain mathematical problems.
  - These problems are time-consuming to solve, but usually faster than trying all possible keys by brute force.
  - Thus, asymmetric keys must be longer for equivalent resistance to attack than symmetric algorithm keys.

Key Lengths and Security Levels

- In symmetric schemes
  - Keys are randomly selected,
  - Key lengths varying between 128 and 256 bits, depending on the required level of security
    - Since 2015, NIST guidance: “the use of keys that provide less than 112 bits of security strength for key agreement is now disallowed.”
- In asymmetric schemes
  - Mathematical relationship between the public and private keys (there is a mathematical pattern between the two keys).
  - This pattern can potentially be exploited by attackers to crack the encryption. Therefore, asymmetric keys need to be much longer to present an equivalent level of security.
    - 80-bit symmetric key and a 1024-bit asymmetric key (RSA) are equivalent in strength.
    - 112-bit symmetric key and a 2048-bit asymmetric key (RSA) offer roughly similar levels of security.
    - 128-bit symmetric key and a 3072-bit asymmetric key (RSA) offer roughly similar levels of security.
  - Key lengths varying between 1,024 and 4,096 bits depending on the required level of security
    - Since 2015, NIST recommends a minimum of 2048-bit keys for RSA (2048-bit keys are sufficient until 2030)

General Facts about Public Key Systems

- Public Key Systems are much slower than Symmetric Key Systems
  - Generally used in conjunction with a symmetric system for bulk encryption
- Public Key Systems are based on "hard" problems
  - Factoring large composites of primes, discrete logarithms, elliptic curves
- Only a handful of public key systems perform both encryption and signatures

Asymmetric Encryption Algorithms

- RSA (Rivest, Shamir, Adleman)
  - Developed in 1977
  - Most widely accepted and implemented approach to public-key encryption
  - Block cipher in which the plaintext and ciphertext are integers between 0 and *n-1* for some *n*.
- Diffie-Hellman key exchange algorithm
  - Enables two users to securely reach agreement about a shared secret that can be used as a secret key for subsequent symmetric encryption of messages
  - Limited to the exchange of the keys

Asymmetric Encryption Algorithms

- Digital Signature Standard (DSS)
    - Originally proposed in 1991, revised in 1993 due to security concerns, and another minor revision in 1996 and 2013
    - Cannot be used for encryption or key exchange
    - Uses an algorithm that is designed to provide only the digital signature function
    - Digital signatures are used to detect unauthorized modifications to data and to authenticate the identity of the signatory.
- Elliptic curve cryptography (ECC)
    - Equal security for smaller bit size than RSA
    - Seen in standards such as IEEE P1363
    - Based on a mathematical construct known as the elliptic curve
    - Bitcoin, for example, uses ECC as its asymmetric cryptosystem because it is so lightweight.

Prime Numbers

- Factors are whole numbers that can be divided evenly into another number.
- Example
  - *1,3,5* and *15* are factors of *15*
- Prime number p
  - p is an integer
  - $p \geq 2$
  - The only divisors of p are *1* and p.
  - i.,e, are numbers with exactly *2* factors.
- Composite number n
  - n is an integer
  - $n > 1$
  - The divisors of n are *1*, n and at least one other number.
  - i.e., have more than *2* factors.
- Example
  - *2,5,11,19* are primes and *4,6,9* are composite numbers.
  - Composite numbers that are a product of two prime numbers.

GCD

- The greatest common divisor *(GCD)* of two positive integers *a* and *b*, denoted *gcd(a,b)*, is the largest positive integer that divides both *a* and *b*.
  - *gcd(12, 20) = 4*
  - *gcd(14, 36) = 2*
- Two integers *a* and *b* are said to be relatively prime or coprime if *gcd(a,b) = 1*
  - *12* and *7*

Modular Arithmetic

- "Wrap around" arithmetic
  - Numbers "wrap around" upon reaching a certain value called the modulus.
  - Example: *12-hour* clock
- Modulo operator for a positive integer n
  - $a \bmod n$ denotes the remainder when a is divided by n.
  - $r \equiv a \bmod n$, that is, $a = r + qn$, where q *is quotient*
  - $5 \equiv 32 \bmod 9$, that is, $32 = 5 + 3 \times 9$
  - $\equiv$ congruence relation (equivalence relation)
- Associativity, Commutativity, and Distributivity hold in Modular Arithmetic
- Inverses also exist in modular arithmetic
  - $a + (-a) \bmod n \equiv 0$
  - $a * a^{-1} \bmod n \equiv 1$

Euler's Totient Function ϕ*(N)*

- Counts the positive integers up to a given integer *N* that are relatively prime to *N*
  - Relatively prime means with no factors in common with *N*
- Example: ϕ*(10)* `= ?`
  - `4` because `{1,3,7,9}` are relatively prime to `10`
- Example: ϕ*(p)* `= ?`, where *p* is a prime
  - *p-1* because all lower numbers are relatively prime

RSA Public-Key Encryption

- By Rivest, Shamir & Adleman of MIT in 1977[5]
- Probably the most commonly used asymmetric cryptosystem today, although elliptic curve cryptography (ECC) becomes increasingly popular
- Unlike the symmetric systems, RSA is based not on substitution and transposition.
- Uses exponentiation of integers modulo a prime
  - RSA is based on arithmetic involving very large integers numbers that are hundreds or even thousands of bits long
- RSA is mainly used for two applications
  - Transport of (i.e., symmetric) keys
  - Digital signatures

[5] A Method for Obtaining Digital Signatures and Public-Key Cryptosystems

RSA Key Generation

- Let p and q be two large prime numbers. Let $N = pq$ be the modulus.
  - p and q very large, chosen at random
  - Important to discard p and q once done
- Choose e relatively prime to $\phi(N) = \phi(p)\phi(q) = (p-1)(q-1)$.
  - ϕ is Euler's totient function: counts the positive integers up to a given integer N that are relatively prime to N
  - i.e., select the public exponent $e \in [1, \phi(N)-1]$ such that $gcd(e, \phi(N)) = 1$
- Find $d \in [1, \phi(N)-1]$ s.t. $e.d \equiv 1 \bmod (p-1)(q-1)$.
  - e and d are multiplicative inverses $\phi(N)$
- *Public key* is (e, N)
- *Private key* is (d, N)
- Remark: $gcd(e, \phi(N)) = 1$ ensures that e has an inverse and, thus, that there is always a private key d

RSA Encryption and Decryption

- Message M is treated as a number.
  - Represent the message as an integer between 0 and $N-1$
  - Must be less than N.
  - Break a long message into a series of blocks, and represent each block as such an integer.
    - Use any standard representation.
    - The purpose here is not to encrypt the message but only to get it into the numeric form necessary for encryption
- To encrypt message M compute $C = M^e \bmod N$
- To decrypt C compute $M = C^d \bmod N$

RSA Keys

- Recall that e and N are public.
- If attacker can factor N, she can use e to easily find d since $ed \equiv 1 \bmod (p-1)(q-1)$.
- Factoring the modulus breaks RSA.
- It is not known whether factoring is the only way to break RSA.

Does RSA Really Work?

- Given $C \equiv M^e \ mod \ N$, show that $C^d \ mod \ N \equiv M^{ed} \equiv M \ mod \ N$
- We'll need Euler's Theorem:
  - If x is relatively prime to n then $x^{\phi(n)} \equiv 1 \ mod \ n$.
- Facts:
  - $ed \equiv 1 \ mod \ (p-1)(q-1)$
  - By definition of "mod", $ed = k(p-1)(q-1) + 1$ for some integer k
  - $\phi(N) = (p-1)(q-1)$
- Then $ed - 1 = k(p-1)(q-1) = k\phi(N)$.
- So, $M^{ed} = M^{(ed-1)+1} = M \cdot M^{ed-1} = M \cdot M^{k\phi(N)} = M \cdot (M^{\phi(N)})^k$

Does RSA Really Work?

- To proof $M \equiv M \cdot (M^{\phi(N)})^k \; mod \; N$, we'll need Euler's Theorem:
  - If $gcd(M,N) = 1$ (M is relatively prime to N) then $M^{\phi(N)} \equiv 1 \; mod \; N$.
  - A minor generalization: $1 \equiv 1^k \equiv (M^{\phi(N)})^k \; mod \; N$
- For the proof we distinguish two cases:
  1. $gcd(M,N) = 1$

     $$M \cdot (M^{\phi(N)})^k \equiv M \cdot 1^k \; mod \; N \equiv M \; mod \; N$$

  2. $gcd(M,N) = gcd(M,p \cdot q) \neq 1$
     - Since p and q are primes, M must have one of them as a factor: $M=r \cdot p$ or $M=s \cdot q$, where r,s are integers s.t. $r<p$ and $s<p$
     - Without loss of generality, assume $M=r \cdot p \Rightarrow gcd(M,q)=1 \Rightarrow$ Euler's Theorem holds $1 \equiv 1^k \equiv (M^{\phi(q)})^k \; mod \; q$
     - $(M^{\phi(N)})^k \equiv (M^{(q-1)(p-1)})^k \equiv ((M^{\phi(q)})^k)^{(p-1)} \equiv 1^{(p-1)} = 1 \; mod \; q$
     - This is equivalent to: $(M^{\phi(N)})^k = 1 + u \cdot q$, where u is some integer.
     - $M \cdot (M^{\phi(N)})^k = M + M \cdot u \cdot q = M + (r \cdot p) \cdot u \cdot q = M + r \cdot u \cdot N$
     - $\Rightarrow d_{k_{pr}} = M \cdot (M^{\phi(N)})^k \equiv M \; mod \; N$

Simple RSA Example

- Select "large" primes $p = 11,\ q = 3$
- Then $N = pq = 33$ and $(p-1)(q-1) = 20$
- Choose $e = 3$ (relatively prime to 20)
- Find d such that $ed \equiv 1 \bmod 20$
  - We find that $d = 7$ works
- *Public key:* $(N,e) = (33,3)$. *Private key:* $d = 7$

Simple RSA Example

- *Public key:* $(N,e) = (33,3)$
- *Private key:* $d = 7$
- Suppose message to encrypt is $M = 8$
- Ciphertext C is computed as
  - $C \equiv M^e \bmod N \equiv 8^3 = 512 \equiv 17 \bmod 33$
- Decrypt C to recover the message M by
  - $M = C^d \bmod N = 17^7 = 410{,}338{,}673 = 12{,}434{,}505 \cdot 33 + 8 \equiv 8 \bmod 33$

Implementation aspects

- The RSA cryptosystem uses only one arithmetic operation (modular exponentiation) which makes it conceptually a simple asymmetric scheme
- Even though conceptually simple, due to the use of very long numbers, RSA is orders of magnitude slower than symmetric schemes, e.g., DES, AES
- When implementing RSA (esp. on a constrained device such as smartcards or cell phones) close attention has to be paid to the correct choice of arithmetic algorithms

More Efficient RSA

- Modular exponentiation of large numbers with large exponents is an expensive operation.
- To make it more manageable, several tricks are used in practice.
- Modular exponentiation example
    - 5^{20} = 95367431640625 $\equiv$ 25 mod 35
    - The naïve approach is to multiply 5 by itself 20 times and then reduce the result mod 35
- When you works with "real" RSA numbers, they get too big to store and would take forever to compute!

More Efficient RSA: Square-and-Multiply

- A better way: square-and-multiply algorithm
- The goal here is to reduce the number of iterations from the order of modulus to the order of the log modulus which equates to the number of bits in the exponent instead of the exponent itself.
- Compute: $x^a \mod n$
- **Basic principle**: Determine binary representation of the exponent, then scan exponent bits from left to right and square/multiply operand accordingly
  - The idea is to build up the exponent one bit at a time.
  - The bits of the exponent a are scanned from left to right.
  - For each bit, a squaring is performed and is followed by a multiplication by x if the bit is equal to 1.
  - Take `mod` whenever possible.

  This method is detailed in next slide.

Efficient RSA: square-and-multiply algorithm

Compute: $x^a \mod n$

```
Require: $x$ $in$ {0, ..., $n-1$} and $a$ = ($a_{l-1}$,...,$a_0$)$_2$
   $r \leftarrow 1$
   for $i$ from $l-1$ downto 0 do
    $r \leftarrow r^2 \mod n$
     if $a_i$ = 1 then
         $r \leftarrow r \times x \mod n$
     end if
   end for
return $r$
```

Efficient RSA: square-and-multiply algorithm

```
Require: $x$ in {0, ..., $n-1$} and a = ($a_{l-1}$,...,$a_0$)$_2$
   $r \leftarrow 1$
   for $i$ from $l-1$ downto 0 do
    $r \leftarrow r^2 \mod n$
      if $a_i$ = 1 then
         $r \leftarrow r \times x \mod n$
      end if
   end for
return $r$
```

- Computes 5^{20} without modulo reduction
  - Binary representation of exponent: $20 = (10100)_2$
  - $(1, 10, 101, 1010, 10100) = (1, 2, 5, 10, 20)$
  - 5^{20}: Note that $20 = 2\times 10$, $10 = 2 \times 5$, $5 = 2\times 2+1$, $2 = 1\times 2$
  - $5^1 \equiv 5 \mod 35$
  - $5^2 = (5^1)^2 = 5^2 \equiv 25 \mod 35$
  - $5^5 = (5^2)^2 \cdot 5^1 = 25^2 \cdot 5 = 3125 \equiv 10 \mod 35$
  - $5^{10} = (5^5)^2 = 10^2 = 100 \equiv 30 \mod 35$
  - $5^{20} = (5^{10})^2 = 30^2 = 900 \equiv 25 \mod 35$
- No huge numbers and it's efficient!

Speed-Up Techniques

- Modular exponentiation is computationally intensive
- Even with the square-and-multiply algorithm, RSA can be quite slow on constrained devices such as smart cards
- Some important tricks: (not covered here)
  - Short public exponent e (e is often set to `65537=2`^{16}`+1` for encryption speed)
    - `65537=2`^{16}`+1` is a good compromise between security and performance.
    - Large enough to avoid the attacks to which small exponents make RSA vulnerable, and
    - Can be computed extremely quickly on binary computers, which often support shift and increment instructions
  - Chinese Remainder Theorem (CRT) (for more efficient decryption)
    - With CRT, the implementation performs the exponentiation $C^d \mod N$ as two separate operations, $C^d \mod p$ and $C^d \mod q$.
    - The two parts are combined at the end.
    - Since p and q are about half the size of N, this speeds up the exponentiation.
  - Exponentiation with pre-computation

RSA in the Real World

- Things are never easy. You cannot use the RSA we talked about for real applications.
  - Deterministic encryption
  - Malleability

- *Public key cryptosystems like RSA are often taught by describing how they work at a high level. Even an algorithm textbook written by one of the inventors of RSA focuses more on exponentiation and how the decryption operation inverts ciphertext than on the underlying security details. However, all public key algorithms have various requirements that are not part of the basic primitive but are vital for security.*

(Plain) RSA is deterministic

- Public key: `(e,N)`. Private key: `d`. Encryption: $E(M) = M^e \bmod N$
- Eve finds matching ciphertexts, she knows the plaintexts match too.
  - Remember ECB?
- Eve can check for potential decryptions.
  - Eve (of course) knows Alice's key.
  - She sees C. She suspects $D_d(C) = M$.
  - She can check! She computes $E_e(M)$ and compares to C

(Plain) RSA is malleable

- Malleability: the property that a ciphertext can be transformed into another ciphertext which decrypts to a related plaintext without knowing the private key
  - i.e., product of two ciphertexts is equal to the encryption of the product of the respective plaintexts
  - it allows an attacker to modify the contents of a message
- Public key: (e,N). Private key: d. Encryption: $E(M) = M^e \bmod N$
- Eve can fiddle with two ciphertexts encrypted under the same key:
  - $E(M_1) \cdot E(M_2) = M_1{}^e \cdot M_2{}^e \bmod N = (M_1 \cdot M_2)^e \bmod N = E(M_1 \cdot M_2)$
- Eve doesn't know M_1 or M_2 but she managed to calculate a function of the plaintext
  - (after decryption, Alice will get the product of M_1 and M_2)

(Plain) RSA is malleable: Example

- Suppose Eve knows that Alice is reporting Eve's salary to Bob's payroll service firm.
  - $C = (Eve\_salary)^e \bmod N$
- Eve intercepts C and sends $2^e \times C \bmod N$ instead.
- Bob decrypts:
  - $(2^e \times C)^d = (2 \times Eve\_salary)^{ed} \bmod N = 2 \times Eve\_salary$

Padding

- To avoid these problems, practical RSA implementations typically embed some form of structured, randomized padding into the value M before encrypting it
  - This padding ensures that M does not fall into the range of insecure plaintexts.
  - Once a given message padded, will encrypt to one of a large number of different possible ciphertexts.

Padding

- We can fix a few issues by introducing padding.
  1. Malleability
     - If we have a strict format for messages, i.e. that the first or last bytes contain a specific value, simply multiplying both message and ciphertext will decrease the probability of creating a valid (in terms of padding) message.
  2. Semantical Security
     - Add randomness such that RSA is not deterministic anymore.

Padding: RSA-OAEP

- Optimal Asymmetric Encryption Padding (OAEP)
- Roughly, to encrypt M
  - Choose random r
  - Encode M as $M' = [X = M \oplus H_1(r)\ ,\ Y = r \oplus H_2(X)\ ]$ where H_1 and H_2 are cryptographic hash functions,
- Usage in RSA
  - The encoded message can then be encrypted with RSA:
    Encrypt $(M')^e \mod n$
  - The deterministic property of RSA is now avoided by using the OAEP encoding.
- To decrypt $M' = [X, Y]$
  - Recover the random string as $r = Y \oplus H_2(X)$
  - Recover the message as $M = X \oplus H_1(r)$
- Unless both X and Y are fully recovered, cannot obtain r, without r, cannot obtain any information of M.

Factoring assumption

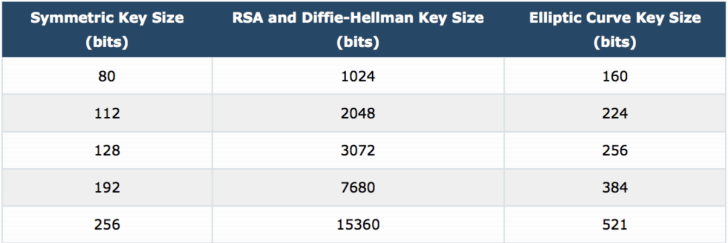
- The factoring problem is to find a prime divisor of a composite number N.
- The factoring assumption is that there is no probabilistic polynomial-time algorithm for solving the factoring problem, even for the special case of an integer N that is the product of just two distinct primes.
- The security of RSA is based on the factoring assumption. No feasible factoring algorithm is known, but there is no proof that such an algorithm does not exist.

How big is big enough?

- The security of RSA depends on N, p, q being sufficiently large.
- What is sufficiently large?
    - Nowadays, N is typically chosen to be *2048 bits* or *3072 bits* long[6].
    - The primes p and q whose product is N are generally chosen be roughly the same length, so each will be about half as long as N.

[6] NIST Special Publication (SP) 800-57 Part 3, Rev. 1.

Key Lengths Comparison



Symmetric Key Size (bits)	RSA and Diffie-Hellman Key Size (bits)	Elliptic Curve Key Size (bits)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	521

Symmetric vs. Asymmetric

- By now you should know that you either get performance or key distribution.
  - Symmetric: fast but need to deal with keys.
  - Asymmetric: slow (orders of magnitude) but resolved key distribution

Security of RSA

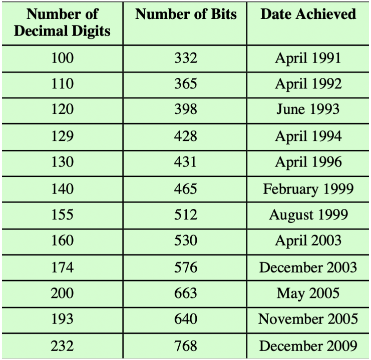
- Brute force
  - Involves trying all possible private keys
- Mathematical attacks
  - There are several approaches, all equivalent in effort to factoring the product of two primes
- Timing attacks
  - These depend on the running time of the decryption algorithm
- Chosen ciphertext attacks
  - This type of attack exploits properties of the RSA algorithm.
  - The cryptanalyst can gather information by obtaining the decryptions of chosen ciphertexts.
  - From these pieces of information, the adversary can attempt to recover the hidden secret key used for decryption

Overview of Mathematical Attacks: Factoring Problem

- Mathematical approaches to attacking RSA
  - Factor N into two prime factors, hence find $\phi(N) = (p-1)\times(q-1)$ and then find $d \equiv e^{-1} \bmod \phi(N)$
  - Determine $\phi(N)$ directly without first determine p and q, then find d
  - Determine d directly without first determine $\phi(N)$
- For a large N with large prime factors, factoring is a hard problem, but not as hard as it used to be.

Progress in Factorization

- For a large N with large prime factors, factoring is a hard problem, but not as hard as it used to be.
- RSA Laboratories issued challenges for the RSA cipher with key sizes of 100, 110, 120, and so on, digits



Number of Decimal Digits	Number of Bits	Date Achieved
100	332	April 1991
110	365	April 1992
120	398	June 1993
129	428	April 1994
130	431	April 1996
140	465	February 1999
155	512	August 1999
160	530	April 2003
174	576	December 2003
200	663	May 2005
193	640	November 2005
232	768	December 2009

- Please watch RSA-129 - Numberphile
- `https://en.wikipedia.org/wiki/RSA_Factoring_Challenge`

Timing Attacks

- Paul Kocher, a cryptographic consultant, demonstrated that a snooper can determine a private key by keeping track of how long a computer takes to decipher messages
- An attacker gains information from the implementation of a cryptosystem rather than from any inherent weakness in the mathematical properties of the system
- Timing attacks are applicable not just to RSA, but also to other public-key cryptography systems
- This attack is alarming for two reasons:
    - It comes from a completely unexpected direction
    - It is a ciphertext-only attack

Timing Attack Countermeasures

- Constant exponentiation time
  - Ensure that all exponentiations take the same amount of time before returning a result
  - This is a simple fix but does degrade performance
- Random delay
  - Better performance could be achieved by adding a random delay to the exponentiation algorithm to confuse the timing attack
  - If defenders do not add enough noise, attackers could still succeed by collecting additional measurements to compensate for the random delays
- Blinding
  - Multiply the ciphertext by a random number before performing exponentiation
  - This process prevents the attacker from knowing what ciphertext bits are being processed inside the computer and therefore prevents the bit-by-bit analysis essential to the timing attack

Key exchange problem

- The key exchange problem is for Alice and Bob to agree on a common random key $\boldsymbol{k}$.
- One way for this to happen is for Alice to choose $\boldsymbol{k}$ at random and then communicate it to Bob over a secure channel.
    - but same issue as with symmetric crypto.
- A better way is to use public key crypto.

The Discrete Log Problem

- The mathematical basis for several algorithms
- For every prime number p, there exists a primitive root[7] (or "generator") α such that

  $$\alpha^1,\ \alpha^2,\ \alpha^3,\ \alpha^4, \dots, \alpha^{p-2},\ \alpha^{p-1} \text{ (all taken } mod\ p\text{)}$$

  are all distinct values (so this is a permutation of $1, 2, 3, \dots, p-1$)
- Example: 3 is a primitive root of 17, with powers:

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
3^i mod 17	3	9	10	13	5	15	11	16	14	8	7	4	12	2	6	1

- α and p are global public parameters
- Discrete Logarithm problem
  - Given integers n, α and prime number p, compute i such that $n \equiv \alpha^i \ mod\ p$
  - Solutions known for small p
  - Solutions computationally infeasible as p grows large

[7] *Primitive root: When it raised to any exponent i, the solution distributes uniformly/is equally likely to be an interger between 0 and p.*

Diffie-Hellman Key Exchange (DHKE)

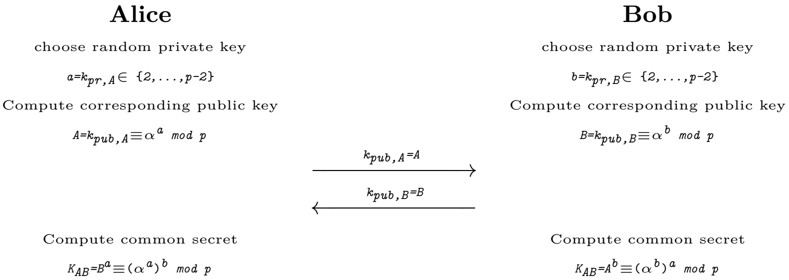
- First published public-key algorithm
- By Diffie and Hellman in 1976 along with the exposition of public key concepts
- Used in a number of commercial products
- Practical method to exchange a secret key securely that can then be used for subsequent encryption of messages
- Security relies on difficulty of computing discrete logarithms
- A “key exchange” algorithm:
    - Used to establish a shared symmetric key.
    - Called a symmetric key exchange protocol
    - Not for encrypting or signing.
- The point is to agree on a key that two parties can use for a symmetric encryption, in such a way that an eavesdropper cannot obtain the key

Diffie-Hellman Key Exchange (DHKE)

- Let p be prime, α be a generator, $\alpha < p$
- Alice selects her private value a
- Bob selects his private value b
- Alice sends $\alpha^{a} \bmod p$ to Bob
- Bob sends $\alpha^{b} \bmod p$ to Alice
- Both compute shared secret, $\alpha^{ab} \bmod p$
- Shared secret can be used as symmetric key

Diffie-Hellman Key Exchange (DHKE)

- Public: p (prime) and $\alpha \bmod p$
- Private: Alice's exponent a, Bob's exponent b



Alice

choose random private key

$a = k_{pr,A} \in \{2, \ldots, p-2\}$

Compute corresponding public key

$A = k_{pub,A} \equiv \alpha^{a} \bmod p$

Bob

choose random private key

$b = k_{pr,B} \in \{2, \ldots, p-2\}$

Compute corresponding public key

$B = k_{pub,B} \equiv \alpha^{b} \bmod p$

$\xrightarrow{k_{pub,A} = A}$

$\xleftarrow{k_{pub,B} = B}$

Compute common secret

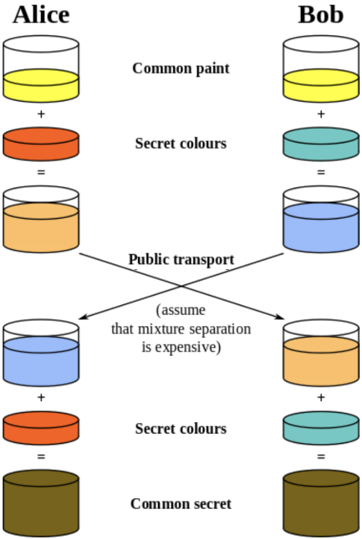
$K_{AB} = B^{a} \equiv (\alpha^{a})^{b} \bmod p$

Compute common secret

$K_{AB} = A^{b} \equiv (\alpha^{b})^{a} \bmod p$

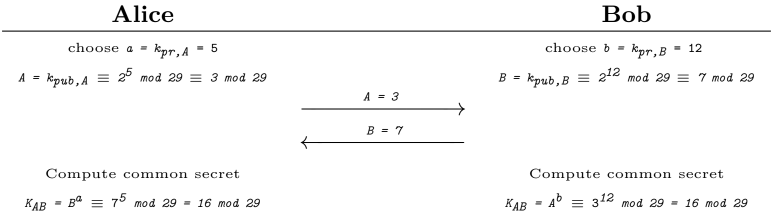
- The key $K = K_{AB} = \alpha^{ab} \bmod p$ can now be used to establish a secure communication between Alice and Bob
  - e.g., by using K_{AB} as key for a symmetric algorithm like AES or 3DES

Diffie-Hellman Key Exchange (DHKE)



Diffie-Hellman Key Exchange: Example

- Have
  - Prime number $p = 29$
  - Primitive root $\alpha = 2$



Alice

choose $a = k_{pr,A} = 5$

$A = k_{pub,A} \equiv 2^5 \bmod 29 \equiv 3 \bmod 29$

Bob

choose $b = k_{pr,B} = 12$

$B = k_{pub,B} \equiv 2^{12} \bmod 29 \equiv 7 \bmod 29$

$\xrightarrow{A = 3}$

$\xleftarrow{B = 7}$

Compute common secret

$K_{AB} = B^a \equiv 7^5 \bmod 29 = 16 \bmod 29$

Compute common secret

$K_{AB} = A^b \equiv 3^{12} \bmod 29 = 16 \bmod 29$

Diffie-Hellman: What can Eve do?

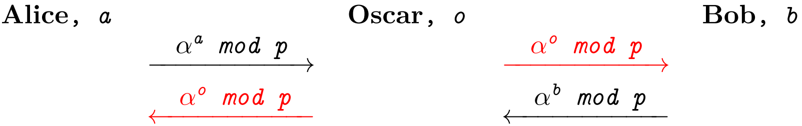
- Suppose Bob and Alice use Diffie-Hellman to determine key $K{=}\alpha^{ab} \bmod p$
- Eve can see $\alpha^{a} \bmod p$ and $\alpha^{b} \bmod p$
    - But... $\alpha^{a} \cdot \alpha^{b} \bmod p = \alpha^{a+b} \bmod p \neq \alpha^{ab} \bmod p$
- If Eve can find a or b, she gets K

Security of DH key exchange

- The security of this protocol relies on Eve's presumed inability to compute K from A and B and the public information p and α.
- This is sometime called the Diffie-Hellman problem and, like discrete log, is believed to be intractable.
  - Compute $\alpha^{ab} \bmod p$ given $\alpha^{a} \bmod p$ and $\alpha^{b} \bmod p$ with given α and p are known.
  - DHKE is believed to be secure for large enough p
- Certainly the Diffie-Hellman problem is no harder that discrete log. However, it is not known to be as hard as discrete log.
  - It is unknown if this could be done without solving discrete logarithm first.
  - If the only way of solving DHP requires the DLP, one would way that "the DHP is equivalent to the DLP". However, this is not proven (yet)

Diffie-Hellman: Man-in-the-Middle Attack

- Subject to man-in-the-middle attack.
- Oscar sits between Alice and Bob, and replaces all messages on either direction.
    - Neither Alice and Bob will be able to detect it!



Alice, a **Oscar**, o **Bob**, b

$\alpha^a \bmod p \rightarrow$

$\leftarrow \alpha^o \bmod p$

$\alpha^o \bmod p \rightarrow$

$\leftarrow \alpha^b \bmod p$

- Oscar shares secret α^{ao} with Alice.
- Oscar shares secret α^{bo} with Bob.
- Alice and Bob don't know Oscar exists (Man-in-the-Middle)

Diffie-Hellman: Man-in-the-Middle

- How to prevent Man-in-the-Middle attack?
  - Encrypt DH exchange with symmetric key.
  - Encrypt DH exchange with public key.
  - Sign DH values with private key.
  - Other?
- At this point, DH may look pointless
  - but it's not (more on this later).

Summary

- Public-key algorithms have capabilities that symmetric ciphers don't have, in particular digital signature and key establishment functions.
- Public-key algorithms are computationally intensive (a nice way of saying that they are slow), and hence are poorly suited for bulk data encryption.
- Only three families of public-key schemes are widely used. This is considerably fewer than in the case of symmetric algorithms.
- The Diffie-Hellman protocol is a widely used method for key exchange.
- The discrete logarithm problem is one of the most important one-way functions in modern asymmetric cryptography. Many public-key algorithms are based on it.
- For a better long-term security, at least, a prime of length ***2048 bits*** should be chosen.

Reading

- *Computer Security: Principles and Practice*
  - Chapter 2 and Chapter 21
- Optional: *Understanding Cryptography: A Textbook for Students and Practitioners*
  - Chapters 6,7, and 8