Chapter 4
Asymmetric Cryptography

❏ Introduction
❏ Encryption: RSA
❏ Key Exchange: Diffie-Hellman

Asymmetric Cryptography

❏ General idea:
  ❏ Use two different keys $-K$ and $+K$ for encryption and decryption
  ❏ Given a random ciphertext $c = E(+K, m)$ and $+K$ it should be infeasible to compute $m = D(-K, c) = D(-K, E(+K, m))$
    ▪ This implies that it should be infeasible to compute $-K$ when given $+K$
  ❏ The key $-K$ is only known to one entity A and is called A's **private key $-K_A$**
  ❏ The key $+K$ can be publicly announced and is called A's **public key $+K_A$**

❏ Applications:
  ❏ **Encryption:** If B encrypts a message with A's public key $+K_A$, he can be sure that only A can decrypt it using $-K_A$
  ❏ **Signing:** If A encrypts a message with his own private key $-K_A$, everyone can verify this signature by decrypting it with A's public key $+K_A$

❏ Attention: It is crucial, that everyone can verify that he really knows A's public key and not the key of an adversary!

Design of Asymmetric Cryptosystems

❑ Difficulty: Find an **encryption algorithm** and a **key generating method** to construct two keys -K, +K such that it is not possible to decipher E(+K, m) with the knowledge of +K

  ❑ Constraints:

  ▪ The key length should be "manageable"

  ▪ Encrypted messages should not be arbitrarily longer than unencrypted messages (we would tolerate a small constant factor)

  ▪ Encryption and decryption should not consume too much resources (time, memory)

  ❑ Basic idea: Take a problem in the area of mathematics / computer science, that is *hard* to solve when knowing only +K, but *easy* to solve when knowing -K

  ▪ Knapsack problems: basis of first working algorithms, which were unfortunately almost all proven to be insecure

  ▪ Factorization problem: basis of the **RSA algorithm**

  ▪ Discrete logarithm problem: basis of **Diffie-Hellman** and ElGamal

RSA – Mathematical Background (Modular Arithmetic)

❑ We say *b is congruent a mod n* if it has the same remainder like *a* when divided by *n*. So, *n* divides *(a-b)*, and we write **b ≡ a mod n**

  ❑ E.g., $4 \equiv 11 \bmod 7$, $25 \equiv 11 \bmod 7$

❑ Greatest common divisor

  ❑ Let $a, b \in Z$ and $d = \gcd(a, b)$. Then there exists $m, n \in Z$ such that:
  d = m × a + n × b

❑ **Euler totient of *n*: Φ(*n*)**

  ❑ Let Φ(*n*) denote the number of positive integers less than *n* and relatively prime to *n*

  ▪ Examples: $\Phi(4) = 2$, $\Phi(15) = 8$

  ▪ If *p* is prime $\Rightarrow \Phi(p) = p - 1$

  ❑ Let *n* and *b* be positive and relatively prime integers, i.e. $\gcd(n, b) = 1$
  $\Rightarrow$ **$b^{\Phi(n)} \equiv 1 \bmod n$**

❑ Euclidean Algorithm

  ❑ The algorithm *Euclid* given *a, b* computes $\gcd(a, b)$

  ❑ *int Euclid(int a, b) {*
  if   (b = 0) { return(a);}
  return(Euclid(b, a MOD b);
  }

❑ **Extended Euclidean Algorithm**

  ❑ The algorithm *ExtEuclid* given *a, b* computes *d, m, n* such that:
  $d = \gcd(a, b) = m \times a + n \times b$

  ❑ *struct{int d, m, n} ExtEuclid(int a, b) {*
  int d, d', m, m', n, n';
  if (b = 0) {return(a, 1, 0); }
  (d', m', n') = ExtEuclid(b, a MOD b);
  (d, m, n) = (d', n', m' - ⌊a / b⌋ × n');
  return(d, m, n);
  }

For more information, please refer to undergraduate CS classes or to the NetSec slides WS 2006/2007

RSA in a Nutshell

- ❑ Invented by R. Rivest, A. Shamir and L. Adleman [RSA78]

- ❑ *Key generation*
  - ❑ Select p, q p and q both prime, $p \neq q$
  - ❑ Calculate n $n = p$ x q
  - ❑ Calculate $\Phi(n)$ $\Phi(n) = (p - 1)(q - 1)$
  - ❑ Select integer e $\gcd(\Phi(n), e) = 1$; $1 < e < \Phi(n)$
  - ❑ Calculate d d x e mod $\Phi(n) = 1$ (extended Euclid)
  - ❑ Public key $+K = \{e, n\}$
  - ❑ Private key $-K = \{d, n\}$

- ❑ *Encryption*
  - ❑ Plaintext $M < n$ (what about 0 , 1, …?)
  - ❑ Ciphertext $C = M^e$ mod n

- ❑ *Decryption*
  - ❑ Ciphertext   C
  - ❑ Plaintext $M = C^d$ mod n

RSA – Encryption / Decryption

- ❑ Let p, q be distinct large primes and $n = p \times q$. Assume, we have also two integers e and d such that $d \times e \equiv 1$ mod $\Phi(n)$

- ❑ Let M be an integer that represents the message to be encrypted, with M positive, smaller than and relatively prime to n.
  - ❑ Example: Encode with <blank> = 99, A = 10, B = 11, ..., Z = 35
    So "HELLO" would be encoded as 1714212124.
    If necessary, break M into blocks of smaller messages: 17142 12124

- ❑ To encrypt, compute: $C = M^e$ MOD n
  - ❑ This can be done efficiently using the *square-and-multiply algorithm*
- ❑ To decrypt, compute: $M' = C^d$ MOD n

- ❑ Proof

  $d \times e \equiv 1$ mod $\Phi(n) \Rightarrow \exists\, k \in Z: (d \times e) - 1 = k \times \Phi(n) \Leftrightarrow (d \times e) = k \times \Phi(n) + 1$

  we have: $M' \equiv E^d \equiv M^{(e \times d)} \equiv M^{(k \times \Phi(n) + 1)} \equiv 1^k \times M \equiv M$ mod n ∎

RSA – Encryption / Decryption

- As $(d \times e) = (e \times d)$ the operation also works in the opposite direction, that means you can encrypt with d and decrypt with e

- This property allows to use the same keys d and e for:
  - Receiving messages that have been encrypted with one's public key
  - Sending messages that have been signed with one's private key

RSA – Security

- The security of the scheme lies in the difficulty of factoring $n = p \times q$ as it is easy to compute $\Phi(n)$ and then d, when p and q are known

- This class will not teach why it is difficult to factor large n's, as this would require to dive deep into mathematics
  - If p and q fulfill certain properties, the best known algorithms are exponential in the number of digits of n
    - Please be aware that if you choose p and q in an "unfortunate" way, there might be algorithms that can factor more efficiently and your RSA encryption is not at all secure:
      - Thus, p and q should be about the same bit length and sufficiently large
      - $(p - q)$ should not be too small
      - If you want to choose a small encryption exponent, e.g. 3, there might be additional constraints, e.g. gcd(p - 1, 3) = 1 and gcd(q - 1, 3) = 1
    - The security of RSA also depends on the primes generated being truly random (like every key creation method for any algorithm)
  - Moral: If you are to implement RSA by yourself, ask a mathematician or better a cryptographer to check your design

RSA - Security

- ❑ Side channel attacks
    - ❑ Optimizations for use of RSA in embedded systems depend on the Chinese remainder theorem (CRT)
        - ■ Applications
            - – Smart cards (token, banking)
            - – Pay-per-view TV
            - – and many others…
        - ■ Use (and storage) of p and q allows to calculate m^e mod p, which can be efficiently manipulated to compute m^e mod n
        - ■ Introducing computation errors allows to reveal the prime p
          p = gcd(s'-s,n) with s' and s being the bogous and correct signatures
    - ❑ Implementation using square and multiply
        - ■ Most RSA implementations rely on the square-and-multiply algorithm for the exponentiations
        - ■ Timing attacks can by used to "guess" the private key

[A. G. Voyiatzis, "An Introduction to Side Channel Cryptanalysis of RSA", ACM Crossroads, vol. 11.3, 2004]

Diffie-Hellman – Mathematical Background

- ❑ *Finite groups*
    - ❑ Abelian group: set S and a binary and a binary operation $\oplus$: (S, $\oplus$), with the following properties: closure, identity, associativity, commutativity and inverse elements
    - ❑ Finite group: Abelian group plus finite set of elements , i.e. $|S| < \infty$

- ❑ *Primitive root, generator*
    - ❑ Let (S, •) be a group, $g \in S$ and $g^a := g \bullet g \bullet ... \bullet g$ (a times with $a \in Z^+$)
      Then g is called a *primitive root* of (S, •) :$\Leftrightarrow$ {g^a | $1 \leq a \leq |S|$} = S
    - ❑ Examples:
        - ■ 1 is a primitive root of (Z_n, $+_n$)
        - ■ 3 is a primitive root of (Z^*_7, $\times_7$)
    - ❑ (Z^*_n, $\times_n$) does have a primitive root $\Leftrightarrow$ $n \in$ {2, 4, p, $2 \times p^e$} where p is an odd prime and $e \in Z^+$

Diffie-Hellman – Mathematical Background

❑ Definition: ***discrete logarithm***
   ❑ Let p be prime, g be a primitive root of $(Z^*_p, \times_p)$ and c be any element of Z^*_p. Then there exists z such that: $g^z \equiv c \bmod p$
   z is called the *discrete logarithm* of c modulo p to the base g
   ❑ Example:
      ▪ 6 is the discrete logarithm of 1 modulo 7 to the base 3 as $3^6 \equiv 1 \bmod 7$

   ❑ The calculation of the discrete logarithm z when given g, c, and p is a computationally difficult problem and the asymptotical runtime of the best known algorithms for this problem is exponential in the bit length of p

For more information, please refer to undergraduate
CS classes or to the NetSec slides WS 2006/2007

Diffie-Hellman Key Exchange

❑ The Diffie-Hellman key exchange was first published in the landmark paper [DH76], which also introduced the fundamental idea of asymmetric cryptography
❑ The DH exchange in its basic form enables two parties A and B to agree upon a *shared secret* using a public channel:
   ❑ *Public channel* means, that a potential attacker E (E stands for eavesdropper) can ***read*** all messages exchanged between A and B
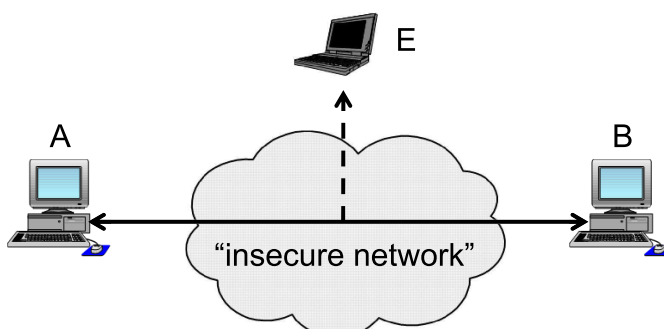
Key Exchange Procedure

- A chooses a prime p, a primitive root g of Z^*_p, and a random number q:
  - A and B can agree upon the values p and g prior to any communication, or A can choose p and g and send them with his first message
  - A computes $v = g^q$ MOD p and sends to B: $\{p, g, v\}$

- B chooses a random number r:
  - B computes $w = g^r$ MOD p and sends to A: $\{p, g, w\}$ (or just $\{w\}$)

- Both sides compute the common secret:
  - A computes $s = w^q$ MOD p
  - B computes $s' = v^r$ MOD p

- As $g^{(q \times r)}$ MOD $p = g^{(r \times q)}$ MOD p it holds: $s = s'$

- Remark: In practice the number g does not necessarily need to be a primitive root of p, it is sufficient if it generates a large subgroup of Z^*_p

Diffie-Hellman Key Exchange

- The mathematical basis for the DH exchange is the problem of finding *discrete logarithms in finite fields*
  - An attacker Eve (E) who is listening to the public channel can only compute the secret s, if she is able to compute either q or r which are the discrete logarithms of v, w modulo p to the base g

- It is important, that A and B can be sure, that the attacker is not able to **alter** messages, as in this case he might launch a *man-in-the-middle attack*

- Remark: The DH exchange is *not* an asymmetric encryption algorithm, but is nevertheless introduced here as it goes well with the mathematical flavor of this lecture... :o)

❑ Eve generates to random numbers *q'* and *r'*:

  ❑ Eve computes $v' = g^{q'}$ MOD *p* and $w' = g^{r'}$ MOD *p*

❑ When A sends {*p, g, v*} she intercepts the message

  ❑ Then, E sends to B: {*p, g, v'* }

❑ When B sends {*p, g, w*} she intercepts the message as well

  ❑ E sends to A: {*p, g, w'* }

❑ When the supposed "shared secret" is computed we get:

  ❑ A computes $s_1 = w'^q$ MOD *p* = $v^{r'}$ MOD *p* the latter computed by E

  ❑ B computes $s_2 = v'^r$ MOD *p* = $w^{q'}$ MOD *p* the latter computed by E

  ❑ So, in fact A and E have agreed upon a shared secret s_1, similarly E and B have agreed upon a shared secret s_2

❑ E can now use the "shared secret" to intercept all the messages encrypted by this key to forge and re-encrypt the messages without being noticed

Diffie-Hellman Key Exchange
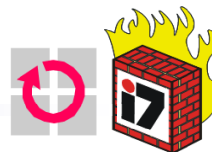
❑ Two countermeasures against the man-in-the-middle attack:

  ❑ The shared secret is *"authenticated"* after it has been agreed upon

    ▪ We will treat this in the section on key management

  ❑ A and B use a so-called *interlock protocol* after agreeing on a shared secret:

    ▪ For this they have to exchange messages that E has to relay before she can decrypt / re-encrypt them

    ▪ The content of these messages has to be checkable by A and B

    ▪ This forces E to invent messages and she can be detected

    ▪ One technique to prevent E from decrypting the messages is to split them into two parts and to send the second part before the first one.

      – If the encryption algorithm used inhibits certain characteristics E can not encrypt the second part before she receives the first one.

      – As A will only send the first part after he received an answer (the second part of it) from B, E is forced to invent two messages, before she can get the first parts.

Conclusion

- Asymmetric cryptography allows to use two different keys for:
    - Encryption / Decryption
    - Signing / Verifying
- The most practical algorithms that are still considered to be secure are:
    - RSA, based on the difficulty of factoring
    - Diffie-Hellman (not an asymmetric algorithm, but a key agreement protocol)
    - ElGamal, like DH based on the difficulty of computing discrete logarithms
- As their security is entirely based on the difficulty of certain mathematical problems, algorithmic advances constitute their biggest threat

- Practical considerations:
    - Asymmetric cryptographic operations are magnitudes slower than symmetric ones
    - Therefore, they are often not used for encrypting / signing bulk data
    - Symmetric techniques are used to encrypt / compute a cryptographic hash value and asymmetric cryptography is just used to encrypt a key / hash value

Summary (what do I need to know)

- Principles of asymmetric cryptography
    - +K, -K for encryption and signing
    - Mathematical problems that are hard to solve
    - Factorization, discrete logarithm

- RSA
    - Key generation
    - Encryption / decryption (how?, why does it work?)

- Diffie-Hellman key exchange
    - Key generation procedure
    - Man-in-the-middle attack

Additional References

[Bre88a]   D. M. Bressoud. *Factorization and Primality Testing.* Springer, 1988.

[Cor90a]   T. H. Cormen, C. E. Leiserson, R. L. Rivest. *Introduction to Algorithms.* The MIT Press, 1990.

[DH76]   W. Diffie, M. E. Hellman. *New Directions in Cryptography.* IEEE Transactions on Information Theory, IT-22 , pp. 644-654, 1976.

[ElG85a]   T. ElGamal. *A Public Key Cryptosystem and a Signature Scheme based on Discrete Logarithms.* IEEE Transactions on Information Theory, Vol.31, Nr.4, pp. 469-472, July 1985.

[Kob87a]   N. Koblitz. *A Course in Number Theory and Cryptography.* Springer, 1987.

[Men93a]   A. J. Menezes. *Elliptic Curve Public Key Cryptosystems.* Kluwer Academic Publishers, 1993.

[Niv80a]   I. Niven, H. Zuckerman. *An Introduction to the Theory of Numbers.* John Wiley & Sons, 4th edition, 1980.

[RSA78]   R. Rivest, A. Shamir und L. Adleman. *A Method for Obtaining Digital Signatures and Public Key Cryptosystems.* Communications of the ACM, February 1978.