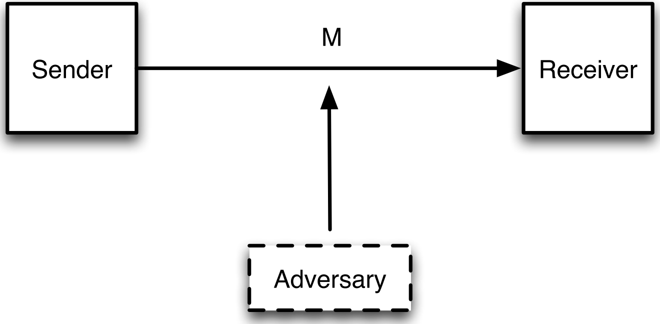
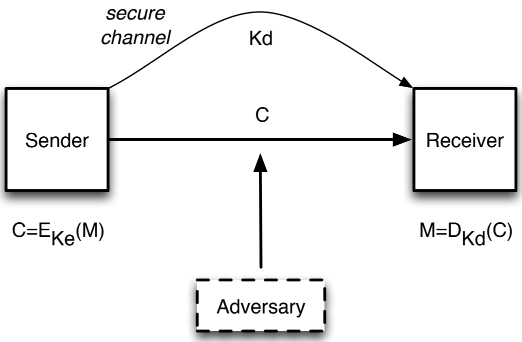
Public-Key Cryptography

Secure Communication over an Insecure Channel



Secret-Key Cryptography



Encryption and decryption functions: $E(\cdot)$ & $D(\cdot)$
Encryption and decryption keys: K_e & K_d
Plaintext and ciphertext: M & C

Secret-Key Cryptography

- $C = E_{K_e}(M)$ and $M = D_{K_d}(C)$
- Either $E(\cdot) = D(\cdot)$ and $K_e \neq K_d$

  K_d is easily deduced from K_e
  K_e is easily deduced from K_d
- Or $E(\cdot) \neq D(\cdot)$ and $K_e = K_d$

  $D(\cdot)$ is easily deduced from $E(\cdot)$
  $E(\cdot)$ is easily deduced from $D(\cdot)$

Example: Hill Algebra

- Encoding: $\{a, b, \ldots, z\} \longrightarrow \{0, 1, \ldots, 25\}$
- Select a $d \times d$ matrix $\mathcal{A}$ of integers and find its inverse $\mathcal{A}^{-1}$ mod 26
- For example, for $d = 2$

$$\mathcal{A} = \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix} \quad \text{and} \quad \mathcal{A}^{-1} = \begin{bmatrix} 15 & 17 \\ 20 & 9 \end{bmatrix}$$

Verify:

$$\begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} 15 & 17 \\ 20 & 9 \end{bmatrix} = \begin{bmatrix} 105 & 78 \\ 130 & 79 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \pmod{26}$$

Hill Cipher

- Encryption function: $c = E(m) = \mathcal{A}\, m \pmod{26}$
- Decryption function: $m = D(c) = \mathcal{A}^{-1}\, c \pmod{26}$
- m and c are $d \times 1$ vectors of plaintext and ciphertext letter encodings
- Encryption key K_e: $\mathcal{A}$
- Decryption key K_d: $\mathcal{A}^{-1} \pmod{26}$
- $\mathcal{A}$ and $\mathcal{A}^{-1}$ are $d \times d$ matrices such that $\det(\mathcal{A}) \neq 0 \pmod{26}$ and $\mathcal{A}^{-1}$ is the inverse of $\mathcal{A}$ mod 26

Secret-Key versus Public-Key Cryptography

- Secret-Key Cryptography:
  - Requires establishment of a secure channel for key exchange
  - Two parties cannot start communication if they never met
  - Secure communication of n parties requires $n(n-1)/2$ keys
  - Keys are "shared", rather than "owned" (secret vs private)
- Public-Key Cryptography:
  - No need for a secure channel
  - May require establishment of a public-key directory
  - Two parties can start communication even if they never met
  - Secure communication of n parties requires n keys
  - Keys are "owned', rather than "shared"
  - Ability to "sign" digital data (secret vs private)

Public-Key Cryptography

- The functions $C(\cdot)$ and $D(\cdot)$ are inverses of one another

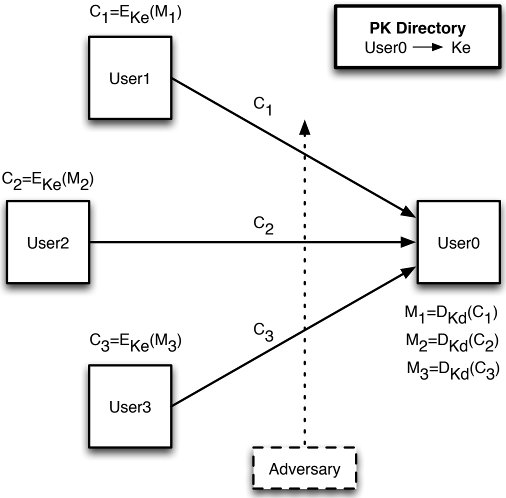
$$C = E_{K_e}(M) \quad \text{and} \quad M = D_{K_d}(C)$$

- Encryption and decryption processes are **asymmetric**:

$$K_e \neq K_d$$

- K_e is **public**, known to everyone
- K_d is **private**, known only to the user
- K_e may be easily deduced from K_d
- However, K_d is **NOT easily** deduced from K_e

Public-Key Cryptography



Public-Key Cryptography

- The User publishes his/her own public key: K_e
- Anyone can obtain the public key K_e and can encrypt a message M, and send the ciphertext to the User

$$C = E_{K_e}(M)$$

- The private key is known only to the User: K_d
- Only the User can decrypt the ciphertext to get the message

$$M = D_{K_d}(C)$$

- The adversary may be able to block the ciphertext, but cannot decrypt

Public-Key Cryptography

- A public-key cryptographic algorithm is based on a function $y = f(x)$ such that
  Given x, computing y is EASY: $y = f(x)$
  Given y, computing x is HARD: $x = f^{-1}(y)$

$$x \quad \underset{\text{hard}}{\overset{\text{easy}}{\rightleftarrows}} \quad y = f(x)$$

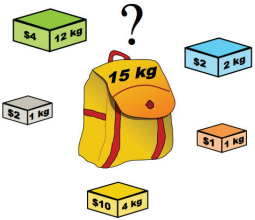
- Such functions are called **one-way**
- In order to decide what is hard: Theory of complexity could help

One-Way Functions for PKC

- However, a one-way function is difficult for anyone to invert
- What we need: a function easy to invert for the legitimate receiver of the encrypted message, but for everyone else: hard
- Such functions are called **one-way trapdoor functions**
- In order to build a public-key encryption algorithm, we need a one-way trapdoor function
- Once that is understood (in around 1975-1976), researchers looked for such special functions which are either based on the known one-way functions or some other constructions

Knapsack Problem

- A problem from combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible



- The decision problem form of the knapsack problem: "Can a value of at least V be achieved without exceeding the weight X?" is NP-complete
- There is no known polynomial-time algorithm on all cases

0-1 Knapsack Problem

- 0-1 Knapsack Problem: Given a set of integers $A = \{a_0, a_1, \ldots, a_{n-1}\}$ and an integer X, is there a subset B of A such that the sum of the elements in the subset B is exactly X?

$$\sum_{a_i \in B} a_i = X$$

- For a randomly generated set of a_is: A hard knapsack problem
- Consider $A = \{3, 4, 5, 12, 13\}$ and $X = 19$
- We need to try all subsets of A to find out which one sums to 19

Knapsack as a One-Way Function

- EASY: Given a randomly generated $A = \{a_0, a_1, \ldots, a_{n-1}\}$, select a subset $B \subset A$, and find the sum

$$X = \sum_{a_i \in B} a_i$$

- HARD: Given a randomly generated $A = \{a_0, a_1, \ldots, a_{n-1}\}$, and the sum X, determine the subset B such that

$$X = \sum_{a_i \in B} a_i$$

Trapdoor Knapsack

- What we need: A knapsack problem is that is hard for everyone else, except the intended recipient
- Consider the set A has the **super-increasing property**:

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

- $A = \{1, 2, 4, 8, 16, 32, 64, \ldots\}$: Super-increasing

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

- Given X, it would be trivial to determine if any of a_is is to be included: if there is a 1 in the binary expansion of X in the ith position

Trapdoor Knapsack

- Take an easy knapsack and disguise it
- Consider $A = \{1, 2, 4, 8, 16\}$
- Select a prime p larger than the sum 31, for example $p = 37$
- Select t and compute t^{-1} mod p, for example, $t = 17$ and $t^{-1} = 24$
- Produce a new knapsack vector A' from A such that

$$a'_i = a_i \cdot t \pmod{p}$$

This gives $A' = \{17, 34, 31, 25, 13\}$, which is not super-increasing

Trapdoor Knapsack

- However, with the special trapdoor information $t = 17$ and $t^{-1} = 24$, and $p = 37$, we can convert this problem to a super-increasing knapsack
- Given A' and $X' = 72$, is there a subset of A' summing to X'?
- First turn the problem into a super-increasing knapsack version, by simply finding X from X' as $X = X' \cdot t^{-1} = 72 \cdot 24 = 26 \pmod{37}$
- Solve the super-increasing knapsack $A = \{1, \mathbf{2}, 4, \mathbf{8}, \mathbf{16}\}$ and $X = 26$, which is easily obtained from the binary expansion of $26 = 16 + 8 + 2$
- This gives the solution for $A' = \{17, \mathbf{34}, 31, \mathbf{25}, \mathbf{13}\}$ and $X' = 72$ as $72 = 34 + 25 + 13$

Trapdoor Knapsack Public-Key Encryption

- User A:
  Selects a super-increasing vector A with $|A| = n > 100$
  Selects a prime p larger than the sum $\sum_{i=0}^{n-1} a_i$
  Selects t and t^{-1} such that $t \cdot t^{-1} = 1 \bmod p$
  Obtains the hard knapsack A' from A using $a_i' = a_i \cdot t \bmod p$
  Publishes A' in a server and keeps A, t, t^{-1}, and p **secret**
- User B:
  Wants to send a message M to User A
  Breaks the message M into n bits: $(m_{n-1} m_{n-2} \cdots m_1 m_0)$
  Obtains A' from the public key server
  Computes the ciphertext C' as $C' = \sum_{i=0}^{n-1} m_i a_i'$
  Sends the ciphertext C' to User A

Trapdoor Knapsack Public-Key Encryption

- User A:
  Receives the ciphertext $'C$
  Computes $C = C' \cdot t^{-1} \bmod p$
  Solves the a super-increasing vector A and C
  Uses this solution to obtain the plaintext M
- Therefore, we obtained the Knapsack public-key encryption algorithm
- Our objective: User A faces an easy problem due to the trapdoor information, while everyone else faces a computationally difficult problem
- We accomplished the first half of our objective nicely: The super-increasing knapsack problem is indeed easy to solve

Trapdoor Knapsack Public-Key Encryption

- The trapdoor knapsack public-key encryption method was proposed by Ralph Merkle and Martin Hellman in 1978 (IEEE Tran. Information Theory)
- In 1984, Adi Shamir published a polynomial-time algorithm for breaking the Merkle-Hellman knapsack public-key encryption method in the same journal
- Does this mean a general (randomly generated) 0-1 knapsack problem is easy to solve? $\rightarrow$ It was supposed to be NP-complete :(
- *A knapsack problem with a disguised super-increasing vector is not the same as a general knapsack problem with a randomly generated vector*

Lessons from Knapsack Public-Key Encryption

- Adi Shamir's attack on the Merkle-Hellman knapsack public-key encryption method essentially exposes the disguise and finds the randomization parameters t, t^{-1} and p
- This shows the difficulty of using the complexity theory for designing public-key encryption methods
- Public-key cryptography requires trapdoor one-way functions
- The complexity theory identifies computationally intractable problems by reducing them into known problems in a difficult-to-solve set (NP-complete)
- Such problems are inherently difficult for randomly generated inputs
- Disguising easy problems for the purpose of trapdoor does not seem to work well for designing public-key cryptographic algorithms

Well-Known One-Way Functions

- Discrete Logarithm:
  Given p, g, and x, computing y in $y = g^x \pmod{p}$ is EASY
  Given p, g, y, computing x in $y = g^x \pmod{p}$ is HARD
- Factoring:
  Given p and q, computing n in $n = p \cdot q$ is EASY
  Given n, computing p or q in $n = p \cdot q$ is HARD
- Discrete Square Root:
  Given x and y, computing y in $y = x^2 \pmod{n}$ is EASY
  Given y and n, computing x in $y = x^2 \pmod{n}$ is HARD
- Discrete eth Root:
  Given x, n and e, computing y in $y = x^e \pmod{n}$ is EASY
  Given y, n and e, computing x in $y = x^e \pmod{n}$ is HARD