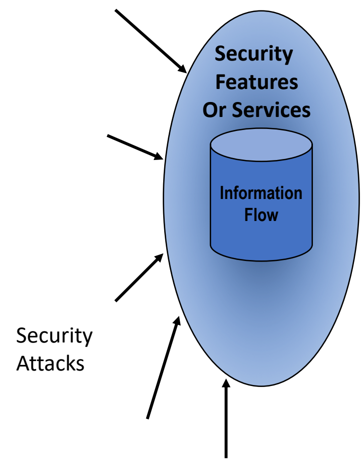
Lecture 8a

Authentication and Hash Functions

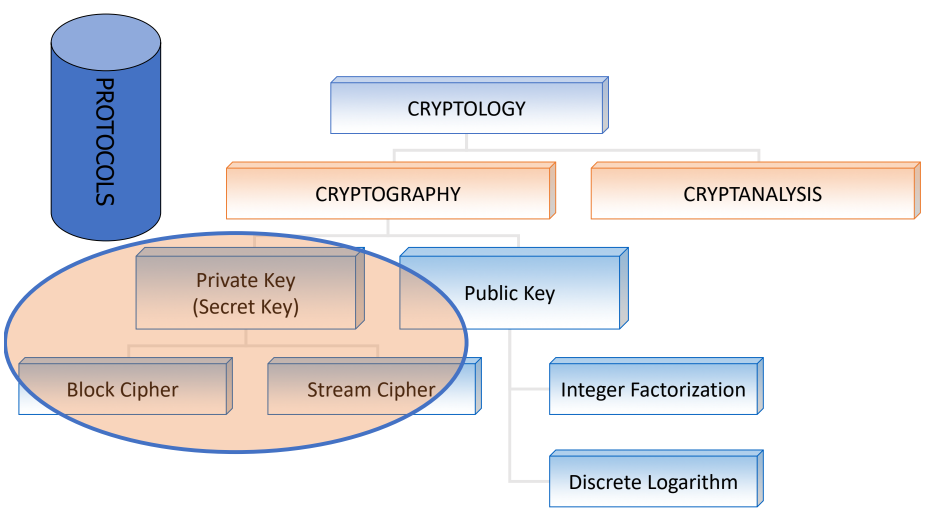
Review



- Confidentiality/Privacy
- Authentication
- Integrity
- Non-repudiation

- Encryption can be used intelligently to provide these services

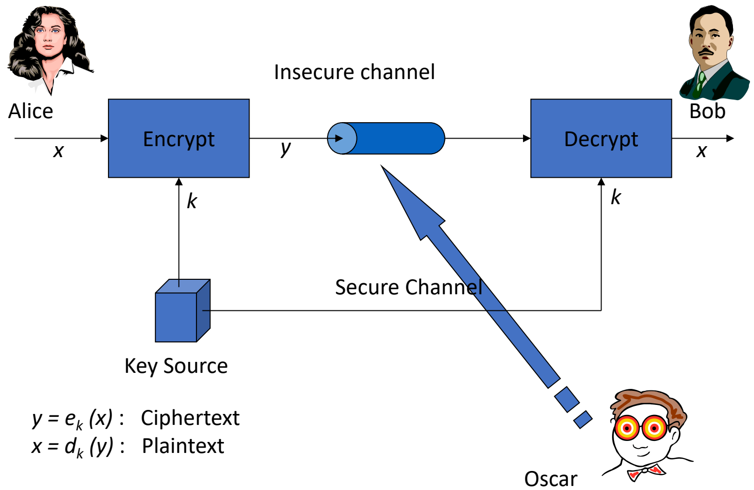
What we have looked at so far



What next?

- Protocols
  - We will not consider "standard" protocols like IPSec or SSL
  - Primitive protocols - "how to?"
- **Message Authentication and Integrity**
- Later
  - Non-repudiation
  - Key management
  - Identification (entity authentication)

Conventional Encryption Model



Confidentiality/Privacy

- Protection of transmitted data from unauthorized access
  - Interception & release of information
  - Clearly, the solution is encryption
    - If the data is encrypted (say using a block cipher in an appropriate mode of operation) the contents are quite secure
  - Traffic analysis
    - Frequency of packets and dependence on time
    - Source and destination networks
    - Much harder to prevent

Traffic confidentiality

- Attack
  - Identification of communicating parties
  - Frequency of communication
  - Message pattern (length, quantity, etc.)
  - Event correlation
- Security measures
  - Link encryption
  - Traffic padding
  - Pad data units to be of fixed size
  - Insert null messages

Security of Secret Key Encryption

- Brute force is the only feasible attack since most block ciphers have no known shortcuts
- Techniques like linear and differential cryptanalysis are almost as complex as brute force
- The key size for good security today is 80 bits
  - 128 bit keys are recommended
  - Mixing may cause poor security...

Message Authentication

Authentication and Integrity

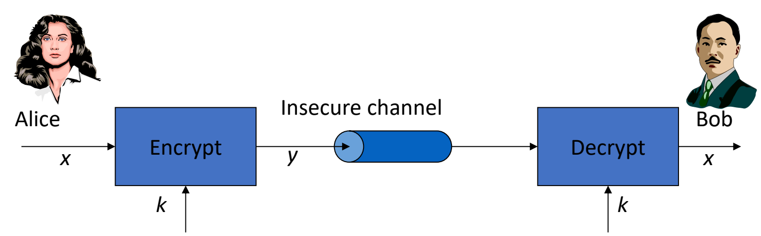
Message Authentication

- Authentication
  - Assurance that a message is coming from an entity that supposedly sent it
  - Protection against masquerade or fraud
- Integrity
  - Assurance that the message has not been modified
    - Contents – insertion, deletion, transposition, etc.
    - Sequence – insertion, deletion, reordering
    - Timing – delay or replay
- Message Authentication = Authentication + Integrity

Authentication

- How do we know whether or not a message is coming from the “claimed” source?
- How do we know that the message has not been modified in between?
- There must be an “authenticator” to verify the authenticity of the message
  - Message encryption
  - Hash functions
  - Message authentication code

Secret key based encryption for message authentication



- Alice and Bob share a key *k*
  - Nobody else is aware of the key *k*
- If a message is received by Bob that can be decrypted using the key *k*, the message MUST have originated at Alice: Yes or No?

Drawbacks of simple encryption

- If the ciphertext *y* can be anything (e.g. a block of 64 bits that look random), Oscar can send spurious or meaningless messages to Bob
- Bob cannot *automatically* say whether Alice sent the meaningless messages
  - Need some structure in the plaintext that can be used to determine spurious messages
  - The structure MUST be secure
- Oscar can "replay" the messages sent by Alice without being detected
  - We look at this later

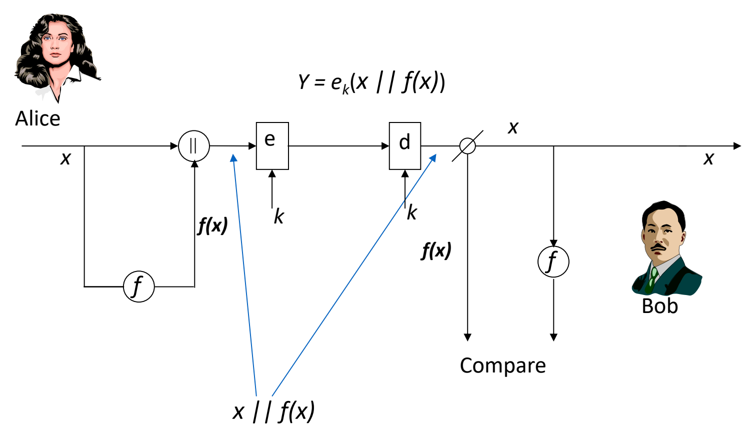
General Idea of using a "function" for Message Authentication

- Generate a function or fingerprint of the message
  - Store it securely if the data is in an insecure place
  - Transmit it securely if the data is transmitted over an insecure channel
- If the data gets altered
  - Hopefully the altered data will NOT have the same fingerprint as the original data
  - If the fingerprint is secure, we can ***detect*** the modification

A simple method for securing the fingerprint

- Append it to the message
- Encrypt the message and the appended function
- A random sequence of bits will not have the properties that the above ciphertext has
- Advantages
  - Using layered communications protocols automatically creates a form of authentication because of the structure

General Idea for Message Authentication



Message Authentication without Privacy

- In some applications, it is only necessary to authenticate but not keep the information secret
  - Broadcast messages and alarm signals
  - Load on receiving side
  - Plaintext messages like shareware etc.
  - SNMPv3 and network management messages
- Since the plaintext is sent without encryption, there is a need to now add a secure authenticator to the message
  - The function auth(x) should be dependent on the message
  - It should not be easily created given the message
  - It should not be easily modified given the message
  - Computational security

How do we generate *auth*(*x*) ?

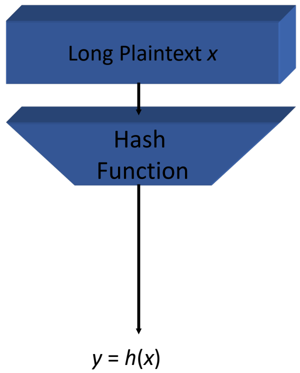
- Use Hash Functions
  - Takes as input a binary string of arbitrary length
  - Creates as output a fingerprint of this string
  - The fingerprint is also called "message digest"
    - Typically a very short string
  - Important in the use of digital signatures
- Use Message Authentication Codes (MAC) or Keyed Hash Functions
  - The hash function is dependent on a shared secret key between Alice and Bob
  - No need for securely keeping the fingerprint
  - Also called an authentication tag

Hash Functions (1)

Hash functions and hash tables

- Hash tables
  - Also a many to *M* mapping (output is one of *M* items)
  - Idea is to be able to quickly check if an element exists in a large set
  - Instead of linear search, a hash table allows the discovery of an element in a set rather quickly
  - Probabilistic – collisions and possibly linear search
- Hash functions for hash tables
  - Modular hashing for integral values in a set
  - Use the remainder when you divide by a number *M*

Hash Functions



- Hash functions create a fixed length "message digest" that is a function of the plaintext
- There is NO key involved
- The hash algorithm is NOT secret
- $h(x)$ can be applied to data of any size
- $h(x)$ produces a fixed length output

Formal Definition

- A hash family is a four-tuple $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$ where
  - $\mathcal{X}$ is a set of possible messages
    - Could be finite or infinite in number
  - $\mathcal{Y}$ is a finite set of possible message digests or authentication tags
  - $\mathcal{K}$ is the keyspace
    - A finite set of possible keys
  - For every possible key $k \in \mathcal{K}$ there exists a hash function $h_k(.) \in \mathcal{H}$ that maps an element from $\mathcal{X}$ to an element in $\mathcal{Y}$

Remarks

- We denote the cardinality of any set S by |S|
  - This is the number of elements in the set S
- For hash functions
  - $|\mathcal{X}| \geq 2|\mathcal{Y}|$ (strong condition)
  - A pair (x, y) is said to be ***valid*** if $y = h_k(x)$ where $x \in \mathcal{X}$ and $y \in \mathcal{Y}$
- Let $\mathcal{F}^{\mathcal{X},\mathcal{Y}}$ denote the set of all functions from $\mathcal{X}$ to $\mathcal{Y}$
  - If $|\mathcal{X}| = N$ and $|\mathcal{Y}| = M$ then $|\mathcal{F}^{\mathcal{X},\mathcal{Y}}| = M^N$
- If $|\mathcal{K}| = 1$, we get an unkeyed hash function

Simple example of an unkeyed Hash Function

- Plaintext consists of blocks of 64 bits $x_1, x_2, x_3, x_4, \ldots$
- The hash function or message digest is the XOR of all the blocks
  - $h(x) = x_1 \oplus x_2 \oplus x_3 \oplus x_4 \ldots$
- Is this secure?
  - What is secure as far as hash functions are concerned?

Requirement (I)

- One-way property or Preimage resistance
  - Suppose the hash value of a plaintext x_0 is $y_0 = h(x_0)$
  - Given $h(.)$ and y_0 it should be computationally infeasible to generate a plaintext x such that $h(x) = y_0$
  - It is easy to generate the hash value but almost impossible to generate the plaintext from a given hash code
  - Why is this important?
- Summary
  - Known: Only y
  - To find: x such that $h(x) = y$

Requirement (II)

- Weak Collision or Second Preimage Resistance
  - Suppose the hash value of a plaintext x_0 is $y_0 = h(x_0)$
  - Given x_0, $h(.)$ and y_0 it should be computationally infeasible to find another message x_1 such that $h(x_1) = y_0$ and $x_1 \neq x_0$
- Note that if it is a feasible problem, then (x_1, y_0) is a valid pair!
- This is a posteriori (the message x_0 is known already)

Requirement (III)

- Strong collision resistance or simply collision resistance
  - It is computationally infeasible to find a pair of plaintexts (x_1, x_2) such that $h(x_1) = h(x_2)$ and $x_1 \neq x_2$
  - This is a priori
    - You are free to pick both x_1 and x_2
- In most cases, we would like the hash function to be collision resistant