Corso di Sicurezza dei Sistemi
Prof. Salvatore D'Antonio

Message Authentication

Message Authentication

- Message authentication is a procedure to verify that received messages come from the alleged source and have not been altered
- Message authentication assures that data received are exactly as sent by (i.e., contain no modification, insertion, deletion, or replay) and that the purported identity of the sender is valid
- Message authentication may also verify sequencing and timeliness
- A digital signature is an authentication technique that also includes measures to counter repudiation by the source

Message authentication functions (1/2)

- Any message authentication mechanism has two levels of functionality
  - At the lower level, there must be some sort of function that produces an authenticator: a value to be used to authenticate a message
  - This lower-level function is then used as a primitive in a higher-level authentication protocol that enables a receiver to verify the authenticity of a message
  - Three classes of functions that may be used to generate an authenticator:
    - Hash function
    - Message encryption
    - Message Authentication Code (MAC)

Message authentication functions (2/2)

- Hash function: A function that maps a message of any length into a fixed length hash value, which serves as the authenticator
- Message encryption: The ciphertext of the entire message serves as its authenticator
- Message authentication code (MAC): A function of the message and a secret key that produces a fixed-length value that serves as the authenticator

Hash function

- A hash function H accepts a variable-length block of data M as input and produces a fixed-size hash value h=H(M)
- A “good” hash function has the property that the results of applying the function to a large set of inputs will produce outputs that are evenly distributed and apparently random
- Hash function and data integrity
  - A change to any bit or bits in M results, with high probability, in a change to the hash code

Cryptographic hash function (1/2)

- A cryptographic hash function is an algorithm for which it is computationally infeasible (because no attack is significantly more efficient than brute force) to find either (a) a data object that maps to a pre-specified hash result (the one-way property) or (b) two data objects that map to the same hash result (the collision-free property)
- Because of these characteristics, hash functions are often used to determine whether or not data has changed

Cryptographic hash function (2/2)

- General operation of a cryptographic hash function
  - The input is padded out to an integer multiple of some fixed length (e.g., 1024 bits), and the padding includes the value of the length of the original message in bits
  - The length field is a security measure to increase the difficulty for an attacker to produce an alternative message with the same hash value
  - The hash value h is h=H(M)

Applications of cryptographic hash functions

- When a hash function is used to provide message authentication, the hash function value is often referred to as a message digest
- Possible ways in which a hash code can be used to provide message authentication
  - The message plus concatenated hash code is encrypted using symmetric encryption
  - Because only A and B share the secret key, the message must have come from A and has not been altered
  - The **hash code** provides the structure or redundancy required to achieve **authentication**. Because **encryption** is applied to the entire message plus hash code, **confidentiality** is also provided

Applications of cryptographic hash functions

- Another method consists in encrypting only the hash code, using symmetric encryption.
  - This reduces the processing burden for those applications that do not require confidentiality
- The third option is to use a hash function but no encryption for message authentication.
  - The technique assumes that the two communicating parties share a common secret value S.
  - A computes the hash value over the concatenation of M and S and appends the resulting hash value to M.
  - Because B possesses S, it can recompute the hash value to verify. Because **the secret value itself is not sent**, an opponent cannot modify an intercepted message and cannot generate a false message

Applications of cryptographic hash functions

- Confidentiality can be added to the approach of the previous method by encrypting the entire message plus the hash code
- There is a growing interest in techniques that **avoid encryption** for the following reasons
  - Encryption software is relatively low
  - Encryption hardware costs are not negligible
  - Encryption hardware is optimized towards large data sizes. For small blocks of data, a high proportion of the time is spent in initialization/invocation overhead.

Encryption

- A message transmitted from source A to destination B is encrypted using a secret key K shared by A and B. If no other party knows the key, then **confidentiality is provided**
- No other party can recover the plaintext of the message
- In addition, **B is assured that the message was generated by A**
- The message must have come from A, because A is the only other party that possesses K and therefore the only other party with the information necessary to construct ciphertext that can be decrypted with K

Encryption

- Furthermore, if M is recovered, B knows that none of the bits of M have been altered
- This is because **an opponent that does not know K would not know how to alter bits in the ciphertext** to produce the desired changes in the plaintext

Message Authentication Code

- Message authentication is achieved using a message authentication code (MAC), also known as a keyed hash function
- Typically, MACs are used between **two parties that share a secret key** to authenticate information exchanged between those parties
- **A MAC function takes as input a secret key and a data block and produces a hash value, referred to as the MAC**
- This can then be transmitted with or stored with the protected message
- If the integrity of the message needs to be checked, the MAC function can be applied to the message and the result compared with the stored MAC value
- **An attacker who alters the message will be unable to alter the MAC value without knowledge of the secret key**
- Note that the verifying party also knows who the sending party is because no one else knows the secret key

Message Authentication Code

- **The combination of hashing and encryption results in an overall function that is, in fact, a MAC** (also said cryptographic checksum)
- T =E(K, H(M)) is a function of a variable-length message M and a secret key K, and it produces a fixed-size output that is secure against an opponent who does not know the secret key
- **T is the fixed-length authenticator, sometimes called a tag**
- The tag is appended to the message at the source at a time when the message is assumed or known to be correct. The receiver authenticates that message by recomputing the tag

Security of MACs

- When an entire message is encrypted for confidentiality, **using either symmetric or asymmetric encryption**, the security of the scheme generally depends on the bit length of the key
- For a k-bit key a brute-force attack using all possible keys will require 2^{k-1} attempts until a decryption result is produced that matches the form of acceptable plaintext
- In the case of a MAC, the considerations are entirely different. In general, the MAC function is a many-to-one function, due to the many-to-one nature of the function

Security of MACs

- If confidentiality is not employed, the opponent has access to plaintext messages and their associated MACs. Suppose $k>n$, that is, suppose that the key size is greater than the MAC size
- Given a known M_1 and T_1 with $T_1=MAC(k , M_1)$ the cryptanalist can perform $T_i=MAC(k_i , M_1)$ for all possible key values k_i
- At least one key is guaranteed to produce a match of $T_i = T_1$
- Note that a total of 2^k tags will be produced, but there are only $2^n < 2^k$ different tag values
- This means that a number of keys will produce the correct tag and the opponent has no way of knowing which is the correct key.
- On average a total of $2^k/ 2^n$ keys will produce a match and therefore the opponent must iterate the attack

HMAC

- Incorporation of a secret key into an existing hash algorithm
- HMAC has been issued as RFC 2104, has been chosen as the mandatory-to-implement MAC for IP security, and is used in other Internet protocols, such as SSL
- HMAC has also been issued as a NIST standard

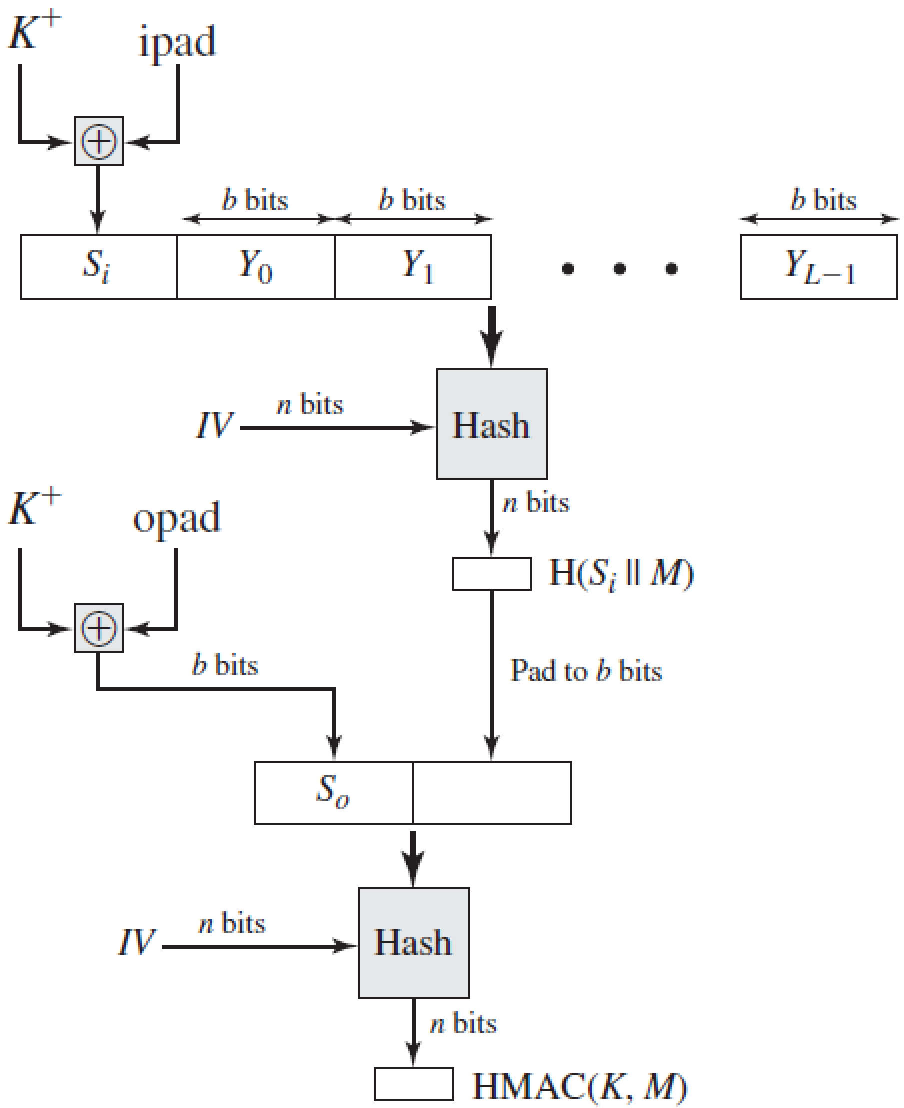
HMAC design objectives (RFC 2104)

- To use, without modifications, available hash functions. In particular, to use hash functions that perform well in software and for which code is freely and widely available.
- To allow for easy replaceability of the embedded hash function in case faster or more secure hash functions are found or required
- To preserve the original performance of the hash function without incurring a significant degradation
- To use and handle keys in a simple way
- To have a well understood cryptographic analysis of the strength of the authentication mechanism based on reasonable assumptions about the embedded hash function

HMAC structure

- H = embedded hash function
- IV = Initial value input to hash function
- M = message input to HMAC (including the padding specified in the embedded hash function)
- L = number of blocks in M
- Y_i = i block of M
- b = number of bits in a block
- n = length of hash code produced by embedded hash function
- K = secret key; recommended length is >= n; if key length is greater than b, the key is input to the hash function to produce an n-bit key
- K^+ = K padded with zeros on the left so that the result is b bits in length
- ipad = 00110110 (36 in hexadecimal) repeated b/8 times
- opad = 01011100 (5C in hexadecimal) repeated b/8 times

HMAC structure



HMAC algorithm

- 1. Append zeros to the left end of K to create a b-bit string (e.g., if K is of length160 bits and b= 512, then will be appended with 44 zeroes).
- 2. XOR (bitwise exclusive-OR) K^+ with ipad to produce the b-bit block S_i.
- 3. Append M to S_i.
- 4. Apply H to the stream generated in step 3.
- 5. XOR K^+ with opad to produce the b-bit block S_0.
- 6. Append the hash result from step 4 to S_0.
- 7. Apply H to the stream generated in step 6 and output the result.