

Message authentication

and secure hashing

Why message authentication

- To prevent against:
 - Masquerade/impersonation
 - Modification of message content
 - Modification of message sequence
 - Acceptance of replayed/delayed messages
 - Source repudiation (denial of origination)
 - Requires signatures/attestation
 - Destination repudiation (denial of receipt)
 - Requires attestation or signature+specific measures

Authentication primitives

- To construct authentication mechanisms, several cryptographic tools can be used:
 - **Keyless primitives:** A *secure hash function* maps any-length messages to a short tag, which can be subsequently authenticated by entering it into a *trusted registry* or by using *keyed primitives*.
 - **Keyed primitives:** *Encryption* (public-key or secret-key), *Message Authentication Codes* (secret-key based) and *Signatures* (public-key based).

Symmetric Encryption

- Symmetric encryption can be used to achieve *both* confidentiality and authentication, via the following heuristic:
 - If modified in transit, the decrypted message “will likely” be garbled to the point where the modification is detected.
- This requires that the message have a particular structure that can be automatically recognized by the receiver.

Examples

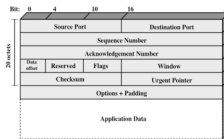


Figure 11.3 TCP Segment

- A TCP packet has a well-defined structure: If it is encrypted within an IP packet, modifications are likely to be detected upon decryption.
- If the authentication is implemented at the application level, even more information about data structure is available to detect modification

Problems with approach

- Encryption schemes are not designed to prevent modification, only to provide confidentiality.
 - For instance, the OFB encryption mode enables arbitrary bit modifications.
 - The resulting security is heuristic, hard to assess, and prone to failure:
 - WEP, Kerberos v.4 PCBC encryption mode, ...

Public key encryption

- Public key encryption by itself provides no authentication (even of the heuristic sort conferred by symmetric-key encryption) because the use of the key does not identify the *message creator*, only *recipient*.
- In interactive protocols, the use of *nonces* can bind *authorship* to *public-key encrypted messages*, resulting in authentication (but not *non-repudiation*).

Hashing

- Hashing is a common computing tool. A hash function f is typically has these properties:
 - Fixed size output:
 - $f(x)$ in $\{0, 1\}^t$, where t is the *hash length*, for any acceptable input x .
 - Larger size input (or arbitrary size input):
 - Input x in $\{0, 1\}^k$, where $k > t$, or x in $\{0, 1\}^*$.
 - Well distributed output:
 - The size of the sets $|F^{-1}(y)| = |\{x : f(x) = y\}|$ does not vary widely with y , i.e., all output values are “approximately equally likely.”

Secure hash

- While the previous definition of hashing is adequate for general purpose computing tasks, such as optimization of data structures/sorting, it is NOT appropriate for secure applications.
- Secure hashing needs to bind the hash output to input.
 - Given the hash value on some input x , it should be difficult to find an alternative input y that has the same hash value.

Hash example (not secure)

- Choose integer n .
- Given binary input x , interpret it as binary expansion of an integer, and let
 - $f(x) = x \bmod n$.
- This results in a fixed-length result for arbitrary length inputs
- The results are often well-distributed
- It does not bind the input.
 - Any y such that the binary representations of x, y satisfy: $x = y \bmod n$, and it is easy to construct other inputs with the same hash value.

Desirable properties of secure hash functions

- A secure hash function is efficiently computable.
- Accepts very long inputs, produces fixed-length (short) output.
- *One-way*:
 - Given y , finding x such that $H(x) = y$ is hard.
- *Second pre-image* or *weak collision resistant*:
 - Given x , finding $y \neq x$ such that $H(x) = H(y)$ is hard.
- *(Strong) collision-resistant*:
 - Find any pair (x, y) , $x \neq y$ such that $H(x) = H(y)$ is hard.

Hash-and-sign

- Public key signature schemes can be used to authenticate messages. But there are two problems with that:
 - The computation of signatures is expensive, and prohibitively so for long messages.
 - Without use of redundancy in the messages, signatures are malleable, and can be modified by other parties.
- By first hashing a message, and then signing the hash, both problems are solved.

Example: RSA hash-and-sign

- RSA public key (n, e) and private key d .
 - Sign message m as $m^d \bmod n$.
- Attacker malleability attack:
 - Change m^d to $c \cdot m^d \bmod n$, signing $c^e \cdot m \bmod n$.
- Hash-and-sign:
 - Send $(m, s = H(m)^d \bmod n)$.
- Malleability attack fails:
 - Changing m to m' and s to s' requires that $s'^e = H(m')$. This is hard to do.
