Computer Security

Message Authentication Codes,
Hashes and Message Digests

What Problems are We Trying to Solve?

- ***Authentication***
  - The process of reliably verifying the identity of someone or something
    - In particular, did a message indeed come from its specified sender?
- Need to guard against:
  - Disclosure of a message to any unauthorized person or system
  - Messages masquerading as being from a source
  - Content modification
    - Was the message changed between sender and recipient?)
  - Source or destination repudiation
    - Can the sender or recipient deny they sent/received the message?

Authentication Functions

- Three main classes:
  - ***Message Encryption***
    - Ciphertext of entire message serves as its authenticator
  - ***Message Authentication Code (MAC)***
    - Function of the message and a secret key which produces a fixed-length authenticator value
  - ***Hash Function***
    - Function which maps a message of any length into a fixed-size hash authenticator value

Message Encryption

- We've already seen various encryption options:
  - ***Symmetric Encryption*** – provides both *confidentiality* and *authentication*
  - ***Public Key Encryption***:

$C = E_{KU_b}(P)$; $P = D_{KR_b}(C)$ ➢ provides *confidentiality* only

$C = E_{KR_b}(P)$; $P = D_{KU_b}(C)$ ➢ provides *authentication* & *signature*

$C = E_{KU_a}(E_{KR_b}(P))$
$P = D_{KR_a}(D_{KU_b}(C))$
➢ provides *confidentiality*, *authentication*, and *signature*

Message Authentication Code (MAC)

- Uses a secret key to generate a small fixed-size block of data: a ***Cryptographic Checksum***, or ***MAC***

- If Alice and Bob share a common secret key, K, then:

  $MAC = F_K(P)$

  where:

  P is the input plaintext

  F is the MAC function

  K is the shared secret key

  MAC is the resulting message authentication code

Message Authentication Code (MAC)

- The MAC is sent to the recipient along with its associated message
  - The recipient uses the same MAC function and key on the message and compares the result with the transmitted MAC.
  - If the two MACs match, the recipient is assured that:
    - The message has not been altered, and
    - The message is from the alleged sender
  - If the message contains a sequence number (e.g. TCP header), then the recipient is assured of the proper sequencing of messages

Message Authentication Code (MAC)

- A MAC function is similar to an encryption function, except that the MAC function ***need not be reversible***.
- A MAC function is typically a ***many-to-one function***:
  - The input messages can be of any length, but
  - The output is of fixed length (n bits)
- If an n-bit MAC function (that is, a MAC function that generates a fixed-size n-bit block) is used, how many possible MAC values are there?

$$2^n$$

Message Authentication Code (MAC)

- For example, assume:
  - We are using a 10-bit MAC function
  - We are using 100-bit messages

- There are:
  - 2^{100} (~ $1.27 * 10^{30}$) different possible messages:
  - only 2^{10} (1024) different possible MACs

  so, on average, each MAC value is generated by a total of

  $$\frac{2^{100}}{2^{10}} = 2^{90} \approx 1.24 \times 10^{27}$$

  different messages

Message Authentication Code (MAC)

- Here are some basic uses of MACs:

$M = P \,||\, F_K(P)$

- Plaintext concatenated with MAC
- Provides ***authentication*** only

P = Plaintext
C = Ciphertext
E = Encryption function
F = MAC function
K = Key
M = Message sent

$M = E_{K_2}(P \,||\, F_{K_1}(P))$

- Symmetric key encrypted ciphertext concatenated with plaintext MAC
- Provides ***authentication*** and ***confidentiality***
- Authentication tied to plaintext

$C = E_{K_2}(P)\ ;\ \ M = C \,||\, F_{K_1}(C))$

- Symmetric key encrypted ciphertext concatenated with ciphertext MAC
- Provides ***authentication*** and ***confidentiality***
- Authentication tied to ciphertext

Message Authentication Code (MAC)

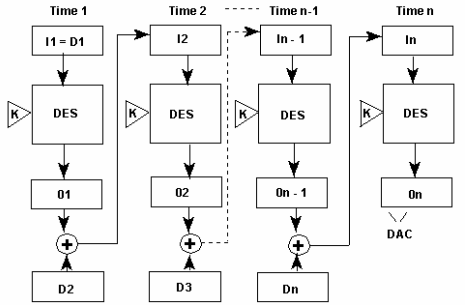
- Why use MACs for authentication, instead of symmetric encryption?
  - Performance – MAC functions are faster than encryption functions
  - Some applications don't care about secrecy/confidentiality, but care a lot about authentication (e.g. SNMP)
  - Separation of authentication and confidentiality provides greater flexibility.
  - Certain applications may wish to prolong the period of authentication so that a message can undergo several phases of processing which rely on that authentication.
    - As soon as the message is decrypted, the message is no longer protected against modification
    - A MAC can be kept with the message to retain this protection.

Message Authentication Code (MAC)

- One of the most widely used MACs is the ***Data Authentication Algorithm (DAA)***, based on on DES
  - DAA is both a FIPS publication (FIPS PUB 113*) and an ANSI standard (X9.17)
  - It uses the Cipher Block Chaining (CBC) mode of operation with an initialization vector of zero.
  - The data are grouped into contiguous 64-bit blocks; if necessary, the final block is padded on the right with zeroes to form a full 64-bit block.
  - Using the DES encryption algorithm, E, and a secret key, K, a ***Data Authentication Code (DAC)*** is calculated...

*http://www.itl.nist.gov/fipspubs/fip113.htm

Digital Authentication Algorithm



One-Way Hash Functions

- Can be viewed as a variation on a MAC function:
  - A ***Hash Function*** accepts a variable-size message, *M*, as input and produces a fixed-size output, referred to as a ***Hash Code***, or ***Message Digest***:

    $$h = H(M)$$

  - Unlike a MAC function, a hash code:
    - ***does not use a key***, and so
    - ***is a function only of the input message***
  - A change to any bit (or bits) of the message results in a change to the hash code, which can provide an ***error-detection capability***.
  - A message digest can be used as a ***fingerprint*** for a message, to allow detection of message modification

One-Way Hash Functions

- Requirements for a hash function are:
  - Can be applied to a block of data of any size
  - Produces a fixed-length output
  - $H(M)$ is relatively easy to compute for any given M, allowing for both software and hardware implementations
  - For any given value h, it is computationally infeasible to find M such that $h = H(M)$. This is the ***One-way Property***.
  - For any given block, M, it is computationally infeasible to find M' != M with $H(M') = H(M)$.
    - This is called ***Weak Collision Resistance***.
  - It is computationally infeasible to find any pair (M, M') such that $H(M) = H(M')$.
    - This is called ***Strong Collision Resistance***.

One-Way Hash Functions

- Here are some basic uses of hash functions:

$M = E_K(P \,||\, H(P))$
- Provides confidentiality and authentication

$M = P \,||\, E_K(H(P))$
- Provides authentication (amounts to a MAC)

$M = P \,||\, E_{KR_a}(H(P))$
- Provides authentication
- Also provides a digital signature (because only the sender could have produced the encrypted hash code)

One-Way Hash Functions

- ...and some more:

$M = E_K(P \,||\, E_{KR_a} H(P))$
- Provides confidentiality, authentication and digital signature

$M = P \,||\, H(P \,||\, S))$
- Provides authentication, using a shared secret, S

$M = E_K(P \,||\, H(P \,||\, S))$
- Adds confidentiality to the above mechanism.

One-Way Hash Functions

- The drive for hash/message digest algorithms began with public key cryptography
  - RSA was invented, but it was slow enough at that time to make it impractical when used alone.
  - A cryptographically secure message digest algorithm with high performance would make RSA much more useful.

One-Way Hash Functions

- Thus, ***MD*** (unpublished) and ***MD2*** (defined in RFC 1319[1]) were invented by Ron Rivest (of RSA fame)
  - MD2 is a 128-bit one-way hash function
- Ralph Merkle (of Xerox at the time, but who had worked at Stanford with Diffie & Hellman) came up with a message digest algorithm called ***SNEFRU***[2]
  - It was several times faster than MD2
  - It is a 128-bit or 256-bit one-way hash function
- This prompted Rivest to respond with ***MD4*** (defined in RFC 1320[3])
  - MD4 produces a 128-bit hash

[1] http://www.faqs.org/rfcs/rfc1319.html
[2] Snefru was an Egyptian Pharaoh
[3] http://www.faqs.org/rfcs/rfc1320.html

One-Way Hash Functions

- Weaknesses were found in the MD4 algorithm, so Rivest came up with ***MD5*** (defined in RFC 1321[1])
  - MD5 is similar to MD4, but improved.
  - More complex than MD4, but similar in design
  - Also produces a 128-bit hash

[1] http://www.faqs.org/rfcs/rfc1321.html

One-Way Hash Functions

- The NSA designed the ***Secure Hash Algorithm (SHA)***
- The National Institute of Standards and Technology (NIST), made it a standard.
  - They revised it very late in the game, because of some (unspecified) weakness that had been found, and changed its name to ***SHA-1***[1].
  - SHA-1 is a 160-bit hash function based on MD4
  - Shares much in common with MD5, but has a much more conservative design.
  - 2 or 3 times slower than MD5

[1] http://www.itl.nist.gov/fipspubs/fip180-1.htm

One-Way Hash Functions

- Because of improving computing environments, the 160-bit hash of SHA-1 is now becoming a liability
- So NIST produced ***SHA-256***, ***SHA-384***, and ***SHA-512***[1].
  - Designed to be used with the 128-, 192- and 256-bit key sizes of AES.
  - Structure similar to SHA-1, but much slower.
  - According to Ferguson & Schneier, they are too new to have experienced significant cryptanalysis attacks, and so are not yet proven. But they also say that MD5 and SHA-1 are not sufficiently secure, so you may have little choice.

[1] http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf

Typical Usage of Digests

- Here's an example of the use of SHA-1 in a real application – Java JAR files:
  - In the **`jce.jar`** (Java Cryptography Extension) JAR file, the ***manifest*** contains the following:

```
Manifest-Version: 1.0
Created-By: 1.4.1-internal (Sun Microsystems Inc.)

Name: javax/crypto/SealedObject.class
SHA1-Digest: R+GWl6Zuqgtty1zOaP5RrRSGfQo=

Name: javax/crypto/KeyAgreementSpi.class
SHA1-Digest: fdmlqpiTKMzV65+93O4tJ3Uo6wg=

Name: javax/crypto/spec/DESedeKeySpec.class
SHA1-Digest: Q7UJvLuk8GST42GW6xDlXHe3Xv8=

Name: javax/crypto/spec/DHParameterSpec.class
SHA1-Digest: y0oY9yd/BQQxEc/2q1Cytta/r2E=

Name: javax/crypto/interfaces/DHPrivateKey.class
SHA1-Digest: jwgw7pakTyKOlLBNivsp6V6Ad4k=

...
```


One-Way Hash Functions

- Ferguson & Schneier[1] recommend the following:
  - They don't feel that any of the existing hash functions is sufficiently secure when used as is.
  - So, they define a double hash function, $h_d = h(h(m))$
  - They recommend the use of SHA$_d$-256, or, for higher security levels, SHA$_d$-512

[1] ***Practical Cryptography***, by Niels Ferguson and Bruce Schneier, Wiley.

HMAC

- HMAC[1] is an attempt to use a hash function to build a MAC.
- Note that:
  - Hash functions provide only $n/2$ bits of security against some attacks
  - A MAC function is expected to provide n bits of security.
  - Defining:

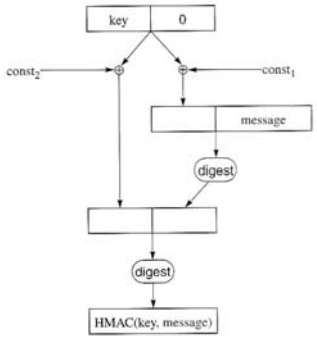
    $MAC(K, m) = h(K \| m)$, or $h(m \| K)$, or $h(K \| m \| K)$

    is not secure if you use one of the standard hash functions.
- HMAC was proven to be secure if the underlying hash function's underlying compression function was secure.

[1] http://www.faqs.org/rfcs/rfc2104.html
There is also a draft FIPS at:
http://csrc.nist.gov/CryptoToolkit/hmac/fr-hmac-200101.html

HMAC

- HMAC takes a variable length key and a variable length message, and produces a fixed-size output that is the same size as the underlying hash function.
  - It first pads the key with 0 bits to 512 bits
  - If the key is larger than 512 bits, it first hashes the key, resulting in 128 or 160 bits, and then pads the result out to 512 bits.
  - It then XORs the padded key with a constant string of octets of value 36 (hex), then concatenates the result with the message, and computes a hash (digest)
  - It XORs the padded key with a different constant string of octets of value 5C (hex), concatenates that with the result of the of the first hash, and computes a second hash on that.



HMAC

- Examples of HMAC algorithms are:
  - HmacMD5 – uses the MD5 hash algorithm
  - HmacSHA1 – uses the SHA-1 hash algorithm

Summary

- We've covered:
  - A little about authentication
  - Somewhat more about Message Authentication Codes (MACs)
  - Even more about One-Way Hash Functions (also known as Message Digests or Message Digest Functions)
- We have also seen how they are all related to previous cryptography topics