UNIT IV

MESSAGE AUTHENTICATION

Authentication Systems – Password and Address – Security Handshake Drawbacks - Authentication Standards – Kerberos- PKI Trust Models -Message Authentication Codes (MAC)– Security features- MAC based on Hash Functions - MAC based on Block Ciphers

4 Authentication Systems

4.0.1. USER AUTHENTICATION

In most computer security contexts, user authentication is the fundamental building block and the primary line of defense. User authentication is the basis for most types of access control and for user accountability. RFC 2828 defines user authentication as the process of verifying an identity claimed by or for a system entity. An authentication process consists of two steps:

- ✓ **Identification step:** Presenting an identifier to the security system. (Identifiers should be assigned carefully, because authenticated identities are the basis for other security services, such as access control service.)
- ✓ **Verification step:** Presenting or generating authentication information that corroborates the binding between the entity and the identifier."

In essence, identification is the means by which a user provides a claimed identity to the system; user authentication is the means of establishing the validity of the claim. Note that user authentication is distinct from message authentication.

Means of User Authentication

There are four general means of authenticating a user's identity, which can be used alone or in combination:

- ✓ **Something the individual knows:** Examples includes a password, a personal identification number (PIN), or answers to a prearranged set of questions.
- ✓ **Something the individual possesses:** Examples include electronic keycards, smart cards, and physical keys. This type of authenticator is referred to as a *token*.

- ✓ **Something the individual is (static biometrics):** Examples include recognition by fingerprint, retina, and face.
- ✓ **Something the individual does (dynamic biometrics):** Examples include recognition by voice pattern, handwriting characteristics, and typing rhythm.

All of these methods, properly implemented and used, can provide secure user authentication. However, each method has problems. An adversary may be able to guess or steal a password. Similarly, an adversary may be able to forge or steal a token. A user may forget a password or lose a token. Further, there is a significant administrative overhead for managing password and token information on systems and securing such information on systems. With respect to biometric authenticators, there are a variety of problems, including dealing with false positives and false negatives, user acceptance, cost, and convenience.

4.0.2. MUTUAL AUTHENTICATION PROTOCOLS

- ✓ used to convince parties of each other's identity and to exchange session keys
- ✓ may be one-way or mutual
- ✓ key issues are
    - confidentiality – to protect session keys
    - timeliness – to prevent replay attacks

Replay Attacks

Replay Attacks are where a valid signed message is copied and later resent. Such replays, at worst, could allow an opponent to compromise a session key or successfully impersonate another party. At minimum, a successful replay can disrupt operations by presenting parties with messages that appear genuine but are not. [GONG93] lists the examples above of replay attacks.

Possible countermeasures include the use of:

- ✓ sequence numbers (generally impractical since must remember last number used with every communicating party)
- ✓ timestamps (needs synchronized clocks amongst all parties involved, which can be problematic)

- ✓ challenge/response (using unique, random, unpredictable nonce, but not suitable for connectionless applications because of handshake overhead)

One-Way Authentication

- ✓ required when sender & receiver are not in communications at same time (eg. email)
- ✓ have header in clear so can be delivered by email system
- ✓ may want contents of body protected & sender authenticated

4.0.3. REMOTE USER – AUTHENTICATION USING SYMMETRIC ENCRYPTION

A two-level hierarchy of symmetric encryption keys can be used to provide confidentiality for communication in a distributed environment.

Usually involves the use of a trusted key distribution center (KDC). Each party in the network shares a secret master key with the KDC. The KDC is responsible for generating session keys, and for distributing those keys to the parties involved, using the master keys to protect these session keys.

Password Authentication

Authentication is the process that ensures the individual requesting access to a system, website, or application is the intended user. Password authentication is a process that involves a user inputting a unique ID and key that are then checked against stored credentials. There are three main methods used for authentication purposes:

- **Knowledge-based:** Also referred to as "something you know." This category includes traditional passwords. When a user creates a unique password for their account, it then becomes the key to re-enter that account time and time again. It's something that the user (and hopefully *only* the user) knows.
- **Possession-based:** Also referred to as "something you have." In this scenario, an individual verifies their identity using something that only the intended user would have. For example, a user could swipe a physical key card or prove access to a separate personal email account to show that they are who they say they are.
- **Inheritance-based:** Also referred to as "something you are." These are typically biometric characteristics such as a fingerprint or facial scan that is then used to verify the user's identity.

4.1.4 Password Storage

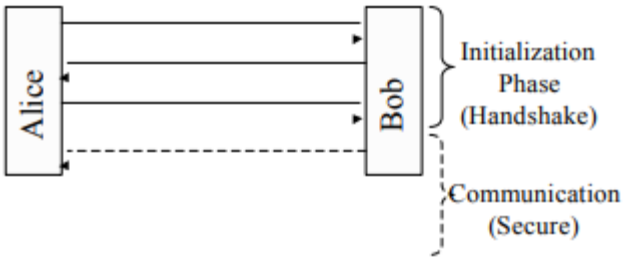
When a user creates a password, a copy of that credential is stored by the system or website in a secure password database against which the server can compare any further login attempts.

Because all those passwords are stored in a centralized location, (which would be a field day for hackers) it's important that password-based authentication systems ensure top-notch security for those databases. Typically, passwords are stored in an encrypted fashion so that even if a hacker is able to access the database, the information they see would be of no use to them. This is called salting and hashing the passwords

Security Handshake Pitfalls

Introduction

During the handshake phase communication parameters are negotiated and initial information are exchanged. Some of these information are secret (e.g. the password), some are not (e.g. the user names).



To cope with different types of threats individual protocols have different strengths and weaknesses.

- Some threats are more likely in some situations.
- Availability of resources may differ

- Computational power
- Specialized hardware

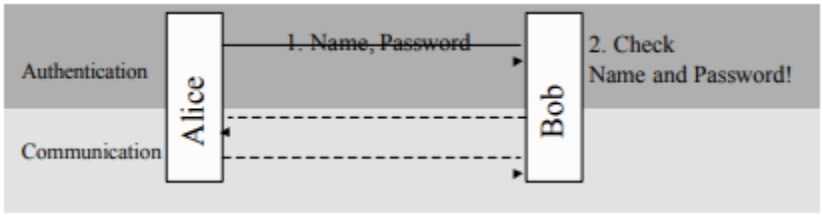
- Humans and computers may behave differently.
- Protocols themselves may be flawed.

Login Only

Many protocols were designed for environments where eavesdropping was not a concern. Authentication in such protocols consists of:

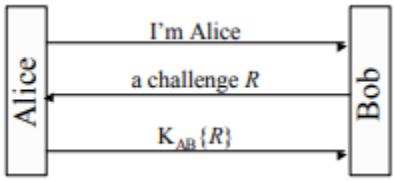
1. Alice send her name and password to Bob.

2. Bob verifies the name and password, and then communication commences, without any further attention to security.



A very common enhancement to such a protocol is to replace the transmission of the clear text password with a cryptographic challenge/response.

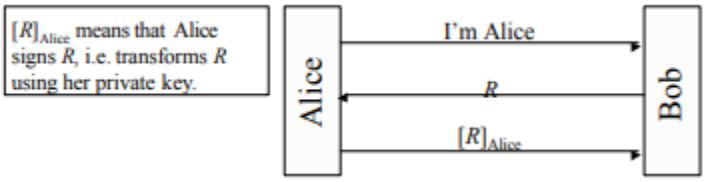
Login Only / Shared Secret



This would be a big improvement over cleartext passwords. An eavesdropper cannot impersonate Alice based on overhearing the exchange, since next time there will be a different challenge. However, there are some weaknesses to this protocol:

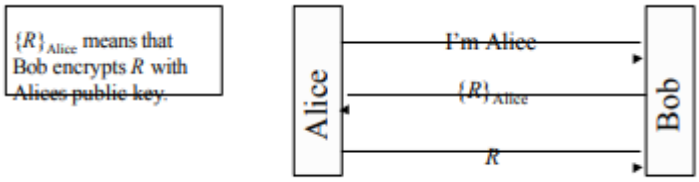
- Authentication is not mutual.
- If this is the entire protocol, then Trudy can hijack the conversation after the initial exchange.
- An eavesdropper could mount an off-line password-guessing attack.
- Someone who has access to Bob's database can impersonate Alice.

Login Only / One-Way Public Key



The above protocol is based on a public key and similar to the first protocol. Bob verifies Alice's signature [R]. Alice using her public key, and accepts the login if the result matches R. The advantage of this protocol:

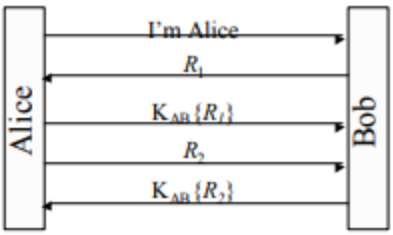
- Reading Bob's database at is no longer a potential security-threat, but it must be protected from unauthorized modification.
- If you can impersonate Bob's network address you can trick Alice into signing something (wait for Alice to try log in and then give her your quantity).



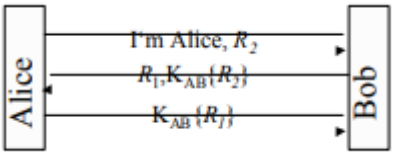
Properties of this protocol:

- Requires a reversible public key algorithm.
- If you can impersonate Bob's network address you can trick Alice into decrypting something.
  Solution:
- A message should have a structure so that it cannot be mistaken for another type.

Mutual Authentication



Mutual authentication means that both communication partners are able to identify each other. An example for a simple protocol for mutual authentication is based on shared secrets. The protocol is inefficient as it needs 5 messages.This can be prevented by reducing the number of messages to three by putting more than one item of information into each message.



This version of the protocol has a security pitfall known as reflection attack.

4.2 Kerberos

Kerberos is an authentication service developed as part of Project Athena at MIT, and is one of the best known and most widely implemented **trusted third party** key distribution systems.

Kerberos provides a centralized authentication server whose function is to authenticate users to servers and servers to users. Unlike most other authentication schemes, Kerberos relies

exclusively on symmetric encryption, making no use of public-key encryption. Two versions of Kerberos are in common use: v4 & v5.

4.0.1. KERBEROS REQUIREMENTS

- ✓ its first report identified requirements as:
    - secure
    - reliable
    - transparent
    - scalable
- ✓ implemented using an authentication protocol based on Needham-Schroeder

4.0.2. KERBEROS V4 OVERVIEW

- ✓ a basic third-party authentication scheme
- ✓ have an Authentication Server (AS)
    - users initially negotiate with AS to identify self
    - AS provides a non-corruptible authentication credential (ticket granting ticket TGT)
- ✓ have a Ticket Granting server (TGS)
    - users subsequently request access to other services from TGS on basis of users TGT
- ✓ using a complex protocol using DES

KerberosVersion4

(1) C -> AS $ID_C \| ID_{tgs} \| TS_1$

(2) AS -> C $E(Kc, [K_{c,\ tgs} \| ID_{tgs} \| TS_2 \| Lifetime_2 \| Ticket_{tgs}$ D

$$Ticket_{tgs} = E\ (K_{tgs}, [Kc_{,tgs} \| ID_C \| AD_C \| ID_{tgs} \| TS2 \| Lifetime_2])$$

(a) Authentication service exchange to obtain ticket-granting ticket

(3) C ->TGS $ID_V \| Ticket_{tgs} \| Authenticator_C$

(4) TGS -> C $E\ (K_{C,\ tgs}, [K_{C,\ V} \| ID_V \| TS_4 \| Ticket_V])$

$Ticket_{tgs} = E (K_{tgs}, [K_{C, tgs} \| ID_C \| AD_C \| ID_{tgs} \| TS_2 \| Lifetime_2])$
$Ticket = E (K_V, [K_{C, V} \| ID_C \| AD_C \| ID_V \| TS_4 \| Lifetime_4])$
$authenticator = E (K_{C, tgs}, [ID_C \| AD_C \| TS_3])$

(b) Ticket granting service exchange to obtain service granting ticket

(5) C -> V $\| Ticket_V \| Authenticator_C$
(6) V -> C $E (K_{C, V}, [TS_5 + 1]$ (for mutual authentication)
$Ticket_V = E (K_V, [K_{C, V} \| ID_C \| AD_C \| ID_V \| TS_4 \| Lifetime_4])$
$Authenticator_C = E (K_{C,V}, [ID_C \| AD_C \| TS_5])$

(c) Client/server authentication exchange to obtain service

The full Kerberos v4 authentication dialogue is shown here, divided into 3 phases.

First, consider the problem of captured ticket-granting tickets and the need to determine that the ticket presenter is the same as the client for whom the ticket was issued. An efficient way of doing this is to use a session encryption key to secure information. The above table shows the technique for distributing the session key. Note that several additional pieces of information have been added to this first phase of the dialogue. Message (1) includes a timestamp; so that the AS knows that the message is timely. Message (2) includes several elements of the ticket in a form accessible to C. This enables C to confirm that this ticket is for the TGS and to learn its expiration time.

Note that the ticket does not prove anyone's identity but is a way to distribute keys securely. It is the authenticator that proves the client's identity. Because the authenticator can be used only once and has a short lifetime, the threat of an opponent stealing both the ticket and the authenticator for presentation later is countered. C then sends the TGS a message that includes the ticket plus the ID of the requested service (message 3). The reply from the TGS, in message (4), follows the form of message (2). C now has a reusable service-granting ticket for V. When C presents this ticket, as shown in message (5), it also sends an authenticator. The server can

decrypt the ticket, recover the session key, and decrypt the authenticator. If mutual authentication is required, the server can reply as shown in message (6)

4.0.2.1. Kerberos Realms and Multiple Kerberi

Kerberos may be represented as follows:

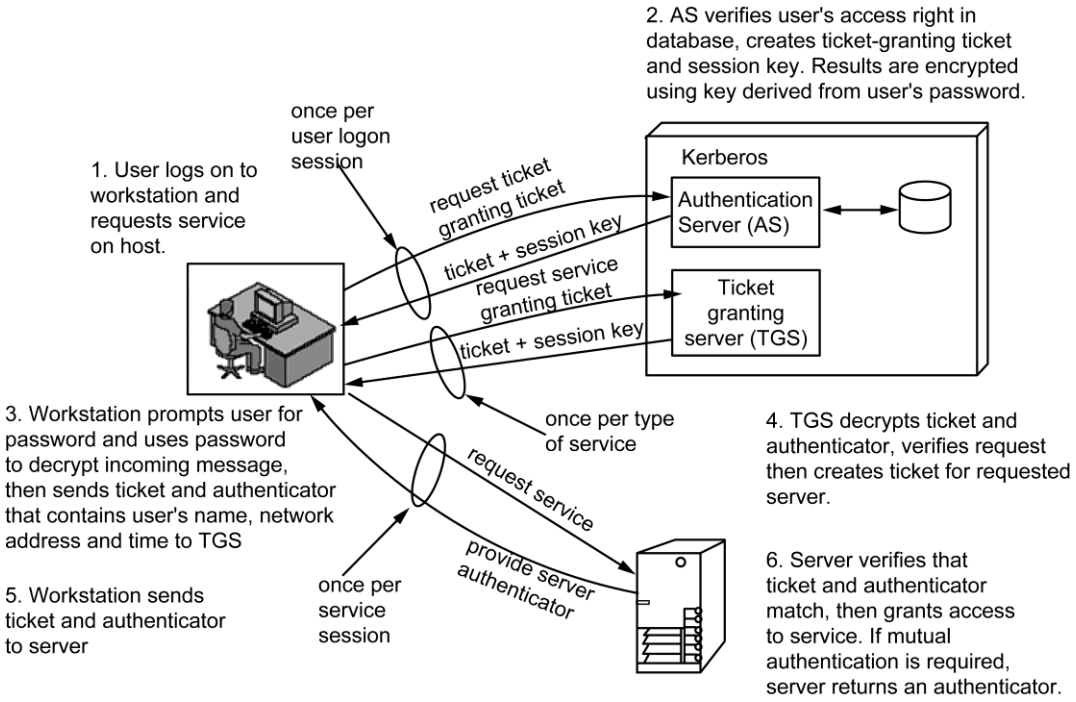


Fig. 4.1. Kerberos

A full-service Kerberos environment consisting of a Kerberos server, a number of clients, and a number of application servers requires the following:

1. The Kerberos server must have the user ID (UID) and hashed password of all participating users in its database. All users are registered with the Kerberos server.
2. The Kerberos server must share a secret key with each server. All servers are registered with the Kerberos server.

Such an environment is referred to as a **realm**. Networks of clients and servers under different administrative organizations typically constitute different realms. That is, it generally is not practical, or does not conform to administrative policy, to have users and servers in one administrative domain registered with a Kerberos server elsewhere. However, user in one real may need access to servers in other realms, and some servers may be willing to provide service

to users from other realms, provided that those users are authenticated.

Kerberos provides a mechanism for supporting such inter-realm authentication. For two realms to support inter-realm authentication, a third requirement is added:

3. The Kerberos server in each interoperating realm shares a secret key with the server in the other realm. The two Kerberos servers are registered with each other.

The scheme requires that the Kerberos server in one realm trust the Kerberos server in the other realm to authenticate its users. Furthermore, the participating servers in the second realm must also be willing to trust the Kerberos server in the first realm.

We can describe the mechanism as follows:

A user wishing service on a server in another realm needs a ticket for that server. The details of the exchange illustrated in Figure are as follows:

1. C -> AS: IDc || Idtgs || TS1
2. AS -> C: E_{Kc}[Kc,tgs || Idtgs || TS2 || Lifetime2 || $Ticket_{tgs}$]
3. C -> TGS: Idtgsrem || $Ticket_{tgs}$ | |$Authenticator_c$
4. TGS -> C: $E_{Kc,tgs}$[Kc,tgsrem || Idtgsrem || TS4 || Lifetime2 || $Ticket_{tgsrem}$]
5. C -> TGS_{rem}: Idvrem || $Ticket_{tgsrem}$ || $Authenticator_c$
6. TGS_{rem} -> C: $E_{Kc,tgsrem}$[Kc,vrem || Idvrem || TS6 || $Ticket_{vrem}$]
7. C -> Vrem: $Ticket_{vrem}$ || $Authenticator_c$

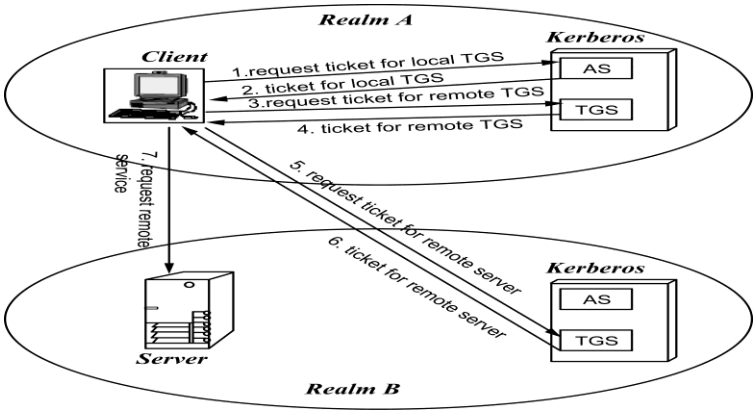


Fig. 4.2. Request for service in Another Realm

The ticket presented to the remote server (Vrem) indicates the realm in which the user was originally authenticated. The server chooses whether to honor the remote request.

One problem presented by the foregoing approach is that it does not scale well to many realms. If there are *N* realms, then there must be *N(N – 1)/2* secure key exchanges so that each Kerberos realm can interoperate with all other Kerberos realms.

4.3 Message Authentication Code

It is an alternative authentication technique that involves the use of a secret key to generate a small fixed size block of data known as a cryptographic Check sum or MAC. This MAC is appended with the message. Assume that two communicating parties, say A and B share a common secret key K. When say wishes a message to dens to B, it calculates the MAC function of the message and the key.

MAC = MAC(K,M) where

M = input message

C = MAC function

K = shared secret key

MAC = Message Authentication Code.

The Recipient on receiving the message performs the same calculation on the message using the same secret key to generate the new MAC. The received MAC and the Computed MAC is then compared. Fig 3.3 shows about the MAC authentication function.

As shown figure.3.3(*a*), the MAC provides authentication but does not provide any confidentiality to the message as it is seen clear. In order to provide confidentiality encryption can be used to the message that is being appended.

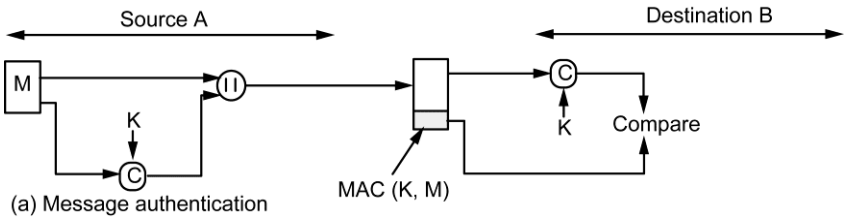


Fig. 3.3. (a) Message Authentication

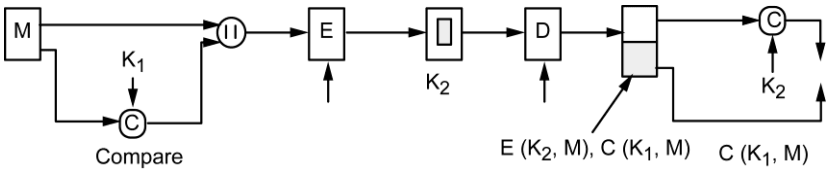


Fig. 3.3. (b) Message Authentication and Confidentiality

As shown in above figure both authentication and confidentiality is achieved by encrypting the plaintext, assuming that both the sender and the receiver use the shared secret key k1 for both encryption and decryption.

4.0.1. MAC PROPERTIES

1. A MAC is a cryptographic checksum

MAC = $C_K(M)$ which condenses a variable-length message M and use a secret key K. The MAC 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 re-computing the MAC.

2. MAC is a many-to-one function

- potentially many messages have same MAC
- but finding these needs to be very difficult

4.0.2. REQUIREMENTS FOR MACS

- ✓ The first requirement deals with message replacement attacks, in which an opponent is able to construct a new message to match a given MAC, even though the opponent does not know and does not learn the key.
- ✓ The second requirement deals with the need to thwart a brute-force attack based on chosen plaintext.

- ✓ The final requirement dictates that the authentication algorithm should not be weaker with respect to certain parts or bits of the message than others.

Need the MAC to satisfy the following:

1. knowing a message and MAC, is infeasible to find another message with same MAC
2. MACs should be uniformly distributed
3. MAC should depend equally on all bits of the message

4.0.3. SECURITY OF HASH FUNCTION AND MACS

Requirements and Security

For a hash value $h = H(x)$, we say that x is the preimage of h. That is x is a data block whose hash function is h. A collision occurs if we have $x \neq y$ and H(x) = H(y), because hash function is used to afford data integrity.

Properties of Hash Function:

1. Hash function H can be applied to a block of data of any variable size.
2. Hash function H produces fixed length output.
3. H(x) is relatively easy to compute for any given x. Hence both hardware and software implementation becomes easy and it improves efficiency.
4. The fourth property is preimage resistant i.e one way property.

One way property: It is easy to generate a message given a code but highly infeasible to generate a message given a code.

5. The fifth property is second preimage resistant, guarantees that it is infeasible to find another message with the same hash value for a given message.
6. The sixth property is collision resistant which is known as collision resistant. A strong hash function protects against an attack in which one party generates a message for another party to sign.

Note: The hash function that satisfies the first five properties are called as weak hash functions.

Just as with symmetric and public-key encryption, we can group attacks on hash functions and MACs into two categories:

1. Brute-force attacks
2. Cryptanalysis.

Brute-force attacks:

1. A brute-force attack on a MAC is a more difficult undertaking than a brute-force attack on a hash function because it requires known message-tag pairs.
2. The strength of a hash function against brute-force attacks depends solely on the length of the hash code produced by the algorithm, with cost O(2^m /2).
3. A brute-force attack on a MAC has cost related to min(2^k , 2^n), similar to symmetric encryption algorithms. It would appear reasonable to require that the key length and MAC length satisfy a relationship such as min(*k*, *n)* >= *N*, where N is perhaps in the range of 128 bits.

Cryptanalytic attacks:

1. Cryptanalytic attacks on hash functions and MAC algorithms seek to exploit some property of the algorithm to perform some attack other than an exhaustive search.
2. The way to measure the resistance of a hash or MAC algorithm to cryptanalysis is to compare its strength to the effort required for a brute-force attack. That is, an ideal hash or MAC algorithm will require a cryptanalytic effort greater than or equal to the brute-force effort.

4.4 MAC Based HASH function

4.4.1 HMAC Design Objectives

RFC 2104 lists the following design objectives for HMAC:

- ✓ To use, without modifications, available hash functions. In particular, 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:

1. The idea of a keyed hash evolved into HMAC, designed to overcome some problems with the original proposals.
2. It involves hashing padded versions of the key concatenated with the message, and then with another outer hash of the result prepended by another padded variant of the key.
3. The hash function need only be used on 3 more blocks than when hashing just the original message (for the two keys + inner hash).
4. HMAC can use any desired hash function, and has been shown to have the same security as the underlying hash function. Can choose the hash function to use based on speed/security concerns.

- ➢ HMAC is specified as Internet standard RFC2104. It uses hash function on the message as given below

  $HMAC_K(M) = Hash [(K^+ XOR opad) || Hash [(K^+ XOR ipad) || M)]]$

elements are:

K^+ is K padded with zeros on the left so that the result is b bits in length

ipad is a pad value of 36 hex repeated to fill block

opad is a pad value of 5C hex repeated to fill block

M is the message input to HMAC (including the padding specified in the embedded hash function).

Fig.3.5, shows the structure of HMAC, which implements the function. The appeal of HMAC is that its designers have been able to prove an exact relationship between the strength of the embedded hash function and the strength of HMAC. The security of a MAC function is generally expressed in terms of the probability of successful forgery with a given amount of time spent by the forger and a given number of message-MAC pairs created with the same key.

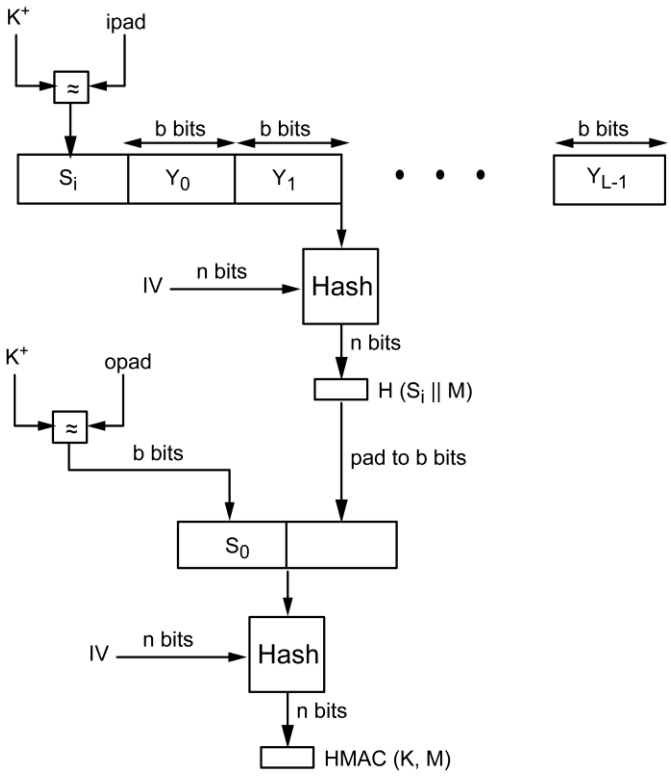


Fig. 3.4.

Have two classes of attacks: brute force attack on key used which has work of order 2^n; or a birthday attack which requires work of order 2^(n/2) - but which requires the attacker to observe 2^n blocks of messages using the same key - very unlikely.

4.5 MAC Based Block Ciphers:

4.5.1 CMAC

CMAC was previously described as the Data Authentication Algorithm, and it is based on DES. The algorithm is FIPS PUB 113, also known as the CBC-MAC (cipher block chaining message authentication code). This cipher-based MAC has been widely adopted in government and industry. Has been shown to be secure, with the following restriction.

- ✓ Only messages of one fixed length of mn bits are processed, where n is the cipher block size and m is a fixed positive integer.
- ✓ This limitation can be overcome using multiple keys, which can be derived from a single key.

- ✓ This refinement has been adopted by NIST as the cipher-based message authentication code (CMAC) mode of operation, for use with AES and triple DES. It is specified in NIST Special Publication 800-38B.

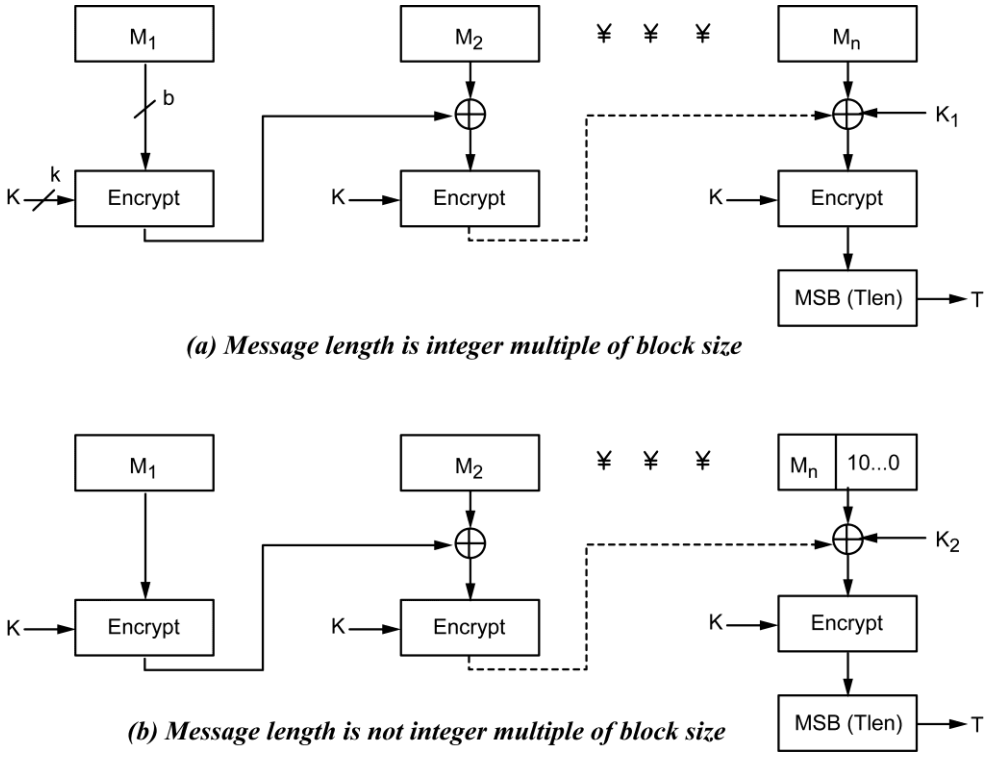


Fig. 3.5. Cipher-based Message Authentication Code (CMAC)

It uses the block size of the underlying cipher (*i.e.*, 128-bits for AES or 64-bits for triple-DES). The message is divided into n blocks $M_1..M_n$, padded if necessary. The algorithm makes use of a k-bit encryption key K and an n-bit constant K_1 or K_2 (depending on whether the message was padded or not). For AES, the key size k is 128,192, or 256 bits; for triple DES, the key size is 112 or 168 bits. The two constants K_1 & K_2 are derived from the original key K using encryption of 0 and multiplication in GF(2^n).

4.6 PKI trust models

RFC 2822 (Internet Security Glossary) defines public-key infrastructure (PKI) as the set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography. Its principal is to enable secure, convenient, and efficient acquisition of public keys. The IETF Public Key Infrastructure

X.509 (PKIX) working group has setup a formal (and generic) model based on X.509 that is suitable for deploying a certificate-based architecture on the Internet.

some key elements:

• End entity: A generic term used to denote end users, devices (e.g., servers, routers), or any other entity that can be identified in the subject field of a public key certificate. End entities canconsume and/or support PKI-related services.

• Certification authority (CA): The issuer of certificates and (usually) certificate revocation lists (CRLs). It may also support a variety of administrative functions, although these are often delegated to Registration Authorities.

• Registration authority (RA): An optional component that can assume a number of administrative functions from the CA. The RA is often associated with the End Entity registration process, but can assist in a number of other areas as well.

• CRL issuer: An optional component that a CA can delegate to publish CRLs.

• Repository: A generic term used to denote any method for storing certificates and CRLs so that they can be retrieved by End Entities.

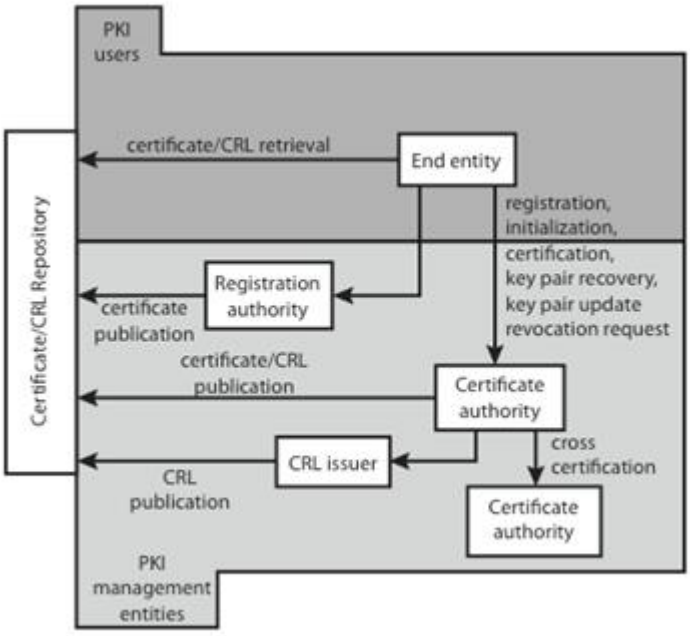


Figure 14.16 Public Key Infrastructure

4.6.1 PKIX Management

PKIX identifies a number of management functions that potentially need to be supported by management protocols, as shown in Figure 14.16:

• **Registration:** whereby a user first makes itself known to a CA, prior to issue of a certificate(s) for that user. It usually involves some off-line or online procedure for mutual authentication.

• **Initialization:** to install key materials that have the appropriate relationship with keys stored elsewhere in the infrastructure.

• **Certification:** process where a CA issues a certificate for a user's public key, and returns it to the user's client system and/or posts it in a repository.

• **Key pair recovery:** a mechanism to recover the necessary decryption keys when normal access to the keying material is no longer possible.

• **Key pair update:** key pairs need to be updated and new certificates issued.

• **Revocation request:** when authorized person advises need for certificate revocation, e.g. private key compromise, affiliation change, name change.

• **Cross certification:** when two CAs exchange information used in establishing a cross-certificate, issued by one CA to another CA that contains a CA signature key used for issuing certificates.

The PKIX working group has defines two alternative management protocols between PKIX entities. RFC 2510 defines the certificate management protocols (CMP), which is a flexible protocol able to accommodate a variety of technical, operational, and business models. RFC 2797 defines certificate management messages over CMS (RFC 2630) called CMC. This is built on earlier work to leverage existing code.