UNIT-3

Message Authentication Algorithms and Hash Function: Authentication Requirements, Functions, Message Authentication Codes, Hash Functions, Secure Hash Algorithms, Whirlpool, HMAC, CMAC, Digital Signatures, Knapsack Algorithm, ***Authentication Applications:*** *Kerberos, X.509 Authentication Services, Public-Key Infrastructure, Biometric 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 may also verify sequencing and timeliness. It is intended against the attacks like content modification, sequence modification, timing modification and repudiation. For repudiation, concept of digital signatures is used to counter it. There are three classes by which different types of functions that may be used to produce an authenticator. They are:

- ***Message encryption***–the ciphertext serves as authenticator
- ***Message authentication code (MAC)***–a public function of the message and a secret key producing a fixed-length value to serve as authenticator. This does not provide a digital signature because A and B share the same key.
- ***Hash function***–a public function mapping an arbitrary length message into a fixed- length hash value to serve as authenticator. This does not provide a digital signature because there is no key.

MESSAGE ENCRYPTION:

Message encryption by itself can provide a measure of authentication. The analysis differs for conventional and public-key encryption schemes. The message must have come from the sender itself, because the ciphertext can be decrypted using his (secret or public) key. Also, none of the bits in the message have been altered because an opponent does not know how to manipulate the bits of the ciphertext to induce meaningful changes to the plaintext. Often one needs alternative authentication schemes than just encrypting the message.

- Sometimes one needs to avoid encryption of full messages due to legal requirements.
- Encryption and authentication may be separated in the system architecture.

The different ways in which message encryption can provide authentication, confidentiality in both symmetric and asymmetric encryption techniques is explained with the table below:

Confidentiality and Authentication Implications of Message Encryption

A → B: $E_K[M]$ •Provides confidentiality —Only A and B share K •Provides a degree of authentication —Could come only from A —Has not been altered in transit —Requires some formatting/redundancy •Does not provide signature —Receiver could forge message —Sender could deny message (a) Symmetric encryption
A → B: $E_{KU_b}[M]$ •Provides confidentiality —Only B has KR_b to decrypt •Provides no authentication —Any party could use KU_b to encrypt message and claim to be A (b) Public-key encryption: confidentiality
A → B: $E_{KR_a}[M]$ •Provides authentication and signature —Only A has KR_a to encrypt —Has not been altered in transit —Requires some formatting/redundancy —Any party can use KU_a to verify signature (c) Public-key encryption: authentication and signature
A → B: $E_{KU_b}[E_{KR_a}(M)]$ •Provides confidentiality because of KU_b •Provides authentication and signature because of Kr_a (d) Public-key encryption: confidentiality, authentication, and signature

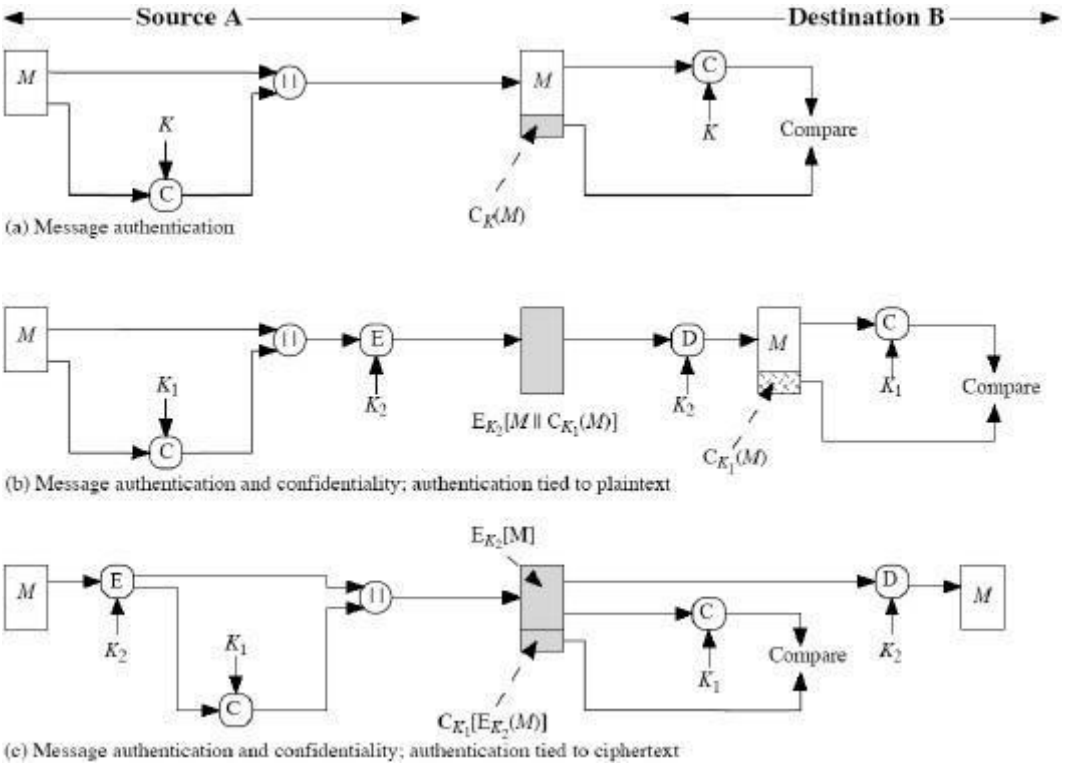
MESSAGE AUTHENTICATION CODE

An alternative authentication technique involves the use of a secret key to generate a small fixed-size block of data, known as cryptographic checksum or MAC, which is appended to the message. This technique assumes that both the

communicating parties say A and B share a common secret key K. When A has a message to send to B, it calculates MAC as a function C of key and message given as:
MAC=Ck(M) The message
and the MAC are transmitted to the intended recipient, who upon receiving performs the same calculation on the received message, using the same secret key to generate a new MAC. The received MAC is compared to the calculated MAC and only if they match, then:

1. The receiver is assured that the message has not been altered: Any alternations been done the MAC's do not match.
2. The receiver is assured that the message is from the alleged sender: No one except the sender has the secret key and could prepare a message with a proper MAC.
3. If the message includes a sequence number, then receiver is assured of proper sequence as an attacker cannot successfully alter the sequence number.

Basic uses of Message Authentication Code (MAC) are shown in the figure:



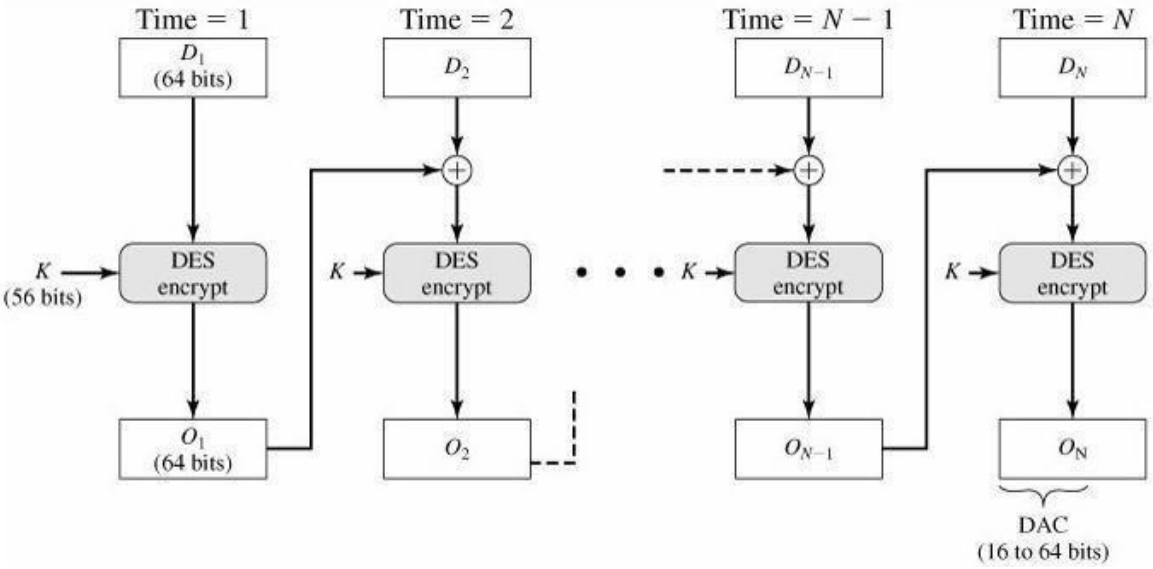
There are three different situations where use of a MAC is desirable:

- If a message is broadcast to several destinations in a network (such as a military control center), then it is cheaper and more reliable to have just one node responsible to evaluate the authenticity –message will be sent in plain with an attached authenticator.
- If one side has a heavy load, it cannot afford to decrypt all messages –it will just check the authenticity of some randomly selected messages.
- Authentication of computer programs in plaintext is very attractive service as they need not be decrypted every time wasting of processor resources. Integrity of the program can always be checked by MAC.

MESSAGE AUTHENTICATION CODE BASED ON DES

The Data Authentication Algorithm, based on DES, has been one of the most widely used MACs for a number of years. The algorithm is both a FIPS publication (FIPS PUB 113) and an ANSI standard (X9.17). But, security weaknesses in this algorithm have been discovered and it is being replaced by newer and stronger algorithms. The algorithm can be defined as using the cipher block chaining (CBC) mode of operation of DES shown below with an initialization vector of zero.

The data (e.g., message, record, file, or program) to be authenticated are grouped into



contiguous 64-bit blocks: D1, D2,..., DN. 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 as follows:

$O_1 = E(K, D_1)$

$O_2 = E(K, [D_2 \oplus O_1])$

$O_3 = (K, [D_3 \oplus O_2])$

•

•

•

$O_N = E(K, [D_N \oplus O_{N1}])$

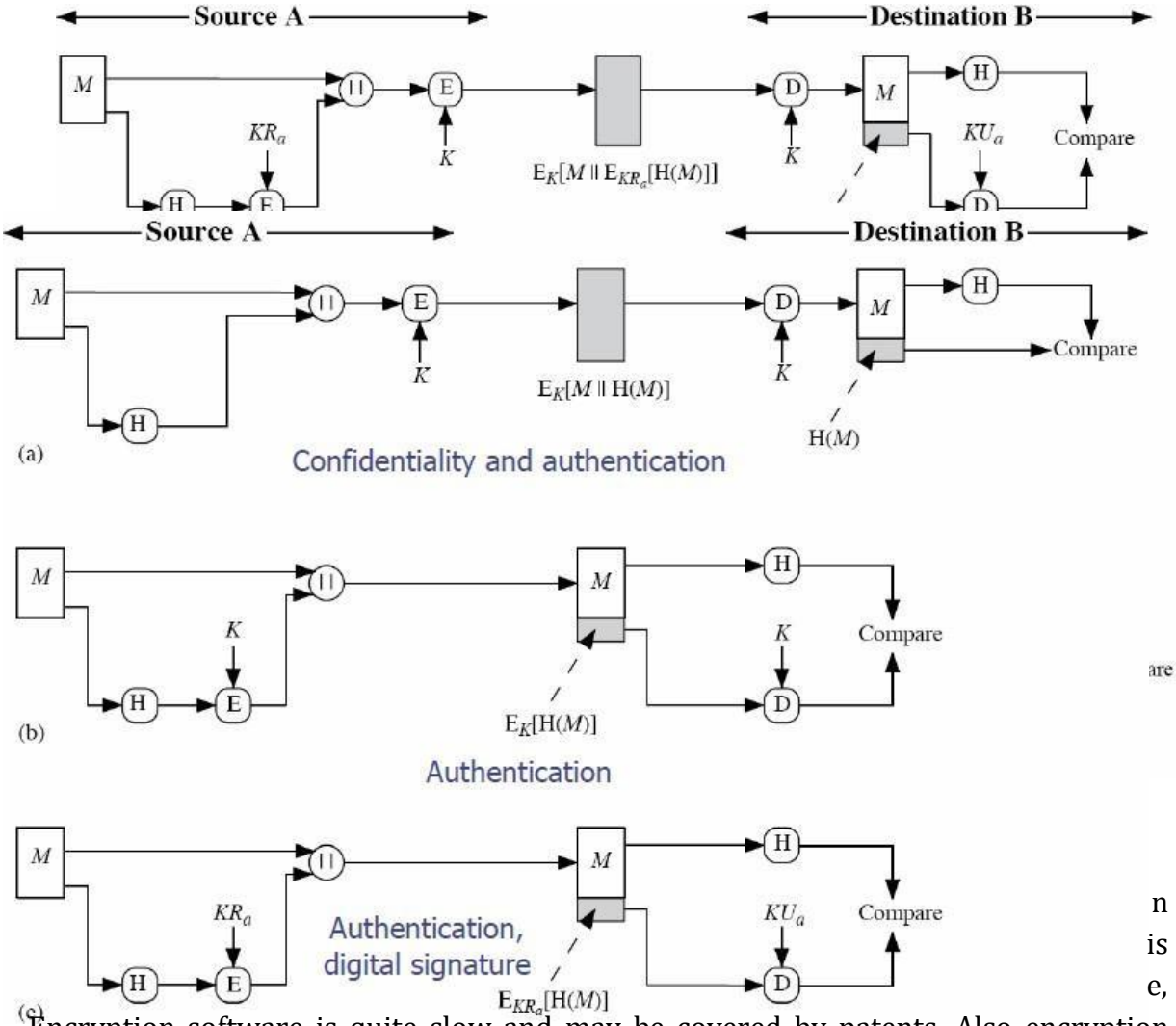
The DAC consists of either the entire block ON or the leftmost M bits of the block, with 16

≤ M ≤ 64 Use of MAC needs a shared secret key between the communicating parties and also MAC does not provide digital signature. The following table summarizes the confidentiality and authentication implications of the approaches shown above.

HASH FUNCTION

A variation on the message authentication code is the one-way hash function. As with the message authentication code, the hash function accepts a variable-size message M as input and produces a fixed-size hash code H(M), sometimes called a message digest, as output. The hash code is a function of all bits of the message and provides an error- detection capability: A change to any bit or bits in the message results in a change to the hash code. A variety of ways in which a hash code can be used to provide message authentication is shown below and explained stepwise in the table.

$A \rightarrow B: E_K[M \parallel H(M)]$ •Provides confidentiality —Only A and B share K •Provides authentication —H(M) is cryptographically protected (a) Encrypt message plus hash code	$A \rightarrow B: E_K[M \parallel E_{KR_a}[H(M)]]$ •Provides authentication and digital signature •Provides confidentiality —Only A and B share K (d) Encrypt result of (c) - shared secret key
$A \rightarrow B: M \parallel E_K[H(M)]$ •Provides authentication —H(M) is cryptographically protected (b) Encrypt hash code - shared secret key	$A \rightarrow B: M \parallel H(M \parallel S)$ •Provides authentication —Only A and B share S (e) Compute hash code of message plus secret value
$A \rightarrow B: M \parallel E_{KR_a}[H(M)]$ •Provides authentication and digital signature —H(M) is cryptographically protected —Only A could create $E_{KR_a}[H(M)]$ (c) Encrypt hash code - sender's private key	$A \rightarrow B: E_K[M \parallel H(M) \parallel S]$ •Provides authentication —Only A and B share S •Provides confidentiality —Only A and B share K (f) Encrypt result of (c)



are

n

is

e,

Encryption software is quite slow and may be covered by patents. Also encryption hardware costs are not negligible and the algorithms are subject to U.S export control. A fixed-length hash value h is generated by a function H that takes as input a message of arbitrary length**: h=H(M).**

sends M and H(M)

authenticates the message by computing H(M) and checking the match

Requirements for a hash function: The purpose of a hash function is to produce a "fingerprint" of a file, message, or other block of data. To be used for message

authentication, the hash function H must have the following properties

can be applied to a message of any size produces fixed-length output

- Computationally easy to compute H(M) for any given M
- H Computationally infeasible to find M such that H(M)=h, for a given h, referred to as the

one-way property

- Computationally infeasible to find M' such that H(M')=H(M), for a given M, referred to as *weak collision resistance*.
- Computationally infeasible to find M,M' with H(M)=H(M') (to resist to birthday attacks), referred to as *strong collision resistance*.

Examples of simple hash functions are:

- Bit-by-bit XOR of plaintext blocks: h= D1⊕D2⊕…⊕DN
- Rotated XOR –before each addition the hash value is rotated to the left with 1 bit
- Cipher block chaining technique without a secret key.

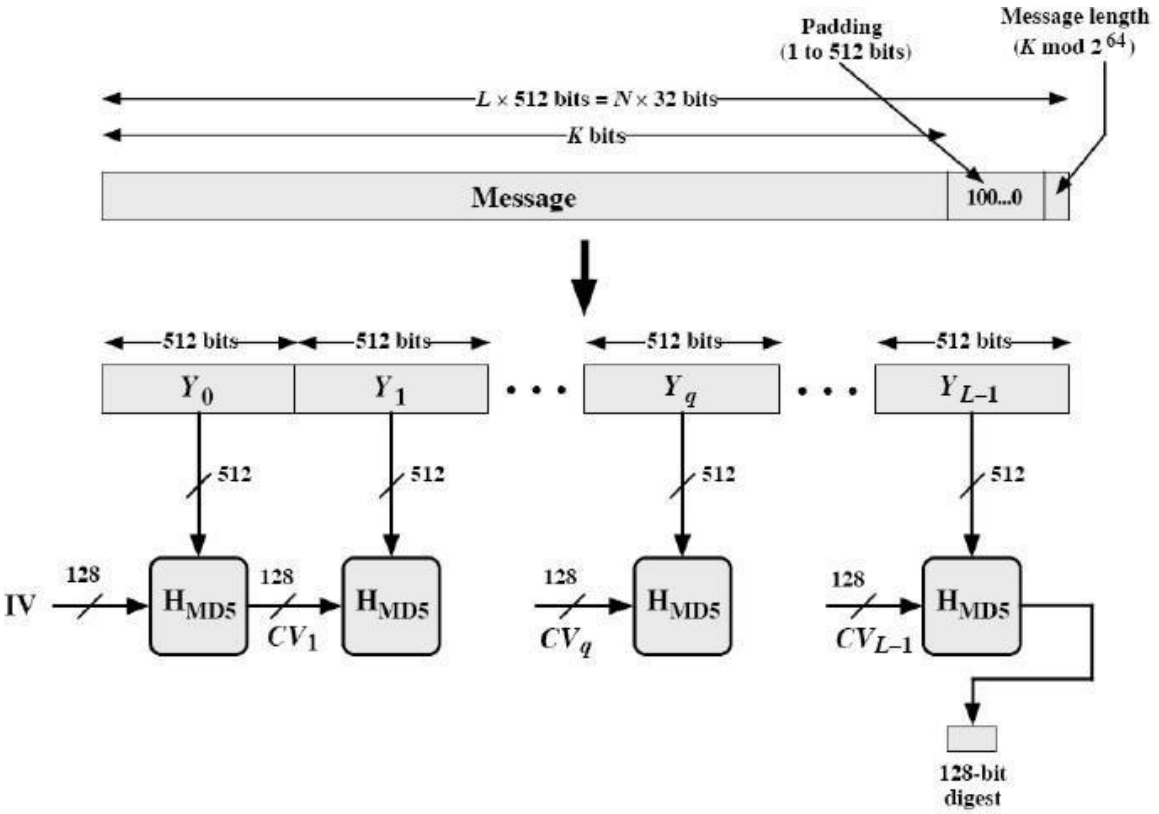
MD5 Message Digest Algorithm

The MD5 message-digest algorithm was developed by Ron Rivest at MIT and it remained as the most popular hash algorithm until recently. The algorithm takes as input, a message of arbitrary length and produces as output, a 128-bit message digest. The input is processed in 512-bit blocks. The processing consists of the following steps:

1.) *Append Padding bits*: The message is padded so that its length in bits is congruent to 448 modulo 512 i.e. the length of the padded message is 64 bits less than an integer multiple of 512 bits. Padding is always added, even if the message is already of the desired length. Padding consists of a single 1-bit followed by the necessary number of 0-bits.

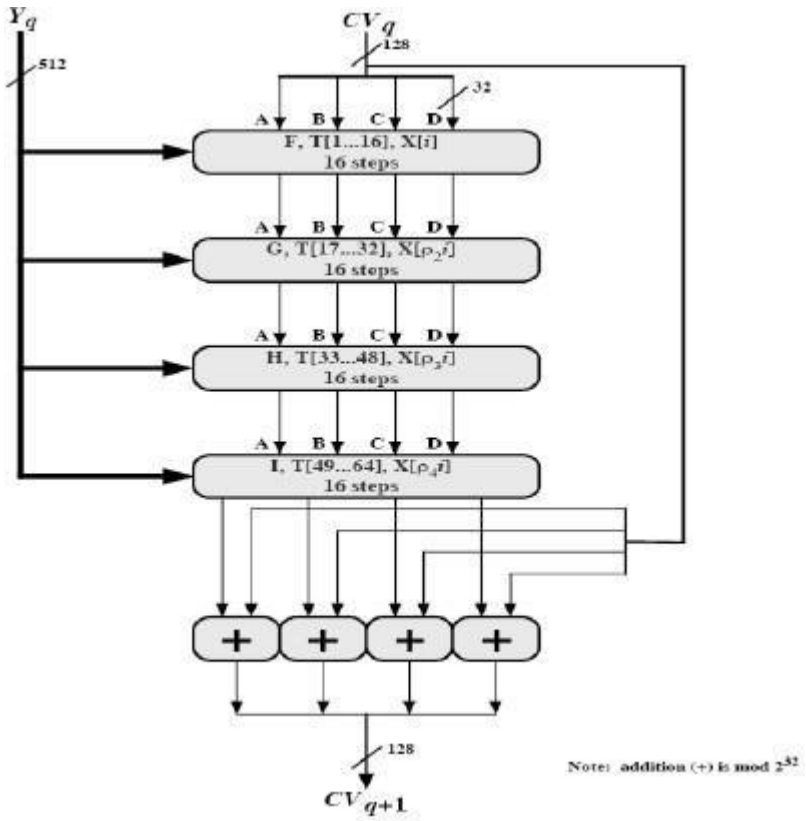
2.) *Append length*: A 64-bit representation of the length in bits of the original message (before the padding) is appended to the result of step-1. If the length is larger than 264, the 64 least representative bits are taken.

3.) *Initialize MD buffer*: A 128-bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as four 32-bit registers (A, B, C, D) and are initialized with A=0x01234567, B=0x89ABCDEF, C=0xFEDCBA98, D=0x76543210 i.e. 32-bit integers (hexadecimal values).



Message Digest Generation Using MD5

4.) *Process Message in 512-bit (16-word) blocks*: The heart of algorithm is the compression function that consists of four rounds of processing and this module is labeled HMD5 in the above figure and logic is illustrated in the following figure. The four rounds have a similar structure, but each uses a different primitive logical function, referred to as F, G, H and I in the specification. Each block takes as input the current 512-bit block being processed Yq and the 128-bit buffer value ABCD and updates the contents of the buffer. Each round also makes use of one-fourth of a 64-element table T*1....64+, constructed from the sine function. The ith element of T, denoted T[i], has the value equal to the integer part of 232 * abs(sin(i)), where i is in radians. As the value of abs(sin(i)) is a value between 0 and 1, each element of T is an integer that can be represented in 32-bits and would eliminate any regularities in the input data. The output of fourth round is added to the input to the first round (CVq) to produce CVq+1. The addition is done independently for each of the four words in the buffer with each of the corresponding words in CVq, using addition modulo 232. This operation is shown in the figure below:



5.) *Output*: After all L 512-bit blocks have been processed, the output from the Lth stage is the 128- bit message digest. MD5 can be summarized as follows:

$CV0 = IV\ CV_{q+1} = SUM_{32}(CV_q, RF_I Y_q RF_H[Y_q, RF_G[Y_q, RF_F[Y_q, CV_q]]]])\ MD = CV_L$

Where,

IV = initial value of ABCD buffer, defined in step 3. Y_q = the q_{th} 512-bit block of the message

L = the number of blocks in the message

CV_q = chaining variable processed with the q_{th} block of the message. RFx = round function using primitive logical function x.

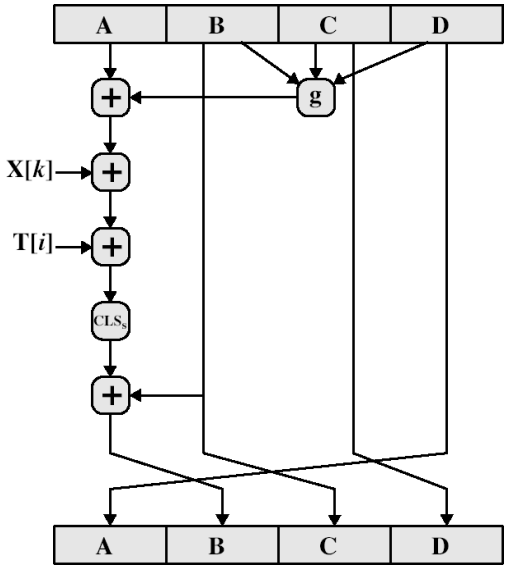
MD = final message digest value

SUM_{32} = Addition modulo 2_{32} performed separately.

MD5 Compression Function:

Each round consists of a sequence of 16 steps operating on the buffer ABCD. Each step is of the form, **a = b+((a+g(b,c,d)+X[k]+T[i])<<<s)**

where a, b, c, d refer to the four words of the buffer but used in varying permutations. After 16 steps, each word is updated 4 times. g(b,c,d) is a different nonlinear function in each round (F,G,H,I). Elementary MD5 operation of a single step is shown below.



The primitive function g of the F,G,H,I is given as:

Round	Primitive function g	g(b, c, d)
1	F(b, c, d)	(b ∧ c) ∨ (b' ∧ d)
2	G(b, c, d)	(b ∧ d) ∨ (c ∧ d')
3	H(b, c, d)	b ⊕ c ⊕ d
4	I(b c, d)	c ⊕ (b ∨ d')

Truth table

b	c	d	F	G	H	I
0	0	0	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	0	1	1
1	0	1	0	1	0	1
1	1	0	1	1	0	0
1	1	1	1	1	1	0

Where the logical operators (AND, OR, NOT, XOR) are represented by the symbols (∧,∨, **~,(+)).**

Each round mixes the buffer input with the next "word" of the message in a complex, non- linear manner. A different non-linear function is used in each of the 4 rounds (but the same function for all 16 steps in a round). The 4 buffer words (a,b,c,d) are rotated from step to step so all are used and updated. g is one of the primitive functions F,G,H,I for the 4 rounds respectively. X[k] is the kth 32-bit word in the current message block. T[i] is the ith entry in the matrix of constants T. The addition of varying constants T and the use of different shifts helps ensure it is extremely difficult to compute collisions. The array of 32-bit words X[0..15] holds the value of current 512-bit input block being processed. Within a round, each of the 16 words of X[i] is used exactly once, during one step. The order in which these words is used varies from round to round. In the first round, the
words are used in their original order. For rounds 2 through 4, the following permutations are used

- p2(i) = (1 + 5i) mod 16
- p3(i) = (5 +

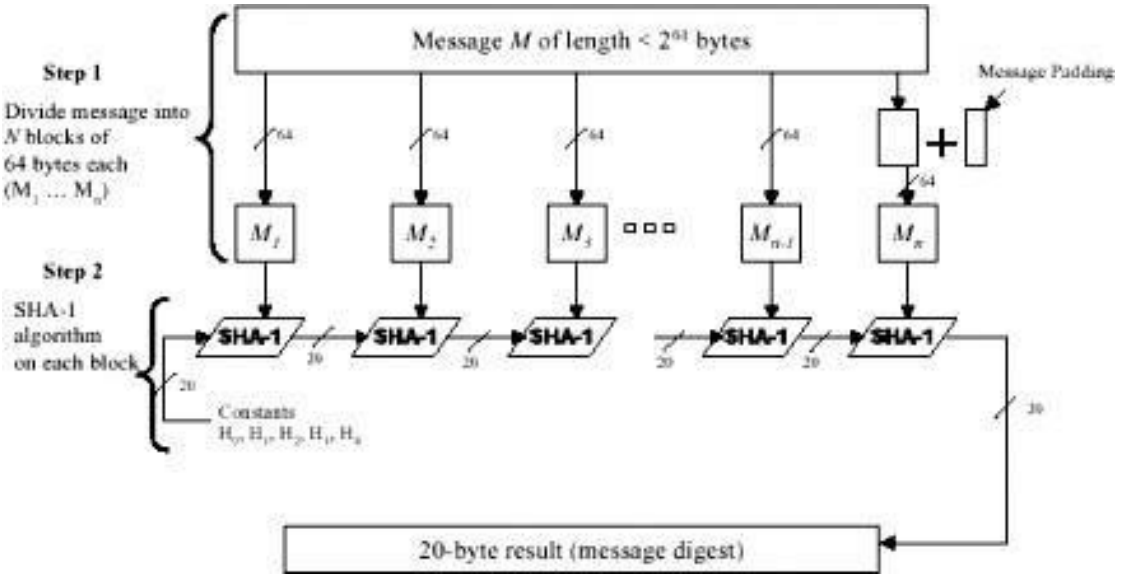
3i) mod 16
p4(I) = 7i mod 16

MD4

- Precursor to MD5
- Design goals of MD4 (which are carried over to MD5) Security
- Speed
- Simplicity and compactness
- Favor little-endian architecture
- Main differences
- A between MD5 and MD4 fourth round has been added.
- Each step now has a unique additive constant.
- The function g in round 2 was changed from (bc v bd v cd) to (bd v cd') to make g less symmetric.
- Each step now adds in the result of the previous step. This promotes a faster "avalanche effect".
- The order in which input words are accessed in rounds 2 and 3 is changed, to make these patterns less like each other.
- The shift amounts in each round have been approximately optimized, to yield a faster "avalanche effect." The shifts in different rounds are distinct.

SECURE HASH ALGORITHM

The secure hash algorithm (SHA) was developed by the National Institute of Standards and Technology (NIST). SHA-1 is the best established of the existing SHA hash functions, and is employed in several widely used security applications and protocols. The algorithm takes as input a message with a maximum length of less than 264 bits and produces as output a 160-bit message digest.



The input is processed in 512-bit blocks. The overall processing of a message follows the structure of MD5 with block length of 512 bits and a hash length and chaining variable length of 160 bits. The processing consists of following steps:

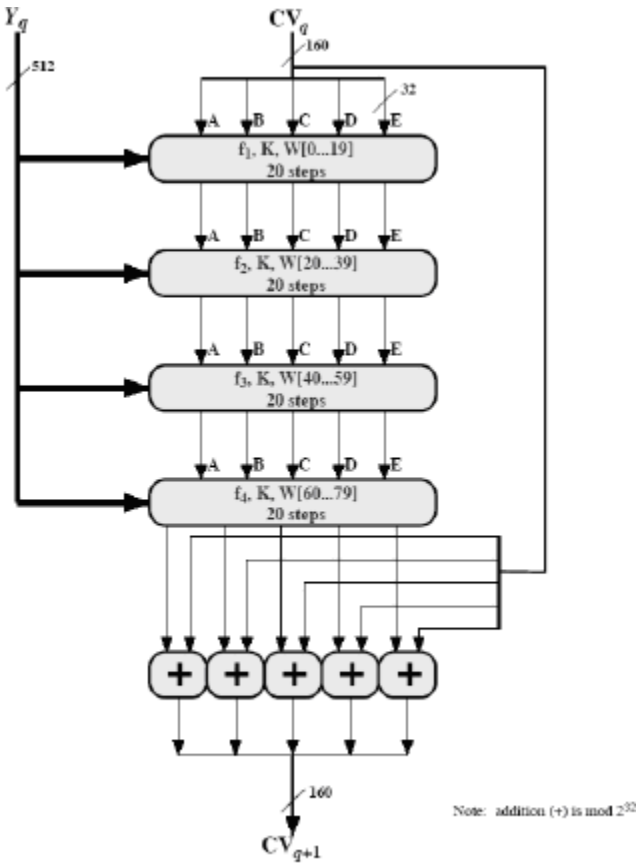
1.) ***Append Padding Bits:*** The message is padded so that length is congruent to 448 modulo 512; padding always added –one bit 1 followed by the necessary number of 0 bits.

2.) ***Append Length:*** a block of 64 bits containing the length of the original message is added. 3.) ***Initialize MD buffer*****:** A 160-bit buffer is used to hold intermediate and final results on the hash function. This is formed by 32-bit registers A,B,C,D,E. Initial values: A=0x67452301, B=0xEFCDAB89, C=0x98BADCFE, D=0x10325476, E=C3D2E1F0. Stores in big-endian format

i.e. the most significant bit in low address.

4.) ***Process message in blocks 512-bit (16-word) blocks*****:** The processing of a single 512-bit block is shown above. It consists of four rounds of processing of 20 steps each. These four rounds have similar structure, but uses a different primitive logical function, which we refer to as f1, f2, f3 and f4. Each round takes as input the current 512-bit block being processed and the 160-bit buffer value ABCDE and updates the contents of the buffer. Each round also makes use of four distinct additive constants Kt. The output of the fourth round i.e. eightieth step is added to the input to the first round to produce CV_{q+1}.

5.) *Output:* After all L 512-bit blocks have been processed, the output from the Lth stage is the 160-bit message digest.

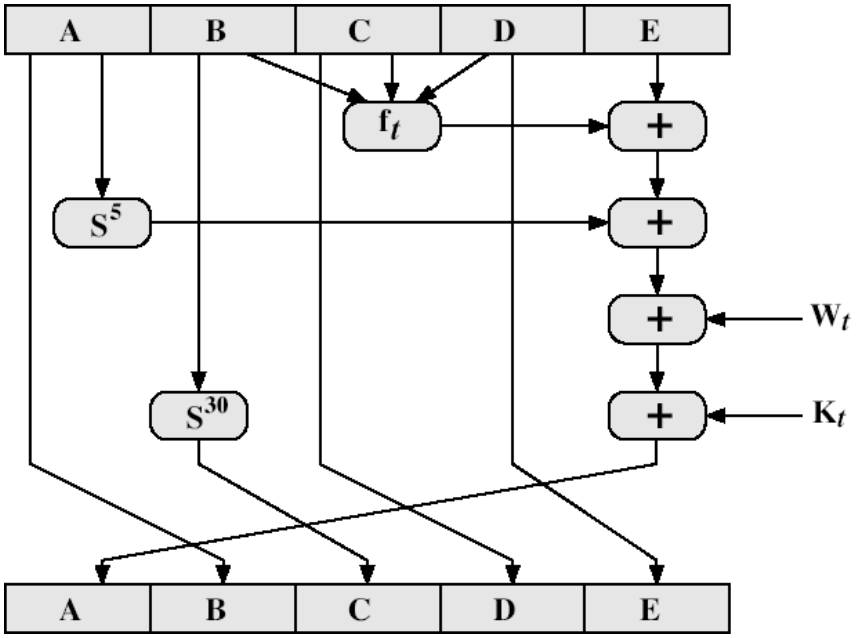


SHA-1 Processing of a Single 512-bit Block
(SHA-1 Compression Function)

The behavior of SHA-1 is as follows: **CV0 = IV CVq+1 = SUM32(CVq, ABCDEq) MD = CVL** Where, IV = initial value of ABCDE buffer ABCDEq = output of last round of processing of qth message block L = number of blocks in the message SUM32 = Addition modulo 232 MD = final message digest value.

SHA-1 Compression Function:

Each round has 20 steps which replaces the 5 buffer words. The logic present in each one of the 80 rounds present is given as **(A,B,C,D,E) <- (E + f(t,B,C,D) + S_5(A)+ Wt+ Kt),A,S_{30}(B),C,D** Where, A, B, C, D, E = the five words of the buffer t = step number; 0< t
< 79 f(t,B,C,D) = primitive logical function for step t S_k = circular left shift of the 32-bit argument by k bits Wt **=** a 32-bit word derived from current 512-bit input block. Kt **=** an additive constant; four distinct values are used + = modulo addition.

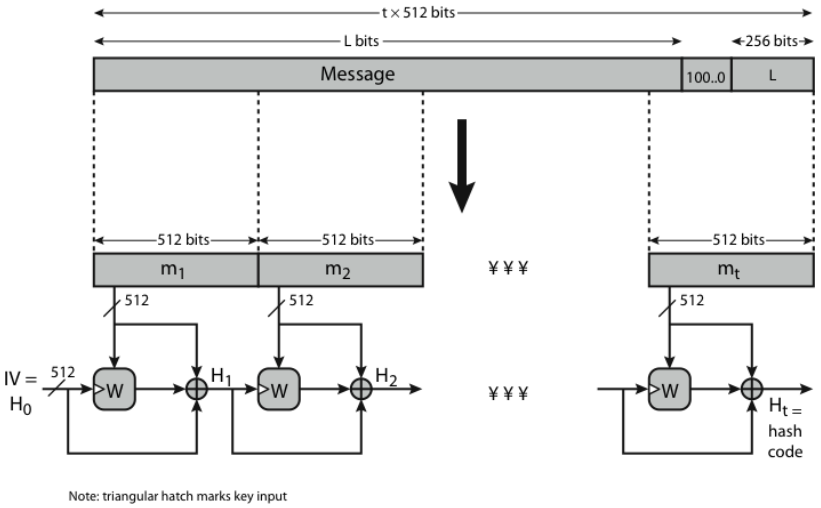


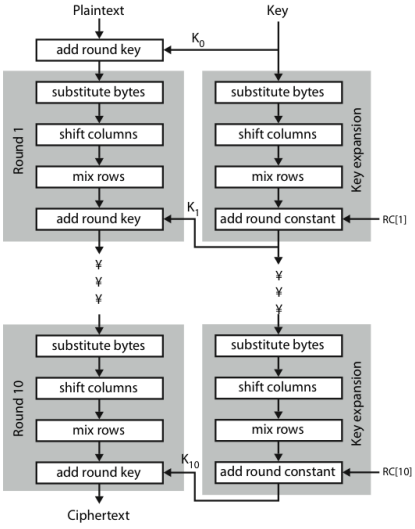
SHA shares much in common with MD4/5, but with 20 instead of 16 steps in each of the 4 rounds. Note the 4 constants are based on sqrt(2,3,5,10). Note also that instead of just splitting the input block into 32-bit words and using them directly, SHA-1 shuffles and mixes them using rotates & XOR's to form a more complex input, and greatly increases the difficulty of finding collisions. A sequence of logical functions f_0, f_1,..., f_{79} is used in the SHA-1. Each f_t, 0<=t<=79, operates on three 32-bit words B, C, D and produces a 32-bit word as output. f_t(B,C,D) is defined as follows: for words B, C, D, **f_t(B,C,D) = (B AND C) OR ((NOT B) AND D) (0 <= t <= 19) f_t(B,C,D) = B XOR C XOR D (20 <= t <= 39) f_t(B,C,D) = (B AND C) OR (B AND D) OR (C AND D) (40 <= t <= 59) f_t(B,C,D) = B XOR C XOR D (60 <= t <= 79).**

WHIRLPOOL HASH FUNCTION

- Created by Vincent Rijmen and Paulo S. L. M. Barreto
- Hashes messages of plaintext length 2^256
- Result is a 512 bit message
- Three versions have been released – WHIRLPOOL-0 – WHIRLPOOL-T – WHIRLPOOL
  - designed specifically for hash function use
  - with security and efficiency of AES
  - but with 512-bit block size and hence hash
  - similar structure & functions as AES but
    - input is mapped row wise
    - has 10 rounds
    - a different primitive polynomial for GF(2^8)
    - uses different S-box design & values
- "W" is a 512-bit block cipher
- "m" is the plaintext, split into 512 bit blocks
- "H" is the blocks formed from the hashes

WHIRLPOOL OVERVIEW





• The block cipher W is the core element of the Whirlpool hash function
• It is comprised of 4 steps.
– Add Round Key
– Shift Columns
– Mix Rows
– Substitute bytes

Add Round Key

• During the Add Round Key step, the message is XOR'd with the key
• If this is the first message block being run through, the key is a block of allzeros
• If this is any block except the first, the key is the digest of the previous block

Shift Columns

• Starting from left to right, each column gets rotated vertically a number of bytes equal to which number column it is, from top to bottom –
Ex:

Mix Rows

- [0,0][0,1][0,2] [0,0][2,1][1,2]
- [1,0][1,1][1,2] ------> [1,0][0,1][2,2]
- [2,0][2,1][2,2] [2,0][1,1][0,2]
- Each row gets shifted horizontally by the number of row it is.

Similar to the shift column function, but rotated left to right –
Ex:

- [0,0][0,1][0,2] [0,0][0,1][0,2]
- [1,0][1,1][1,2] ------> [1,2][1,0][1,2]
- [2,0][2,1][2,2] [2,1][2,2][0,2]

Substitute bytes

- Each byte in the message is passed through a set of s-boxes
- The output of this is then set to be the key for the next round

HMAC

Interest in developing a MAC, derived from a cryptographic hash code has been increasing mainly because hash functions are generally faster and are also not limited by export restrictions unlike block ciphers. Additional reason also would be that the library code for cryptographic hash functions is widely available. The original proposal is for incorporation of a secret key into an existing hash algorithm and the approach that received most support is HMAC. HMAC is specified as Internet standard RFC2104. It
makes use of the hash function on the given message. Any of MD5,
SHA-1, RIPEMD-160 can be used.

HMAC Design Objectives

- To use, without modifications, available hash functions
- To allow for easy replaceability of the embedded hash function To preserve the original performance of the hash function
- To use and handle keys in a simple way
- To have a well understood cryptographic analysis of the strength of the MAC based on reasonable assumptions on the embedded hash function

The first two objectives are very important for the acceptability of HMAC. HMAC treats the hash function as a “black box”, which has two benefits. First is that an existing implementation of the hash function can be used for implementing HMAC making the bulk of HMAC code readily available without modification. Second is that if ever an existing hash function is to be replaced, the existing hash function module is removed and new module is dropped in. The last design objective provides the main advantage of HMAC over other proposed hash-based schemes. HMAC can be proven secure provided that the embedded hash function has some reasonable cryptographic strengths.

Steps involved in HMAC algorithm:

1. Append zeroes to the left end of K to create a b-bit string K_+ (ex: If K is of length 160- bits and b = 512, then K will be appended with 44 zero bytes).
2. XOR(bitwise exclusive-OR) K_+ with ipad to produce the b-bit block S_i.
3. Append M to S_i.
4. Now 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.

HMAC Algorithm

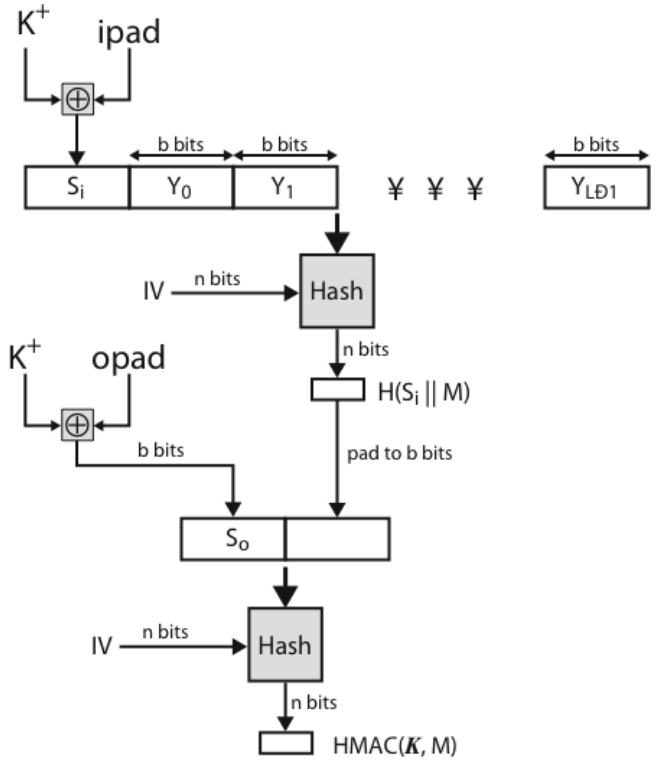
- **Define the following terms**

H = embedded hash function
M = message input to HMAC
Y_i = i^{th} block of M, $0 \le i \le L - 1$
L = number of blocks in M
b = number of bits in a block
n = length of hash code produced by embedded hash function
K = secret key; if key length is greater than b, the key is input to the hash function to produce an n-bit key; recommended length ≥ n
K^+ = K padded with 0's on the left so that the result is b bits in length
ipad = 00110110 repeated b/8 times
opad = 01011100 repeated b/8 times

- **Then HMAC can be expressed as**

$$HMAC_K = H[(K^+ \oplus opad) \,||\, H[K^+ \oplus ipad) \,||\, M]]$$

HMAC Structure



The XOR with ipad results in flipping one-half of the bits of K. Similarly, XOR with opad results in flipping one-half of the bits of K, but different set of bits. By passing S_i

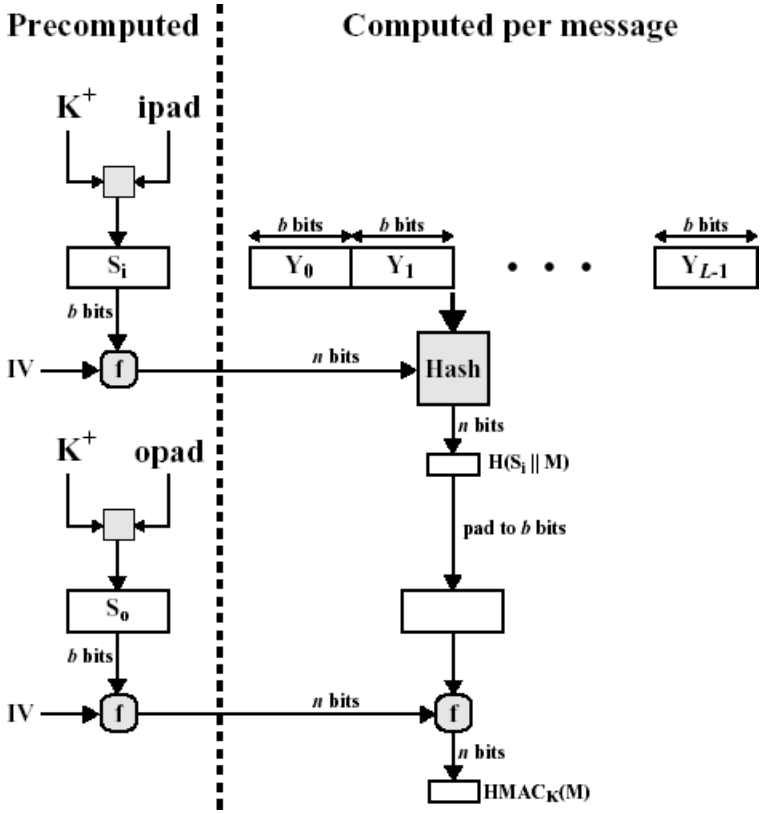
and S_0 through the compression function of the hash algorithm, we have pseudorandomly generated two keys from K.

HMAC should execute in approximately the same time as the embedded hash function for long messages. HMAC adds three executions of the hash compression function (for S_0, S_i, and the block produced from the inner hash)

A more efficient implementation is possible. Two quantities are precomputed. f(IV, (K_+ ⊕ ipad)

f(IV, (K_+ ⊕ opad)

where f is the compression function for the hash function which takes as arguments a chaining variable of n bits and a block of b-bits and produces a chaining variable of n bits.



As shown in the above figure, the values are needed to be computed initially and every time a key changes. The precomputed quantities substitute for the initial value (IV) in the hash function. With this implementation, only one additional instance of the compression function is added to the processing normally produced by the hash function. This implementation is worthwhile if most of the messages for which a MAC is computed are short.

Security of HMAC:

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. Have two classes of attacks on the embedded hash function:

1. The attacker is able to compute an output of the compression function even with an IV that is random, secret and unknown to the attacker.
2. The attacker finds collisions in the hash function even when the IV is random and

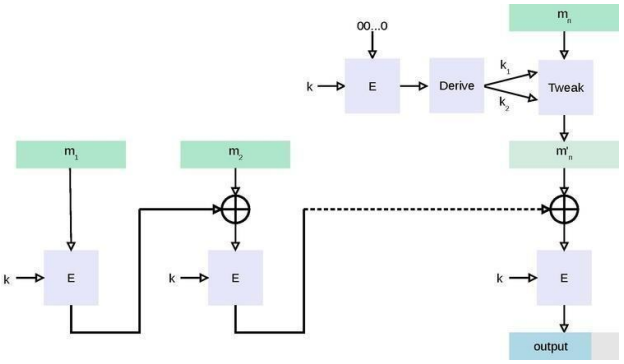
secret.

These attacks are likely to be caused by 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. So even MD5 is still secure for use in HMAC given these constraints.

CMAC

In cryptography, **CMAC** (Cipher-based Message Authentication Code)[1] is a block cipher- based message authentication code algorithm. It may be used to provide assurance of the authenticity and, hence, the integrity of binary data. This mode of operation fixes security deficiencies of CBC-MAC (CBC-MAC is secure only for fixed-length messages).

The core of the CMAC algorithm is a variation of CBC-MAC that Black and Rogaway proposed and analyzed under the name XCBC[2] and submitted to NIST.[3] The XCBC algorithm efficiently addresses the security deficiencies of CBC-MAC, but requires three keys. Iwata and Kurosawa proposed an improvement of XCBC and named the resulting algorithm One-Key CBC-MAC (OMAC) in their papers.[4][5] They later submitted OMAC1[6], a refinement of OMAC, and additional security analysis.[7] The OMAC algorithm reduces the amount of key material required for XCBC. CMAC is equivalent to OMAC1.



To generate an ℓ-bit CMAC tag (t) of a message (m) using a b-bit block cipher (E) and a secret key (k), one first generates two b-bit sub-keys (k_1 and k_2) using the following algorithm (this is equivalent to multiplication by x and x^2 in a finite field GF(2^b)). Let $\ll$ denote the standard left-shift operator and $\oplus$ denote exclusive or:

1. Calculate a temporary value $k_0 = E_k(0)$.
2. If msb(k_0) = 0, then $k_1 = k_0 \ll 1$, else $k_1 = (k_0 \ll 1) \oplus C$; where C is a certain constant that depends only on b. (Specifically, C is the non-leading coefficients of the lexicographically first irreducible degree-b binary polynomial with the minimal number of ones.)
3. If msb(k_1) = 0, then $k_2 = k_1 \ll 1$, else $k_2 = (k_1 \ll 1) \oplus C$.
4. Return keys (k_1, k_2) for the MAC generation process.

As a small example, suppose $b = 4$, $C = 0011_2$, and $k_0 = E_k(0) = 0101_2$. Then $k_1 = 1010_2$ and $k_2 = 0100 \oplus 0011 = 0111_2$.

The CMAC tag generation process is as follows:

1. Divide message into b-bit blocks $m = m_1 \| \ldots \| m_{n-1} \| m_n$ where $m_1, \ldots, m_{n-1}$ are

complete blocks. (The empty message is treated as 1 incomplete block.)

2. If m_n is a complete block then $m_n' = k_1 \oplus m_n$ else $m_n' = k_2 \oplus (m_n \| 10...0_2)$. 3. Let $c_0 = 00...0_2$.
4. For $i = 1, ..., n-1$, calculate $c_i = E_k(c_{i-1} \oplus m_i)$.
5. $c_n = E_k(c_{n-1} \oplus m_n')$
6. Output $t = \text{msb}_\ell(c_n)$.

The verification process is as follows:

1. Use the above algorithm to generate the tag.
2. Check that the generated tag is equal to the received tag.

DIGITAL SIGNATURE

The most important development from the work on public-key cryptography is the digital signature. Message authentication protects two parties who exchange messages from any third party. However, it does not protect the two parties against each other. A digital signature is analogous to the handwritten signature, and provides a set of security capabilities that would be difficult to implement in any other way. It must have the following properties: • It must verify the author and the date and time of the signature

• It must to authenticate the contents at the time of the signature • It must be verifiable by third parties, to resolve disputes Thus, the digital signature function includes the authentication function. A variety of approaches has been proposed for the digital signature function. These approaches fall into two categories: direct and arbitrated. **Direct Digital Signature**

Direct Digital Signatures involve the direct application of public-key algorithms involving only the communicating parties. A digital signature may be formed by encrypting the entire message with the sender's private key, or by encrypting a hash code of the message with the sender's private key. Confidentiality can be provided by further encrypting the entire message plus signature using either public or private key schemes. It is important to perform the signature function first and then an outer confidentiality function, since in case of dispute, some third party must view the message and its signature. But these approaches are dependent on the security of the sender's private-key. Will have problems if it is lost/stolen and signatures forged. Need time-stamps and timely key revocation. **Arbitrated Digital Signature**

The problems associated with direct digital signatures can be addressed by using an arbiter, in a variety of possible arrangements. The arbiter plays a sensitive and crucial role in this sort of scheme, and all parties must have a great deal of trust that the arbitration mechanism is working properly. These schemes can be implemented with either private or public-key algorithms, and the arbiter may or may not see the actual message contents. **Using Conventional encryption**

X A : M || E (Kxa ,[IDx || H (M)])

A Y : E(Kay ,[IDx || M || E (Kxa ,[IDx ||H(M))]) || T])

It is assumed that the sender X and the arbiter A share a secret key Kxa and that A and Y share secret key Kay. X constructs a message M and computes its hash value H(m) . Then X transmits the message plus a signature to A. the signature consists of an identifier IDx of X plus the hash value, all encrypted using Kxa.

A decrypts the signature and checks the hash value to validate the message. Then A transmits a message to Y, encrypted with Kay. The message includes IDx, the original message from X, the signature, and a timestamp.

Arbiter sees message

Problem : the arbiter could form an alliance with sender to deny a signed message, or with the receiver to forge the sender's signature.

Using Public Key Encryption

X : IDx ||E(PRx,[IDx|| E (PUy, E(PRx, M)]])

A Y : E(PRa, [IDx ||E (PUy, E (PRx, M))|| T])

X double encrypts a message M first with X's private key,PRx, and then with Y's public key, PUy. This is a signed, secret version of the message. This signed message, together with X's identifier , is encrypted again with PRx and, together with IDx, is sent to A. The inner, double encrypted message is secure from the arbiter (and everyone else exceptY)

A can decrypt the outer encryption to assure that the message must have come from X (because only X has PRx). Then A transmits a message to Y, encrypted with PRa. The message includes IDx, the double encrypted message, and a timestamp.

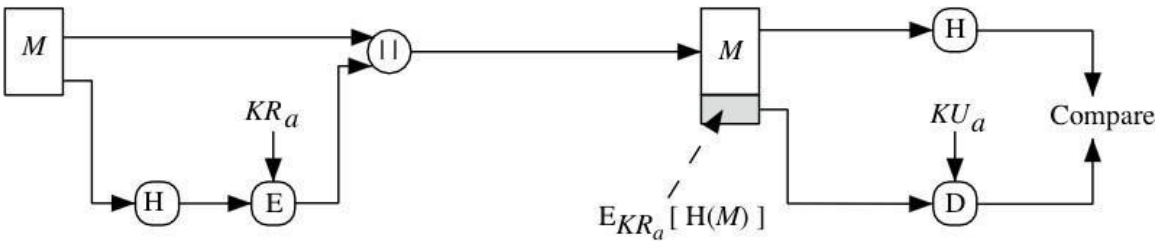
Arbiter does not see message

Digital Signature Standard (DSS)

The National Institute of Standards and Technology (NIST) has published Federal Information Processing Standard FIPS 186, known as the Digital Signature Standard (DSS). The DSS makes use of the Secure Hash Algorithm (SHA) and presents a new digital signature technique, the Digital Signature Algorithm (DSA). The DSS uses an algorithm that is designed to provide only the digital signature function and cannot be used for encryption or key exchange, unlike RSA.

The RSA approach is shown below. The message to be signed is input to a hash function that produces a secure hash code of fixed length. This hash code is then encrypted using the sender's private key to form the signature. Both the message and the signature are then transmitted.

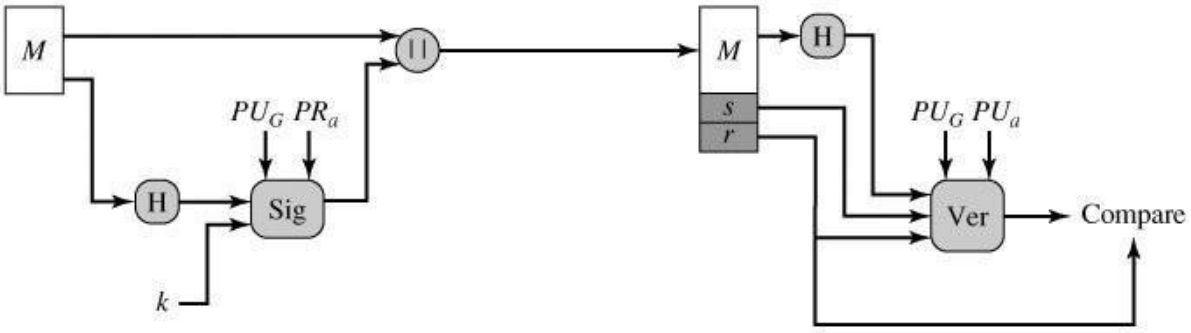
The recipient takes the message and produces a hash code. The recipient also



decrypts the signature using the sender's public key. If the calculated hash code matches the decrypted signature, the signature is accepted as valid. Because only the sender knows the private key, only the sender could have produced a valid signature.

The DSS approach also makes use of a hash function. The hash code is provided as input to a signature function along with a random number k generated for this particular signature. The signature function also depends on the sender's private key (PRa) and a set of parameters known to a group of communicating principals. We can consider this set to constitute a global public key (PUG).The result is a signature consisting of two components, labeled s and r.

At the receiving end, the hash code of the incoming message is generated. This plus the signature is input to a verification function. The verification function also



(b) DSS approach

depends on the global public key as well as the sender's public key (PUa), which is paired with the sender's private key. The output of the verification function is a value that is equal to the signature component r if the signature is valid. The signature function is such that only the sender, with knowledge of the private key, could have produced the validsignature.

KNAPSACK ALGORITHM

Public-Key cryptography was invented in the 1970s by Whitfield Diffie, Martin Hellman and Ralph Merkle.

Public-key cryptography needs two keys. One key tells you how to encrypt (or code) a message and this is "public" so anyone can use it. The other key allows you to decode (or decrypt) the message. This decryption code is kept secret (or private) so only the person who knows the key can decrypt the message. It is also possible for the person with the private key to encrypt a message with the private key, then anyone holding the public key can decrypt the message, although this seems to be of little use if you are trying to keep something secret!

The First General Public-Key Algorithm used what we call the Knapsack Algorithm. Although we now know that this algorithm is not secure we can use it to look at how these types of encryption mechanisms work.

The knapsack algorithm works like this:
Imagine you have a set of different weights which you can use to make any total weight that you need by adding combinations of any of these weights together.
Let us look at an example:
Imagine you had a set of weights 1, 6, 8, 15 and 24. To pack a knapsack weighing 30, you could use weights 1, 6, 8 and 15. It would not be possible to pack a knapsack that

weighs 17 but this might not matter.
You might represent the weight 30 by the binary code 11110 (one 1, one 6, one 8, one 15 and no 24).
Example:

Plain text	10011	11010	01011	00000
Knapsack	1 6 8 15 24	1 6 8 15 24	1 6 8 15 24	1 6 8 15 24
Cipher text	1 + 15 + 24 = 40	1 + 6 + 15 = 22	6 + 15 + 24 = 45	0 = 0

What total weights is it possible to make?

So, if someone sends you the code 38 this can only have come from the plain text 01101. When the Knapsack Algorithm is used in public key cryptography, the idea is to create two different knapsack problems. One is easy to solve, the other not. Using the easy knapsack, the hard knapsack is derived from it. The hard knapsack becomes the public key. The easy knapsack is the private key. The public key can be used to encrypt messages, but cannot be used to decrypt messages. The private key decrypts the messages.

The Superincreasing Knapsack Problem

An easy knapsack problem is one in which the weights are in a superincreasing sequence. A superincreasing sequence is one in which the next term of the sequence is greater than the sum of all preceding terms. For example, the set {1, 2, 4, 9, 20, 38} is
superincreasing, but the set {1, 2, 3, 9, 10, 24} is not because 10 < 1+2+3+9.
It is easy to solve a superincreasing knapsack. Simply take the total weight of the knapsack and compare it with the largest weight in the sequence. If the total weight is less than the number, then it is not in the knapsack. If the total weight is greater then the number, it is in the knapsack. Subtract the number from the total, and compare with the next highest number. Keep working this way until the total reaches zero. If the total doesn't reach zero, then there is no solution.

So, for example, if you have a knapsack that weighs 23 that has been made from the weights of the superincreasing series {1, 2, 4, 9, 20, 38} then it does not contain the weight 38 (as 38 > 23)
but it does contain the weight 20; leaving 3;
which does not contain the weight 9 still leaving 3; which does not contain the weight 4 still leaving 3;
which contains the weight 2, leaving 1; which contains the weight 1. The binary code is therefore 110010.

It is much harder to decrypt a non-superincreasing knapsack problem. Give a friend a

non-superincreasing knapsack and a total and see why this is the case.
One algorithm that uses a superincreasing knapsack for the private (easy) key and a non-superincreasing knapsack for the public key was created by Merkle and Hellman They did this by taking a superincreasing knapsack problem and converting it into a non-superincreasing one that could be made public, using modulus arithmetic.

Making the Public Key

To produce a normal knapsack sequence, take a superincreasing sequence; e.g. {1, 2, 4, 10, 20, 40}. Multiply all the values by a number, n, modulo m. The modulus should be a number greater than the sum of all the numbers in the sequence, for example, 110. The
multiplier should have no factors in common with the modulus. So let's choose 31.
The normal knapsack sequence would be:

1×31 mod(110) = 31
2×31 mod(110) = 62
4×31 mod(110) = 14
10×31 mod(110) = 90
20×31 mod(110) = 70
40×31 mod(110) = 30

So the public key is: {31, 62, 14, 90, 70, 30} and
the private key is {1, 2, 4, 10, 20.40}.

Let's try to send a message that
is in binary code:
100100111100101110
The knapsack contains six weights so we need to split the message
into groups of six: 100100
111100
101110
This corresponds to three sets of weights with
totals as follows 100100 = 31 + 90 = 121
111100 = 31+62+14+90 = 197
101110 = 31+14+90+70 =205
So the coded message is 121 197 205.

Now the receiver has to decode the message...
The person decoding must know the two numbers 110 and 31 (the modulus and the multiplier). Let's call the modulus "m" and the number you multiply by "n".
We need $n-1$, which is a multiplicative inverse of n mod m, i.e. $n(n-1) = 1 \bmod m$ In this case I have calculated $n-1$ to be 71.
All you then have to do is multiply each of the codes 71 mod 110 to find the total in the knapsack which contains {1, 2, 4, 10, 20, 40} and hence to decode the message.

The coded message is 121 197 205:

121×71 mod(110) = 11 = 100100
197×71 mod(110) = 17 = 111100
205×71 mod(110) = 35 = 101110

The
decoded
messag
e is:
100100
111100
101110.
Just as I thought!

Simple and short knapsack codes are far too easy to break to be of any real use. For a knapsack code to be reasonably secure it would need well over 200 terms each of length 200 bits.

AUTHENTICATION

APPLICATIONS

KERBEROS

Kerberos is an authentication service developed as part of Project Athena at MIT. It addresses the threats posed in an open distributed environment in which users at workstations wish to access services on servers distributed throughout the network. Some of these threats are:

- A user may gain access to a particular workstation and pretend to be another user operating from that workstation.
- A user may alter the network address of a workstation so that the requests sent from the altered workstation appear to come from the impersonated workstation.
- A user may eavesdrop on exchanges and use a replay attack to gain entrance to a server or to disrupt operations.

Two versions of Kerberos are in current use: Version-4 and Version-5. The first published report on Kerberos listed the following requirements:

Secure: A network eavesdropper should not be able to obtain the necessary information to impersonate a user. More generally, Kerberos should be strong enough that a potential opponent does not find it to be the weak link.

Reliable: For all services that rely on Kerberos for access control, lack of availability of the Kerberos service means lack of availability of the supported services. Hence, Kerberos should be highly reliable and should employ a distributed server architecture, with one system able to back up another.

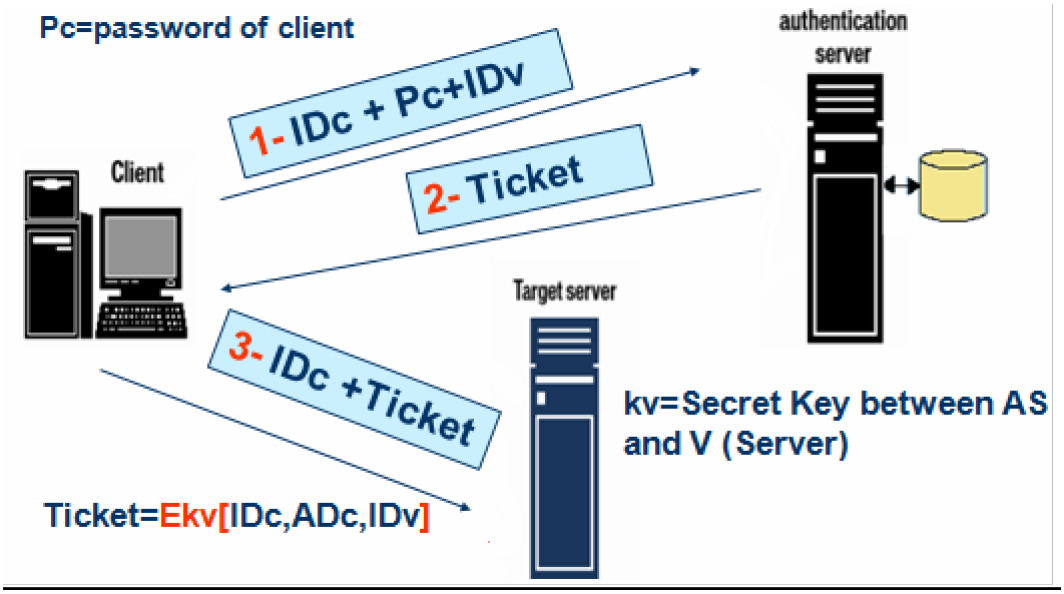
Transparent: Ideally, the user should not be aware that authentication is taking place, beyond the requirement to enter a password.

Scalable: The system should be capable of supporting large numbers of clients and servers. This suggests a modular, distributed architecture

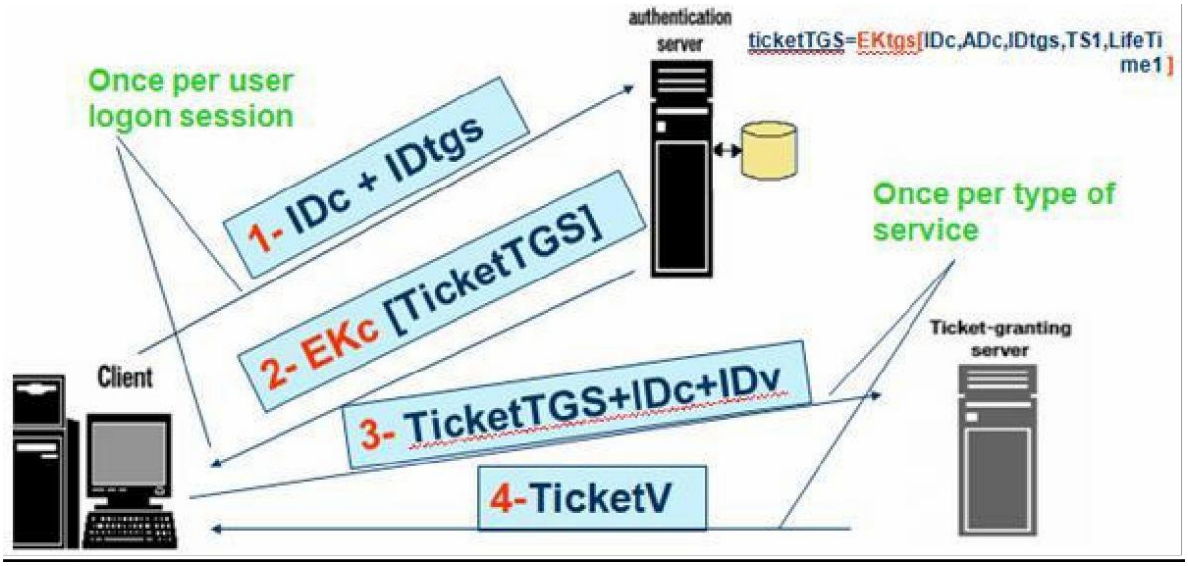
Two versions of Kerberos are in common use: Version 4 is most widely used version. Version 5 corrects some of the security deficiencies of Version 4. Version 5 has been issued as a draft Internet Standard (RFC 1510)

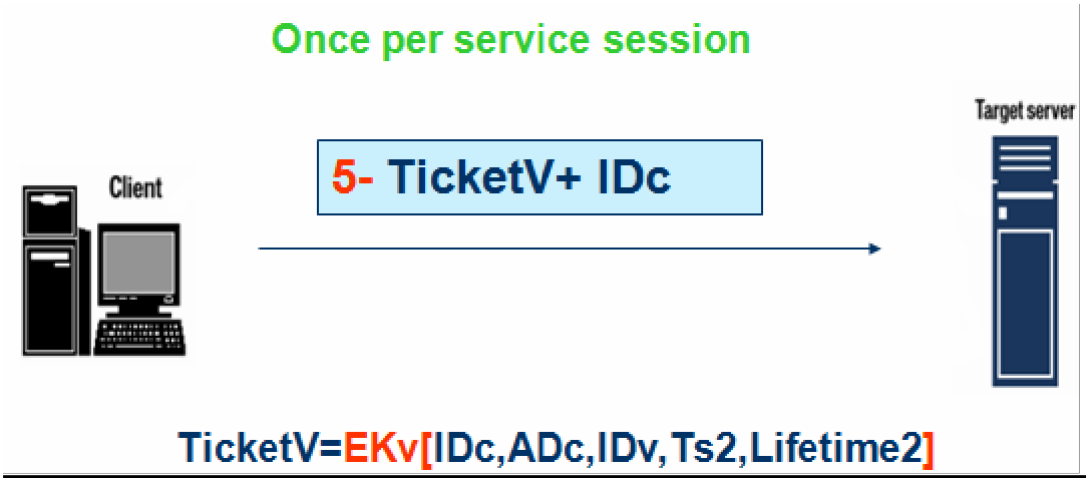
KERBEROS VERSION 4

1.) **SIMPLE DIALOGUE:**



MORE SECURE DIALOGUE





The Version 4 Authentication Dialogue The full Kerberos v4 authentication dialogue is shown here divided into 3 phases.

(1) C → AS $ID_c \| ID_{tgs} \| TS_1$

(2) AS → C $E(K_c, [K_{c,tgs} \| ID_{tgs} \| TS_2 \| Lifetime_2 \| Ticket_{tgs}])$

$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} \| ID_C \| AD_C \| ID_{tgs} \| TS_2 \| Lifetime_2])$

(a) Authentication Service Exchange to obtain ticket-granting ticket

(3) C → TGS $ID_v \| Ticket_{tgs} \| Authenticator_c$

(4) TGS → C $E(K_{c,tgs}, [K_{c,v} \| ID_v \| TS_4 \| Ticket_v])$

$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} \| ID_C \| AD_C \| ID_{tgs} \| TS_2 \| Lifetime_2])$

$Ticket_v = E(K_v, [K_{c,v} \| ID_C \| AD_C \| ID_v \| TS_4 \| Lifetime_4])$

$Authenticator_c = E(K_{c,tgs}, [ID_C \| AD_C \| TS_3])$

(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)

$Ticket_v = E(K_v, [K_{c,v} \| ID_C \| AD_C \| ID_v \| TS_4 \| Lifetime_4])$

$Authenticator_c = E(K_{c,v}, [ID_C \| AD_C \| TS_5])$

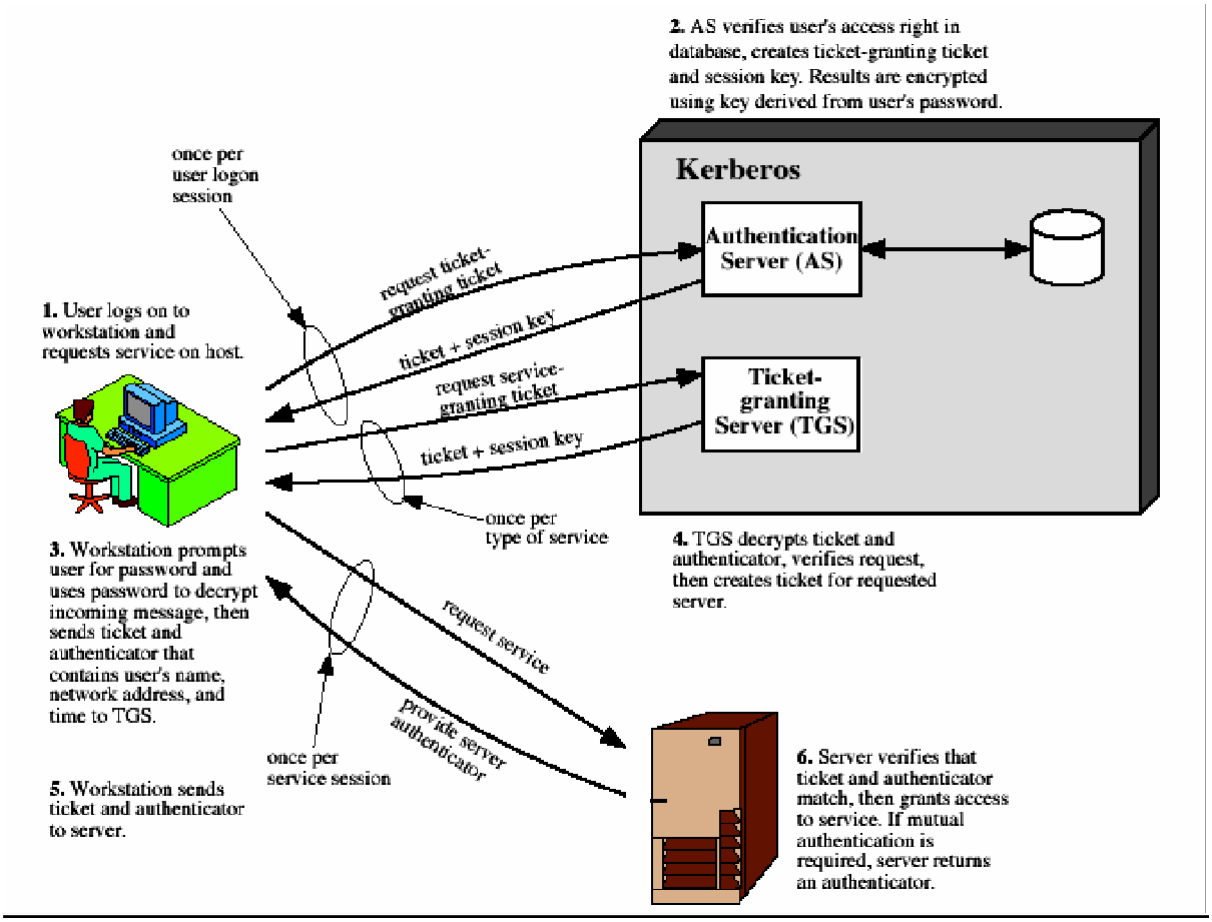
(c) Client/Server Authentication Exchange to obtain service

There is a 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.

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

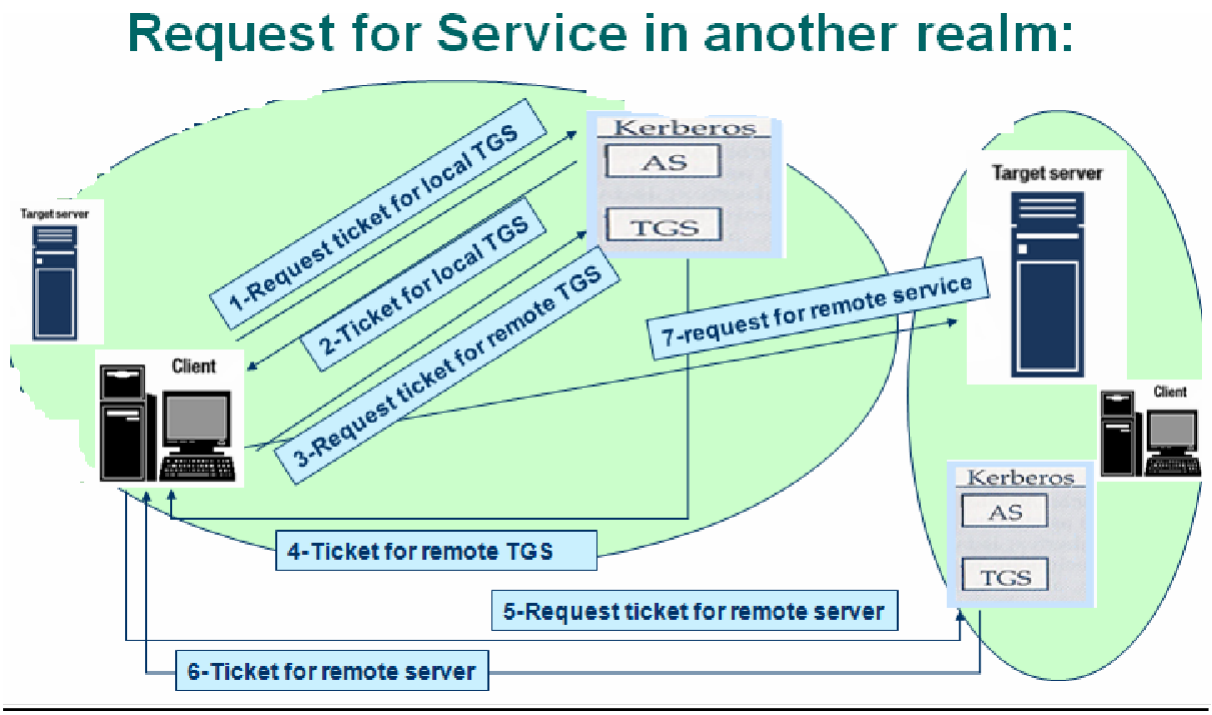
presents this ticket, as shown in message (5), it also sends an authenticator. The server can decrypt the ticket, recover the session key, and decrypt the authenticator. If mutual authentication is required, the server can reply as shown in message (6).

Overview of Kerberos



Kerberos Realms A full-service Kerberos environment consisting of a Kerberos server, a number of clients, and a number of application servers is referred to as a Kerberos realm. A Kerberos realm is a set of managed nodes that share the same Kerberos database, and are part of the same administrative domain. If have multiple realms, their Kerberos servers must share keys and trust each other.

The following figure shows the authentication messages where service is being requested from another domain. The ticket presented to the remote server 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, as each pair of realms need to share akey.



The limitations of Kerberos version-4 are

- categorised into two types: Environmental shortcomings of Version 4:
  - Encryption system dependence: DES
  - Internet protocol dependence
  - Ticket lifetime
  - Authentication forwarding
  - Inter-realm authentication
- Technical deficiencies of Version 4:
  - Double encryption
  - Session Keys
  - Password attack

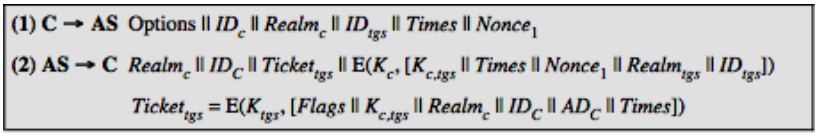
KERBEROS VERSION 5

Kerberos Version 5 is specified in RFC 1510 and provides a number of improvements over version 4 in the areas of environmental shortcomings and technical deficiencies. It includes some new elements such as:

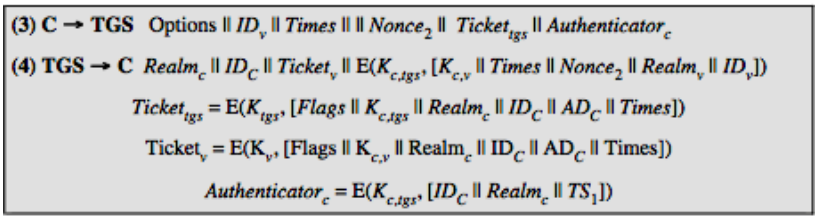
- Realm: Indicates realm of the user
- Options
- Times
  - From: the desired start time for the ticket
  - Till: the requested expiration time
  - Rtime: requested renew-till time
- Nonce: A random value to assure the response is fresh

The basic Kerberos version 5 authentication dialogue is shown here First, consider the

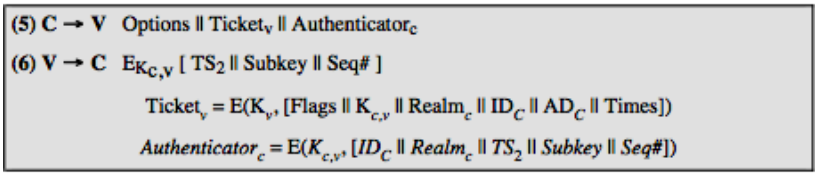
authentication service exchange.



(a) Authentication Service Exchange to obtain ticket-granting ticket



(b) Ticket-Granting Service Exchange to obtain service-granting ticket



(c) Client/Server Authentication Exchange to obtain service

Message (1) is a client request for a ticket-granting ticket. Message (2) returns a ticket- granting ticket, identifying information for the client, and a block encrypted using the encryption key based on the user's password. This block includes the session key to be used between the client and the TGS. Now compare the **ticket-granting service** exchange for versions 4 and 5. See that message (3) for both versions includes an authenticator, a ticket, and the name of the requested service. In addition, version 5 includes requested times and options for the ticket and a nonce, all with functions similar to those of message (1). The authenticator itself is essentially the same as the one used in version 4. Message (4) has the same structure as message (2), returning a ticket plus information needed by the client, the latter encrypted with the session key now shared by the client and the TGS. Finally, for the client/server authentication exchange, several new features appear in version 5, such as a request for mutual authentication. If required, the server responds with message (6) that includes the timestamp from the
authenticator. The flags field included in tickets in version 5 supports
expanded functionality compared to that available in version 4.

Advantages of Kerberos:

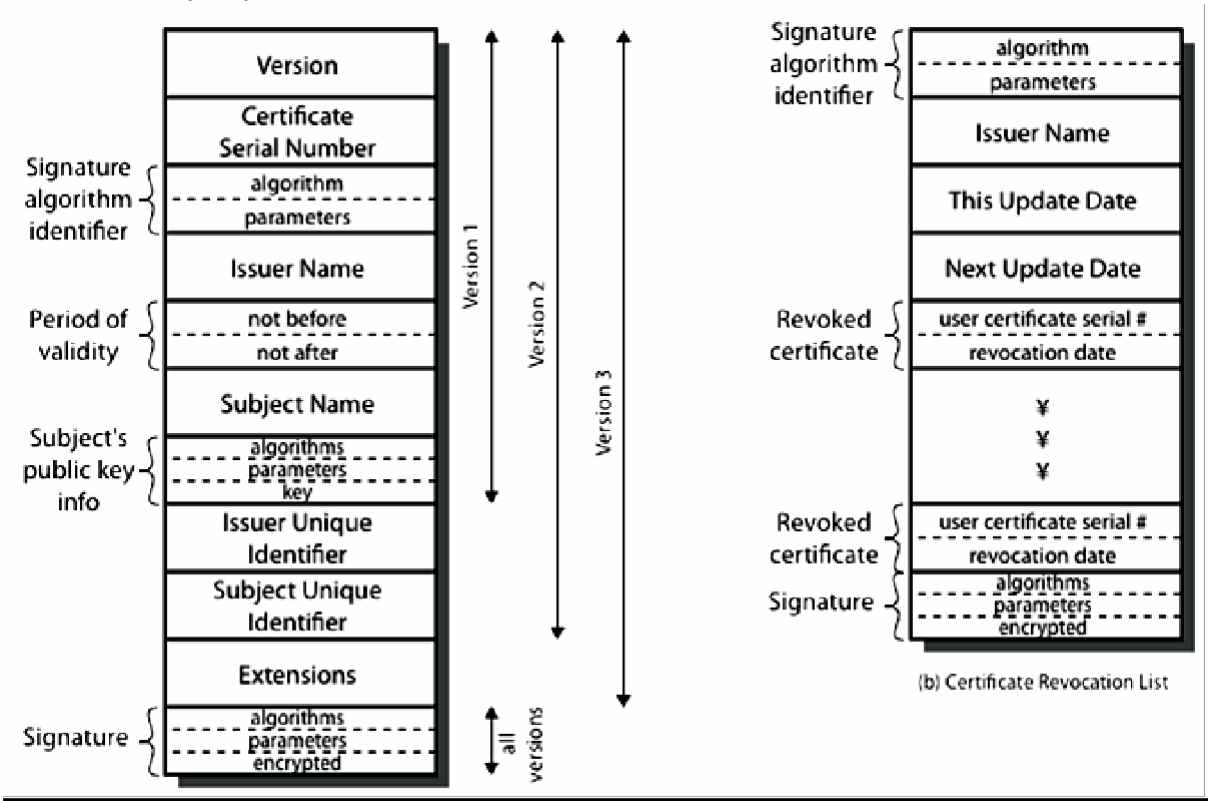
- User's passwords are never sent across the network,
- encrypted or in plain text Secret keys are *only* passed across
- the network in encrypted form
- Client and server systems
- It mutually authenticate limits the duration of their users' authentication.
- Authentications are **reusable** and **durable**
- Kerberos has been scrutinized by many of the top programmers,
- cryptologists and security experts in the industry

X.509 AUTHENTICATION SERVICE

ITU-T recommendation X.509 is part of the X.500 series of recommendations that define a directory service. The directory is, in effect, a server or distributed set of servers that maintains a database of information about users. The information includes a mapping from user name to network address, as well as other attributes and information about the users. X.509 is based on the use of public-key cryptography and digital signatures. The heart of the X.509 scheme is the public-key certificate associated with each user. These user certificates are assumed to be created by some trusted certification authority (CA) and placed in the directory by the CA or by the user. The directory server itself is not responsible for the creation of public keys or for the certification function; it merely provides an easily accessible location for users to obtain certificates.

The general format of a certificate is shown above, which includes the following elements:

- version 1, 2, or 3
- serial number (unique within CA)
- identifying certificate signature algorithm identifier
- issuer X.500 name (CA)
- period of validity (from - to dates)
- subject X.500 name (name of owner)
- subject public-key info (algorithm, parameters, key) issuer unique identifier (v2+)



- subject

unique
identifier
(v2+)
extension
fields (v3)
- signature (of hash of all fields in certificate)

The standard uses the following notation to define a certificate:

CA<<A>> = CA {V, SN, AI, CA, TA, A, Ap}

Where Y<<X>>= the certificate of user X issued by certification authority Y
Y {I} == the signing of I by Y. It consists of I with an encrypted
hash code appended User certificates generated by a CA have the
following characteristics:

- Any user with CA's public key can verify the user public key that was certified No party other than the CA can modify the
- certificate without being detected because they cannot be
- forged, certificates can be placed in a public directory

Scenario: Obtaining a User Certificate If both users share a common CA then they are assumed to know its public key. Otherwise CA's must form a hierarchy and use certificates linking members of hierarchy to validate other CA's. Each CA has certificates for clients (forward) and parent (backward). Each client trusts parents certificates. It

enables verification of any certificate from one CA by users of all other CAs in hierarchy. A has obtained a certificate from the CA X1. B has obtained a certificate from the CA X2. A can read the B's certificate but cannot verify it. In order to solve the problem ,the Solution: **X1<<X2> X2<<B>>.** A obtain the certificate of X2 signed by X1 from directory.

obt

- ain X2's public key. A goes back to directory and obtain the certificate of B signed by X2. obtain B's public key securely. The directory entry for each CA includes two types of certificates: Forward certificates: Certificates of X generated by other CAs Reverse

certificates: Certificates generated by X that are the certificates of other CAs

X.509 CA Hierarchy

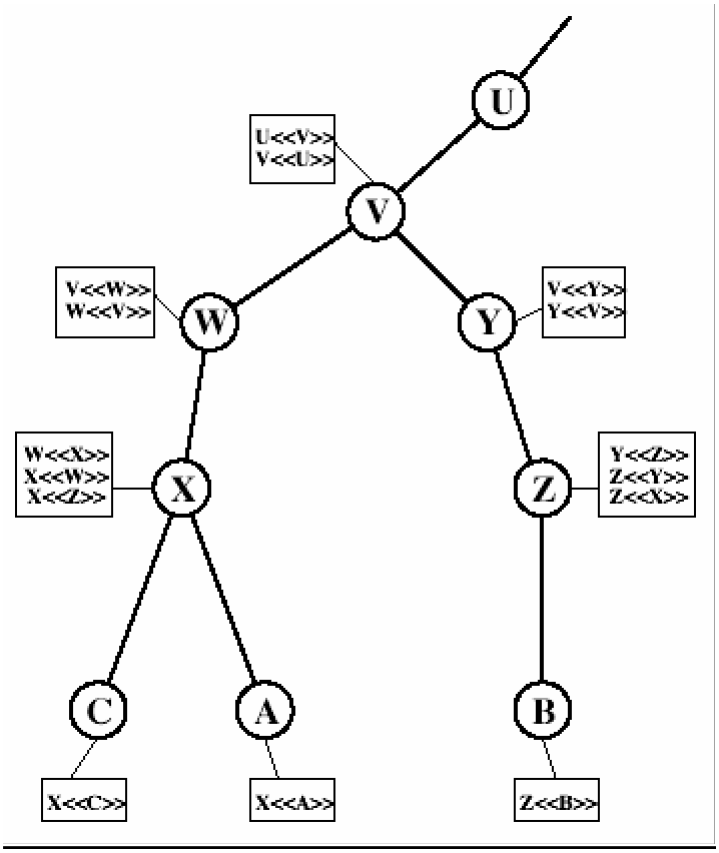
A acquires B certificate using chain:

X<<W>>W<<V>>V<<Y>>Y<<Z>> Z<<B>>

B acquires A certificate using chain:

Z<<Y>>Y<<V>>V<<W>>W<<X>> X<<A>>

Revocation of Certificates Typically, a new certificate is issued



just before the expiration of the old one. In addition, it may be desirable on occasion to revoke a certificate before it expires, for one of the following reasons:

- The user's private key is assumed to be compromised. The user is no longer certified by this CA.
- The CA's certificate is assumed to be compromised.

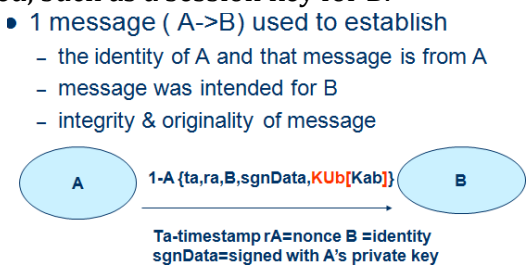
Each CA must maintain a list consisting of all revoked but not expired certificates issued by that CA, including both those issued to users and to other CAs. These lists should also be posted on the directory. Each **certificate revocation list (CRL) posted** to the directory is signed by the issuer and includes the issuer's name, the date the list was created, the date the next CRL is scheduled to be issued, and an entry for each revoked certificate. Each entry consists of the serial number of a certificate and revocation date for that certificate. Because serial numbers are unique within a CA, the serial number is sufficient to identify the certificate.

AUTHENTICATION PROCEDURES

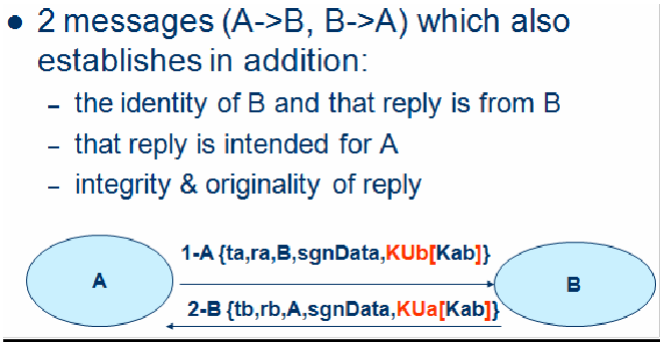
X.509 also includes three alternative authentication procedures that are intended for use across a variety of applications. All these procedures make use of public-key signatures. It is assumed that the two parties know each other's public key, either by obtaining each other's certificates from the directory or because the certificate is included in the initial message from each side. 1. One-Way Authentication: One way authentication involves a single transfer of information from one user (A) to another (B), and establishes the details shown above.

Note that only the identity of the initiating entity is verified in this process, not that of the

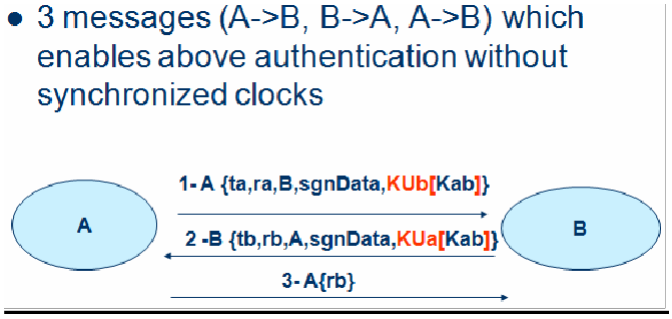
responding entity. At a minimum, the message includes a timestamp ,a nonce, and the identity of B and is signed with A's private key. The message may also include information to be conveyed, such as a session key for B.



Two-Way Authentication: Two-way authentication thus permits both parties in a communication to verify the identity of the other, thus additionally establishing the above details. The reply message includes the nonce from A, to validate the reply. It also includes a timestamp and nonce generated by B, and possible additional information for A.



Three-Way Authentication: Three-Way Authentication includes a final message from A to B, which contains a signed copy of the nonce, so that timestamps need not be checked, for use when synchronized clocks are not available.



BIOMETRIC AUTHENTICATION

Biometric authentication is a type of system that relies on the unique biological characteristics of individuals to verify identity for secure access to electronic systems.

Biometric verification is considered a subset of biometric authentication. The biometric technologies involved are based on the ways in which individuals can be uniquely identified through one or more distinguishing biological traits, such as fingerprints, hand geometry, earlobe geometry, retina and iris patterns, voice waves, keystroke dynamics, DNA and signatures. Biometric authentication is the application of that proof of identity as part of a process validating a user for access to a

system. Biometric technologies are used to secure a wide range of electronic communications, including enterprise security, online commerce and banking -- even just logging in to a computer or smartphone.
Biometric authentication systems compare the current biometric data capture to stored, confirmed authentic data in a database. If both samples of the biometric data match,
authentication is confirmed and access is granted. The process is sometimes part of amultifactor authentication system. For example, a smartphone user might log on with his personal identification number (PIN) and then provide an iris scan to complete the authentication process.

Types of biometric authentication technologies:
Retina scan

Iris recognition is used to identify individuals based on unique patterns within the ring- shaped region surrounding the pupil of the eye.

Fingerscanning, the digital version of the ink-and-paper fingerprinting process, works with details in the pattern of raised areas and branches in a human finger image.

Finger vein ID is based on the unique vascular pattern in an individual's finger.

Facial recognition systems work with numeric codes called faceprints, which identify 80 nodal points on a human face.

Voice identification systems rely on characteristics created by the shape of the speaker's mouth and throat, rather than more variable conditions.

Once seen mostly in spy movies (where it might be used to protect access to a top-secret military lab, for example), biometric authentication is becoming relatively commonplace. In addition to the security provided by hard-to-fake individual biological traits, the acceptance of biometric verification has also been driven by convenience: One can't easily forget or lose ones biometrics.

The history of biometric verification:
The oldest known use of biometric verification is fingerprinting. Thumbprints made on clay seals were used as a means of unique identification as far back as ancient China.
Modern biometric verification has become almost instantaneous, and is increasingly accurate with the advent of computerized databases and the digitization of analog data.
The market for biometrics products is still too fractured to name specific top providers. The physical characteristics of the biometrics products available today vary from the mundane, such as fingerprinting, to the esoteric, like typing speeds and electrophysiological signals.

Until recently, biometrics was typically used at a physical security level – protecting facilities at military bases or impenetrable bank vaults, for example. But, because single- factor authentication methods are easy to break, companies have started looking to
two-factor solutions, like biometrics.

However, the following five fundamental barriers may limit the growth of biometric authentication:

1. Biometrics can be complicated and costly to deploy. All biometric deployments require installation of their own hardware and application servers.
2. The market is still fractured. Should you buy a fingerprint reader, a voice recognition system or an iris scanner? Since each product differs greatly in its approach and installation, it is difficult to compare them during a typical company bid process.
3. Biometric data is like any other data. It sits on servers, which are bait forhackers if not properly hardened and secured. Therefore, when reviewing any biometric product, make sure it transmits data securely, meaningencrypted, from the biometric reader back to the authenticating server. And, make sure the authenticating server has been hardened, patched and protected.
4. Biometric readers are prone to errors. Fingerprints can smudge, faces and voices can be changed and all of them can be misread, blocking a legitimate user, or permitting access to an unauthorized or malicious user.
5. Difficulties with user acceptance. Properly trained employees may be willing to use biometrics devices, but customers, like those logging on to your Web site, may be more reluctant to use – or worse, forced to purchase – a device that's difficult to use or makes doing business, such as banking, on your site, a hassle instead of a convenience. And both your employees and customers may be squeamish about exposing their eyes to devices like iris scanners, even if they appear harmless.

Despite these issues, biometrics is slowly gaining acceptance for two-factor authentication purposes. The products are getting better, lighter and easier to use. Error rates are going down, and fingerprint readers installed on tokens and laptops are getting smaller and less intrusive. And, like the rest of the security product industry, vendors will eventually merge and consolidate, uniting a fractured market, which will make it easier to choose a product that suits your business needs.

***Email Privacy**: Pretty Good Privacy (PGP) and S/MIME. **IP Security:** IP Security Overview, IP Security Architecture, Authentication Header, Encapsulating Security Payload, Combining Security Associations and Key Management.*

PRETTY GOOD PRIVACY

In virtually all distributed environments, electronic mail is the most heavily used network-based application. But current email services are roughly like "postcards", anyone who wants could pick it up and have a look as it's in transit or sitting in the recipients mailbox. PGP provides a confidentiality and authentication service that can be used for electronic mail and file storage applications. With the explosively growing reliance on electronic mail for every conceivable purpose, there grows a demand for authentication and confidentiality services. The Pretty Good Privacy (PGP) secure email program, is a remarkable phenomenon, has grown explosively and is now widely used. Largely the effort of a single person, Phil Zimmermann, who selected the best available crypto algorithms to use & integrated them into a single program, PGP provides a confidentiality and authentication service that can be used for electronic mail and file storage applications. It is independent of government organizations and runs on a wide range of systems, in both free & commercial versions. *There are **five** important services in PGP*

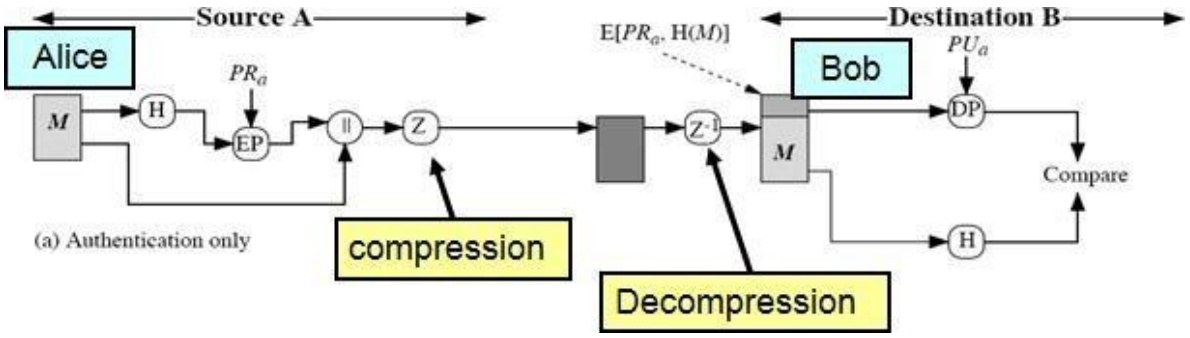
- *Authentication (Sign/Verify)*
- *Confidentiality*
- *(Encryption/Decryption) Compression*
- *Email compatibility*
- *Segmentation and Reassembly*

The last three are **transparent** to the user

PGP Notations:

Ks	=session key used in symmetric encryption scheme
PRa	=private key of user A, used in public-key encryption scheme
PUa	=public key of user A, used in public-key encryption scheme
EP	= public-key encryption
DP	= public-key decryption
EC	= symmetric encryption
DC	= symmetric decryption
H	= hash function
||	= concatenation
Z	=compression using ZIP algorithm
R64	= conversion to radix 64 ASCII format

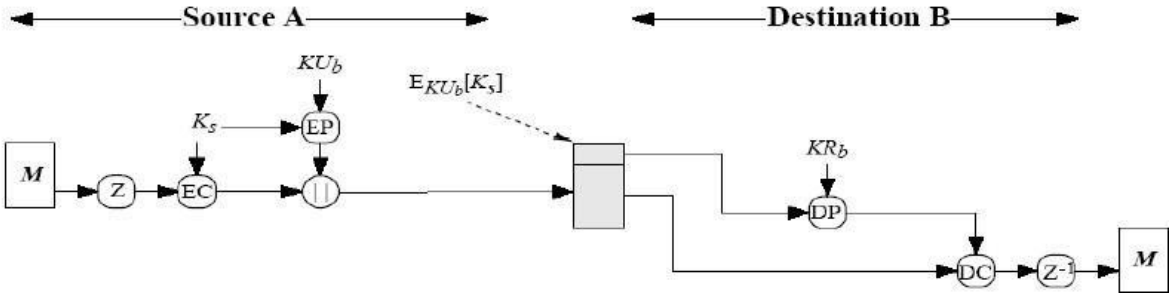
PGP Operation- Authentication



1. sender creates message
2. use SHA-1 to generate 160-bit hash of message
3. signed hash with RSA using sender's private key, and is attached to message
4. receiver uses RSA with sender's public key to decrypt and recover hash code
5. receiver verifies received message using hash of it and compares with decrypted hash code

PGP Operation- Confidentiality

PGP Operation- Confidentiality



Sender:

1. Generates message and a random number (session key) only for this message
2. Encrypts message with the session key using AES, 3DES, IDEA or CAST-128
3. Encrypts session key itself with recipient's public key using RSA
4. Attaches it to message

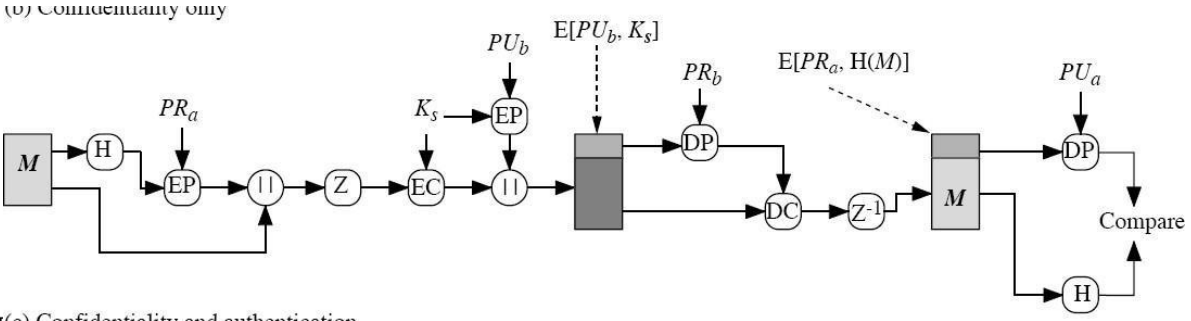
Receiver:

1. Recovers session key by decrypting using his private key
2. Decrypts message using the session key

Confidentiality service provides no assurance to the receiver as to the identity of sender (i.e. no authentication). Only provides confidentiality for sender that only the recipient can read the message (and no one else) can use both services on same message o create signature & attach to message o encrypt both message & signature o attach RSA/ElGamal encrypted session key

o *is called* ***authenticated confidentiality***

PGP Operation – Confidentiality & Authentication



(c) Confidentiality and authentication

PGP Operation – Compression

As a default, PGP compresses the message after applying the signature but before encryption. This has the benefit of saving space both for e-mail transmission and for file storage. The placement of the compression algorithm, indicated by Z for compression and Z^{-1} for decompression is critical. The compression algorithm used is ZIP.

The signature is generated before compression for two reasons:

1. so that one can store only the uncompressed message together with signature for later verification

2. Applying the hash function and signature after compression would constrain all PGP implementations to the same version of the compression algorithm as the PGP compression algorithm is not deterministic

Message encryption is applied after compression to strengthen cryptographic security. Because the compressed message has less redundancy than the original plaintext, cryptanalysis is more difficult.

PGP Operation – Email Compatibility

When PGP is used, at least part of the block to be transmitted is encrypted, and thus consists of a stream of arbitrary 8-bit octets. However many electronic mail systems only permit the use of ASCII text. To accommodate this restriction, PGP provides the service

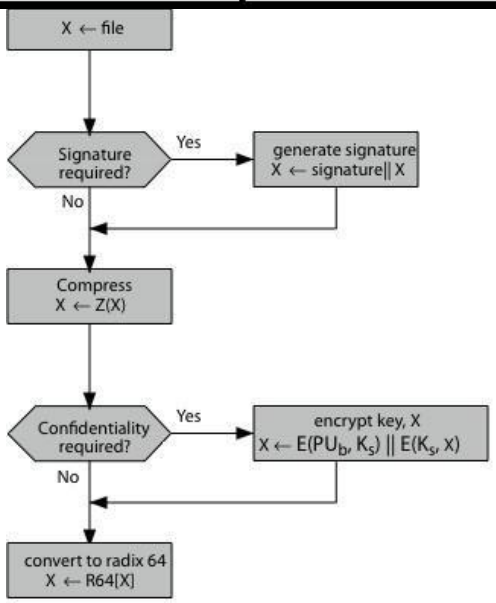
of converting the raw 8-bit binary stream to a stream of printable ASCII characters. It uses radix-64 conversion, in which each group of three octets of binary data is mapped into four ASCII characters. This format also appends a CRC to detect transmission errors. The use of radix 64 expands a message by 33%, but still an overall compression of about one- third can be achieved.

PGP Operation - Segmentation/Reassembly

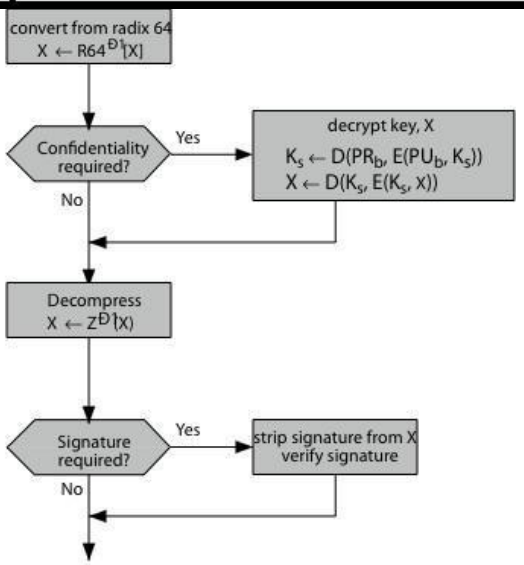
E-mail facilities often are restricted to a maximum message length. For example, many of the facilities accessible through the Internet impose a maximum length of 50,000 octets. Any message longer than that must be broken up into smaller segments, each of which is mailed separately. To accommodate this restriction, PGP automatically subdivides a message that is too large into segments that are small enough to send via e-mail. The segmentation is done after all of the other processing, including the radix-64 conversion. Thus, the session key component and signature component appear only once, at the beginning of the first segment. Reassembly at the receiving end is required before verifying signature or decryption

PGP Operations – Summary

Function	Algorithms Used	Description
Digital signature	DSS/SHA or RSA/SHA	A hash code of a message is created using SHA-1. This message digest is encrypted using DSS or RSA with the sender's private key, and included with the message.
Message encryption	CAST or IDEA or Three-key Triple DES with Diffie-Hellman or RSA	A message is encrypted using CAST-128 or IDEA or 3DES with a one-time session key generated by the sender. The session key is encrypted using Diffie-Hellman or RSA with the recipient's public key, and included with the message.
Compression	ZIP	A message may be compressed, for storage or transmission, using ZIP.
Email compatibility	Radix 64 conversion	To provide transparency for email applications, an encrypted message may be converted to an ASCII string using radix 64 conversion.
Segmentation	—	To accommodate maximum message size limitations, PGP performs segmentation and reassembly.



(a) Generic Transmission Diagram (from A)

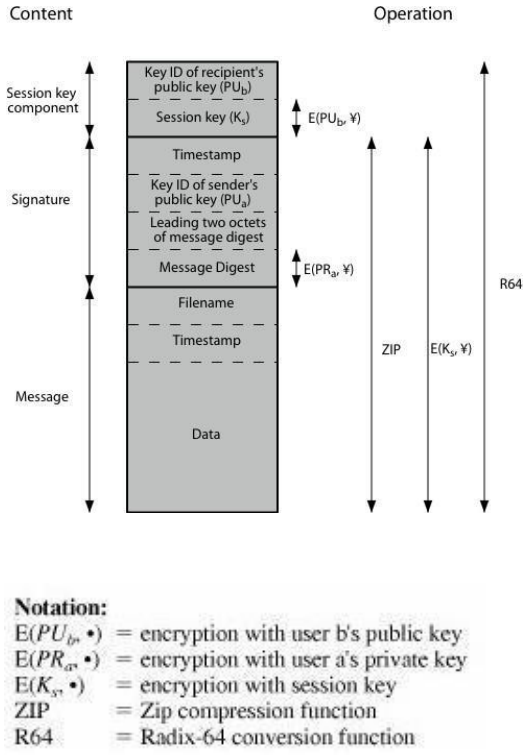


(b) Generic Reception Diagram (to B)

PGP Message Format

A message consists of three components: the message component, a signature (optional), and a session key component (optional). The *message component* includes the actual data to be stored or transmitted, as well as a filename and a timestamp that specifies the time of creation. The *signature component* includes the following:

- *Timestamp*: The time at which the signature was made.
- *Message digest*: The 160-bit SHA-1 digest, encrypted with the sender's private signature key.
- *Leading two octets of message digest*: To enable the recipient to determine if the correct public key was used to decrypt the message digest for authentication, by comparing this plaintext copy of the first two octets with the first two octets of the decrypted digest. These octets also serve as a 16-bit frame check sequence for the message.
- *Key ID of sender's public key*: Identifies the public key that should be used to decrypt the message digest and, hence, identifies the private key that was used to encrypt the message digest



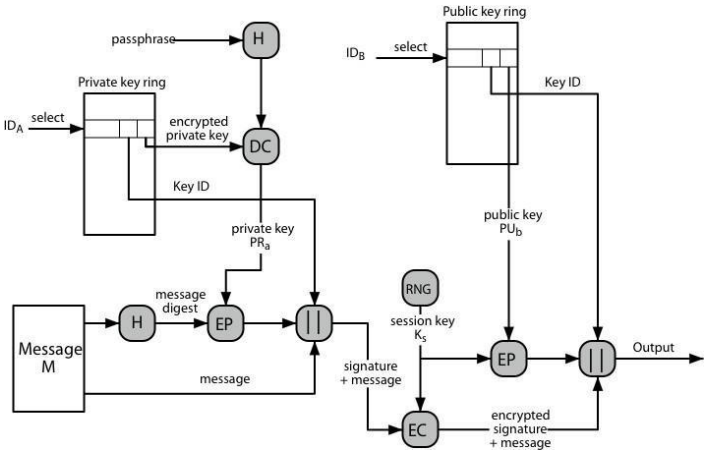
Notation:
E(PU_b, •) = encryption with user b's public key
E(PR_a, •) = encryption with user a's private key
E(K_s, •) = encryption with session key
ZIP = Zip compression function
R64 = Radix-64 conversion function

The *session key component* includes the session key and the identifier of the recipient's public key that was used by the sender to encrypt the session key. The entire block is usually encoded with radix-64 encoding.

PGP Message Transmission and Reception

Message transmission

The following figure shows the steps during message transmission assuming that the message is to be both signed and encrypted.



The sending PGP entity performs the following steps:

Signing the message

a. PGP retrieves the sender's private key from the private-key ring using your_userid as an index. If your_userid was not provided in the command, the first private key on the ring is retrieved.

b. PGP prompts the user for the passphrase to recover the unencrypted private key.

c. The signature component of the message is constructed

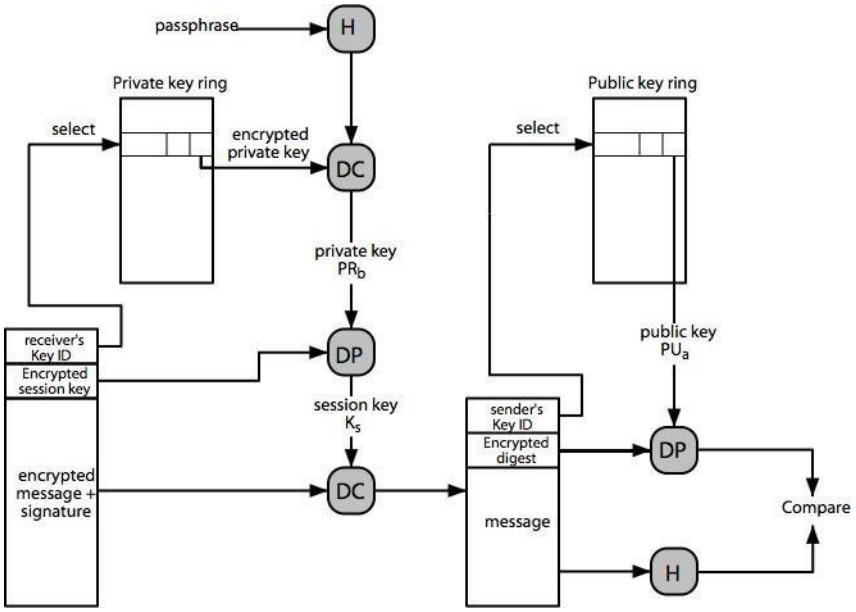
Encrypting the message

a. PGP generates a session key and encrypts the message.

b. PGP retrieves the recipient's public key from the public-key ring using her_userid as an index.

c. The session key component of the message is constructed.

Message Reception



The receiving PGP entity performs the following steps:

Decrypting the message

a. PGP retrieves the receiver's private key from the private-key ring, using the Key ID

field in the session key component of the message as an index.
b. PGP prompts the user for the passphrase to recover the unencrypted private key.
c. PGP then recovers the session key and decrypts the message.

Authenticating the message

a. PGP retrieves the sender's public key from the public-key ring, using the Key ID field in the signature key component of the message as an index.
b. PGP recovers the transmitted message digest.
c. PGP computes the message digest for the received message and compares it to the transmitted message digest to authenticate.

S/MIME

S/MIME (Secure/Multipurpose Internet Mail Extension) is a security enhancement to the MIME Internet e-mail format standard, which in turn provided support for varying content types and multi-part messages over the text only support in the original Internet RFC822 email standard. MIME allows encoding of binary data to textual form for transport over traditional RFC822 email systems. S/MIME is defined in a number of documents, most importantly RFCs 3369, 3370, 3850 and 3851 and S/MIME support is now included in many modern mail agents.

RFC 822

RFC 822 defines a format for text messages that are sent using electronic mail and it has been the standard for Internet-based text mail message. The overall structure of a message that conforms to RFC 822 is very simple. A message consists of some number of header lines (the header) followed by unrestricted text (the body). The header is separated from the body by a blank line. A header line usually consists of a keyword, followed by a colon, followed by the keyword's arguments; the format allows a long line to be broken up into several lines. The most frequently used keywords are *From, To*, *Subject*, and *Date*.

Multipurpose Internet Mail Extensions

MIME is an extension to the RFC 822 framework that is intended to address some of the problems and limitations of the use of SMTP (Simple Mail Transfer Protocol) or some other mail transfer protocol and RFC 822 for electronic mail. **Problems with RFC 822 and SMTP**

• Executable files or other binary objects must be converted into ASCII. Various schemes exist (e.g., Unix UUencode), but a standard is needed
• Text data that includes special characters (e.g., Hungarian text) cannot be transmitted as SMTP is limited to 7-bit ASCII
• Some servers reject mail messages over a certain size
• Some common problems exist with the SMTP implementations which do not adhere completely to the SMTP standards defined in RFC 821. They are:

- delete, add, or reorder CR and LF characters truncate or wrap
- lines longer than 76 characters remove trailing white space
- (tabs and spaces) pad lines in a message to the same length
- convert tab characters into multiple spaces
-

MIME is intended to resolve these problems in a manner that is compatible with existing RFC 822 implementations and the specification is provided in RFC's 2045

through 2049.

The MIME specification includes the following elements:

1. Five new message header fields are defined, which provide information about the body of the message.
2. A number of content formats are defined, thus standardizing representations that support multimedia electronic mail.
3. Transfer encodings are defined that protect the content from alteration by the mail system.

MIME - New header fields The five header fields defined in MIME are as follows:

• *MIME-Version:* Must have the parameter value 1.0. This field indicates that the message conforms to RFCs 2045 and 2046.

• *Content-Type*: Describes the data contained in the body with sufficient detail that the receiving user agent can pick an appropriate agent or mechanism to represent the data to the user or otherwise deal with the data in an appropriate manner.

• *Content-Transfer-Encoding*: Indicates the type of transformation that has been used to represent the body of the message in a way that is acceptable for mail transport.

• Content-ID: Used to identify MIME entities uniquely in multiple contexts.

• *Content-Description*: A text description of the object with the body; this is useful when the object is not readable (e.g., audio data).

MIME Content Types The bulk of the MIME specification is concerned with the definition of a variety of content types. There are seven different major types of content and a total of 15 subtypes. In general, a content type declares the general type of data, and the subtype specifies a particular format for that type of data. For the text type of body, the primary subtype is plain text, which is simply a string of ASCII characters or ISO 8859 characters. The enriched subtype allows greater formatting flexibility. The multipart type indicates that the body contains multiple, independent parts. The Content-Type header field includes a parameter called boundary that defines the delimiter between body parts. This boundary should not appear in any parts of the message. Each boundary starts on a new line and consists of two hyphens followed by the boundary value. The final boundary, which indicates the end of the last part, also has a suffix of two hyphens. Within each part, there may be an optional ordinary MIME header. There are four subtypes of the multipart type, all of which have the same overall syntax.

Message	rfc822	The body is itself an encapsulated message that conforms to RFC 822.
	Partial	Used to allow fragmentation of large mail items, in a way that is transparent to the recipient.
	External-body	Contains a pointer to an object that exists elsewhere.
Image	jpeg	The image is in JPEG format, JFIF encoding.
	gif	The image is in GIF format.
Video	mpeg	MPEG format.
Audio	Basic	Single-channel 8-bit ISDN mu-law encoding at a sample rate of 8 kHz.
Application	PostScript	Adobe Postscript.
	octet-stream	General binary data consisting of 8-bit bytes.
		each part is message/rfc822.

The message type provides a number of important capabilities in MIME. The message/rfc822 subtype indicates that the body is an entire message, including

header
and body. Despite the name of this subtype, the encapsulated message may be not only a simple RFC 822 message, but also any MIME message. The message/partial subtype enables fragmentation of a large message into a number of parts, which must be reassembled at the destination. For this subtype, three parameters are specified in the Content-Type: Message/Partial field: an id common to all fragments of the same message, a sequence number unique to each fragment, and the total number of fragments. The message/external-body subtype indicates that the actual data to be conveyed in this message are not contained in the body. Instead, the body contains the information needed to access the data. The application type refers to other kinds of data, typically either uninterpreted binary data or information to be processed by a mail-basedapplication.

MIME Transfer Encodings The other major component of the MIME specification, in addition to content type specification, is a definition of transfer encodings for message bodies. The objective is to provide reliable delivery across the largest range of environments.

MIME Transfer Encodings

7bit	The data are all represented by short lines of ASCII characters.
8bit	The lines are short, but there may be non-ASCII characters (octets with the high-order bit set).
binary	Not only may non-ASCII characters be present but the lines are not necessarily short enough for SMTP transport.
quoted-printable	Encodes the data in such a way that if the data being encoded are mostly ASCII text, the encoded form of the data remains largely recognizable by humans.
base64	Encodes data by mapping 6-bit blocks of input to 8-bit blocks of output, all of which are printable ASCII characters.
x-token	A named nonstandard encoding.

The MIME standard defines two methods of encoding data. The Content-Transfer-Encoding field can actually take on six values. Three of these values (7bit, 8bit, and binary) indicate that no encoding has been done but provide some information about the nature of the data. Another Content-Transfer-Encoding value is x-token, which indicates that some other encoding scheme is used, for which a name is to be supplied. The two actual encoding schemes defined are quoted- printable and base64. Two schemes are defined to provide a choice between a transfer technique that is essentially human
readable and one that is safe for all types of data in a way that is reasonably compact. The quoted-printable transfer encoding is useful when the data consists largely of octets that correspond to printable ASCII characters. In essence, it represents nonsafe characters by the hexadecimal representation of their code and introduces reversible (soft) line breaks to limit message lines to 76 characters. The base64 transfer encoding, also known as radix-64 encoding, is a common one for encoding arbitrary binary data in such a way as to be invulnerable to the processing by mail transport programs.

Canonical Form

An important concept in MIME and S/MIME is that of canonical form. Canonical form is a format, appropriate to the content type, that is standardized for use between systems. This is in contrast to native form, which is a format that may be peculiar to a particular system.

S/MIME Functionality

S/MIME has a very similar functionality to PGP. Both offer the ability to sign and/or

Native Form	The body to be transmitted is created in the system's native format. The native character set is used and, where appropriate, local end-of-line conventions are used as well. The body may be a UNIX-style text file, or a Sun raster image, or a VMS indexed file, or audio data in a system-dependent format stored only in memory, or anything else that corresponds to the local model for the representation of some form of information. Fundamentally, the data is created in the "native" form that corresponds to the type specified by the media type.
Canonical Form	The entire body, including "out-of-band" information such as record lengths and possibly file attribute information, is converted to a universal canonical form. The specific media type of the body as well as its associated attributes dictate the nature of the canonical form that is used. Conversion to the proper canonical form may involve character set conversion, transformation of audio data, compression, or various other operations specific to the various media types. If character set conversion is involved, however, care must be taken to understand the semanties of the media type, which may have strong implications for any character set conversion (e.g. with regard to syntactically meaningful characters in a text subtype other than "plain").

encrypt messages.

Functions

S/MIME provides the following functions:

• **Enveloped data**: This consists of encrypted content of any type and encrypted-content encryption keys for one or more recipients.

• **Signed data**: A digital signature is formed by taking the message digest of the content to be signed and then encrypting that with the private key of the signer. The content plus signature are then encoded using base64 encoding. A signed data message can only be viewed by a recipient with S/MIME capability.

• **Clear-signed data**: As with signed data, a digital signature of the content is formed. However, in this case, only the digital signature is encoded using base64. As a result, recipients without S/MIME capability can view the message content, although they cannot verify the signature.

• **Signed and enveloped data**: Signed-only and encrypted-only entities may be nested, so that encrypted data may be signed and signed data or clear-signed data may be encrypted.

IP SECURITY OVERVIEW

Definition: Internet Protocol security (IPSec) is a framework of open standards for protecting communications over Internet Protocol (IP) networks through the use of cryptographic security services. IPSec supports network-level peer authentication, data origin authentication, data integrity, data confidentiality (encryption), and replay protection.

Need for IPSec

In Computer Emergency Response Team (CERT)'s 2001 annual report it listed 52,000 security incidents in which most serious types of attacks included **IP spoofing**, in which intruders create packets with false IP addresses and exploit applications that use authentication based on IP and various forms of **eavesdropping and packet sniffing**, in which attackers read transmitted information, including logon information and database contents. In response to these issues, the IAB included authentication and encryption as necessary security features in the next-generation IP i.e. IPv6.

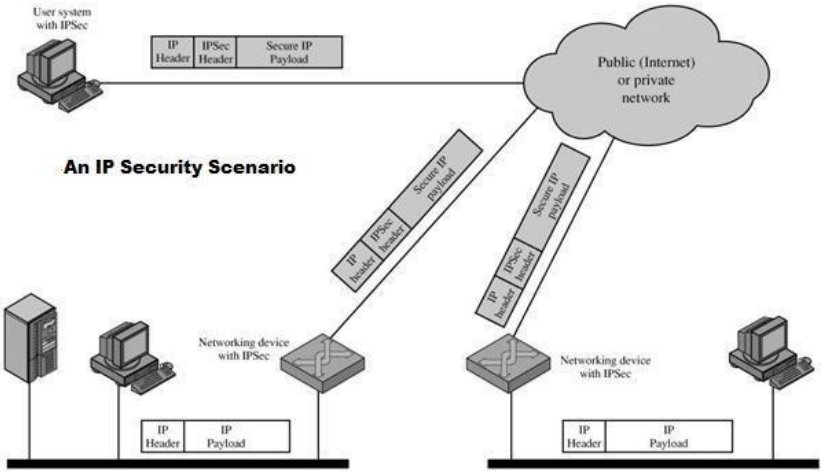
Applications of IPSec

IPSec provides the capability to secure communications across a LAN, across private and public wide area networks (WAN's), and across the Internet.

- ***Secure branch office connectivity over the Internet***: A company can build a secure virtual private network over the Internet or over a public WAN. This enables a business to rely heavily on the Internet and reduce its need for private networks, saving costs and network management overhead.
- ***Secure remote access over the Internet***: An end user whose system is equipped with IP security protocols can make a local call to an Internet service provider (ISP) and gain secure access to a company network. This reduces the cost of toll charges for travelling employees and telecommuters.
- ***Establishing extranet and intranet connectivity with partners***: IPSec can be used to secure communication with other organizations, ensuring authentication and confidentiality and providing a key exchange mechanism.
- ***Enhancing electronic commerce security***: Even though some Web and electronic commerce applications have built-in security protocols, the use of IPSec enhances that security.

The principal feature of IPSec enabling it to support varied applications is that it can encrypt and/or authenticate all traffic at IP level. Thus, all distributed applications, including remote logon, client/server, e-mail, file transfer, Web access, and so on, can be secured.

The following figure shows a typical scenario of IPSec usage. An organization maintains LANs at dispersed locations. Non secure IP traffic is conducted on each LAN.



The IPSec protocols operate in networking devices, such as a router or firewall that connect each LAN to the outside world. The IPSec networking device will typically encrypt and compress all traffic going into the WAN, and decrypt and decompress traffic coming from the WAN; these operations are transparent to workstations and servers on the LAN. Secure transmission is also possible with individual users who dial into the WAN. Such user workstations must implement the IPSec protocols to provide security.

Benefits of IPSec

The benefits of IPSec are listed below:

- IPSec in a firewall/router provides strong security to all traffic crossing the perimeter
- IPSec in a firewall is resistant to bypass
- IPSec is below transport layer(TCP,UDP), hence transparent to applications
- IPSec can be transparent to end users
- IPSec can provide security for individual users if needed (useful for offsite workers and setting up a secure virtual subnetwork for sensitive applications)

Routing Applications

IPSec also plays a vital role in the routing architecture required for internetworking. It assures that:

- router advertisements come from authorized routers
- neighbor advertisements come from authorized routers
- redirect messages come from the router to which initial packet was sent
- A routing update is not forged

IP Security Architecture

To understand IP Security architecture, we examine IPSec documents first and then move on to IPSec services and Security Associations.

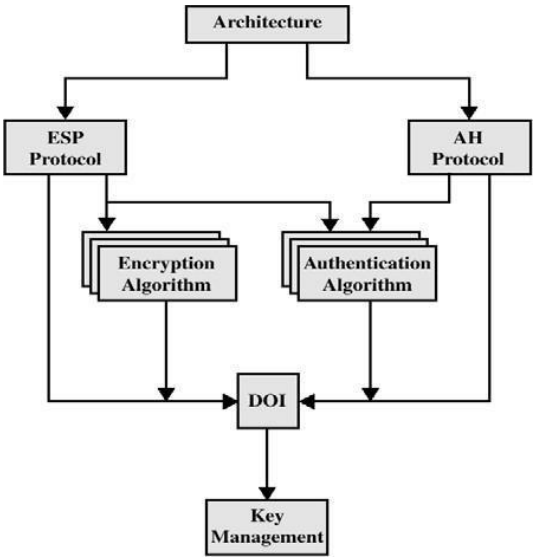
IPSec Documents

The IPSec specification consists of numerous documents. The most important of these, issued in November of 1998, are RFCs 2401, 2402, 2406, and 2408:

- RFC 2401: An overview of a security architecture
- RFC 2402: Description of a packet authentication extension to IPv4 and IPv6
- RFC 2406: Description of a packet encryption extension to IPv4 and IPv6
- RFC 2408: Specification of key management capabilities

Support for these features is mandatory for IPv6 and optional for IPv4. In both cases, the security features are implemented as extension headers that follow the main IP header.

The extension header for authentication is known as the Authentication header; that for encryption is known as the Encapsulating Security Payload (ESP) header. In addition to these four RFCs, a number of additional drafts have been published by the IP Security Protocol Working Group set up by the IETF. The documents are divided into seven groups, as depicted in following figure:



• **Architecture**: Covers the general concepts, security requirements, definitions, and mechanisms defining IPSec technology

• **Encapsulating Security Payload (ESP):** Covers the packet format and general issues related to the use of the ESP for packet encryption and, optionally, authentication.

• **Authentication Header (AH):** Covers the packet format and general issues related to the use of AH for packet authentication.

• **Encryption Algorithm**: A set of documents that describe how various encryption algorithms are used for ESP.

• **Authentication Algorithm:** A set of documents that describe how various authentication algorithms are used for AH and for the authentication option of ESP.

• **Key Management**: Documents that describe key management schemes.

• **Domain of Interpretation (DOI):** Contains values needed for the other documents to relate to each other. These include identifiers for approved encryption and authentication algorithms, as well as operational parameters such as key lifetime.

IPSec Services

IPSec architecture makes use of two major protocols (i.e., Authentication Header and ESP protocols) for providing security at IP level. This facilitates the system to beforehand choose an algorithm to be implemented, security protocols needed and any cryptographic keys required to provide requested services. The IPSec services are as follows:

Connectionless Integrity:- Data integrity service is provided by IPSec via AH which prevents the data from being altered during transmission.

Data Origin Authentication:- This IPSec service prevents the occurrence of

replay attacks, address spoofing etc., which can be fatal

- **Access Control**:- The cryptographic keys are distributed and the traffic flow is controlled in both AH and ESP protocols, which is done to accomplish access control over the data transmission.
- **Confidentiality**:- Confidentiality on the data packet is obtained by using an encryption technique in which all the data packets are transformed into ciphertext packets which are unreadable and difficult to understand.
- **Limited Traffic Flow Confidentiality**:- This facility or service provided by IPSec ensures that the confidentiality is maintained on the number of packets transferred or received. This can be done using padding in ESP.
- **Replay packets Rejection**:- The duplicate or replay packets are identified and discarded using the sequence number field in both AH and ESP.

	AH	ESP (encryption only)	ESP (encryption plus authentication)
Access control	✔	✔	✔
Connectionless integrity	✔		✔
Data origin authentication	✔		✔
Rejection of replayed packets	✔	✔	✔
Confidentiality		✔	✔
Limited traffic flow confidentiality		✔	✔

SECURITY ASSOCIATIONS

Since IPSEC is designed to be able to use various security protocols, it uses Security Associations (SA) to specify the protocols to be used. SA is a database record which specifies security parameters controlling security operations. They are referenced by the sending host and established by the receiving host. An index parameter called the Security Parameters Index (SPI) is used. SAs are in one direction only and a second SA must be established for the transmission to be bi-directional. A security association is uniquely identified by three parameters:

• **Security Parameters Index (SPI):** A bit string assigned to this SA and having local significance only. The SPI is carried in AH and ESP headers to enable the receiving system to select the SA under which a received packet will be processed.

• **IP Destination Address**: Currently, only unicast addresses are allowed; this is the address of the destination endpoint of the SA, which may be an end user system or a network system such as a firewall or router.

• **Security Protocol Identifier**: This indicates whether the association is an AH or ESP security association.

SA Parameters

In each IPSec implementation, there is a nominal Security Association Database that defines the parameters associated with each SA. A security association is normally defined by the following parameters:

• **Sequence Number Counter:** A 32-bit value used to generate the Sequence Number field in AH or ESP headers

• **Sequence Counter Overflow:** A flag indicating whether overflow of the Sequence Number Counter should generate an auditable event and prevent further transmission of packets on this SA (required for all implementations).

• **Anti-Replay Window:** Used to determine whether an inbound AH or ESP packet is a replay
• **AH Information:** Authentication algorithm, keys, key lifetimes, and related

parameters being used with AH (required for AH implementations).

• **ESP Information:** Encryption and authentication algorithm, keys, initialization values, key lifetimes, and related parameters being used with ESP (required for ESP implementations).

• **Lifetime of This Security Association:** A time interval or byte count after which an SA must be replaced with a new SA (and new SPI) or terminated, plus an indication of which of these actions should occur (required for all implementations).

• **IPSec Protocol Mode:** Tunnel, transport, or wildcard (required for all implementations). These modes are discussed later in this section.

• **Path MTU:** Any observed path maximum transmission unit (maximum size of a packet that can be transmitted without fragmentation) and aging variables (required for all implementations).

Transport and Tunnel Modes

Both AH and ESP support two modes of use: transport and tunnel mode.

	Transport Mode SA	Tunnel Mode SA
AH	**Authenticates** IP payload and selected portions of IP header and IPv6 extension headers	**Authenticates** entire inner IP packet plus selected portions of outer IP header
ESP	**Encrypts** IP payload and any IPv6 extesion header	**Encrypts** inner IP packet
ESP with authentication	**Encrypts** IP payload and any IPv6 extesion header. **Authenticates** IP payload but no IP header	**Encrypts** inner IP packet. **Authenticates** inner IP packet

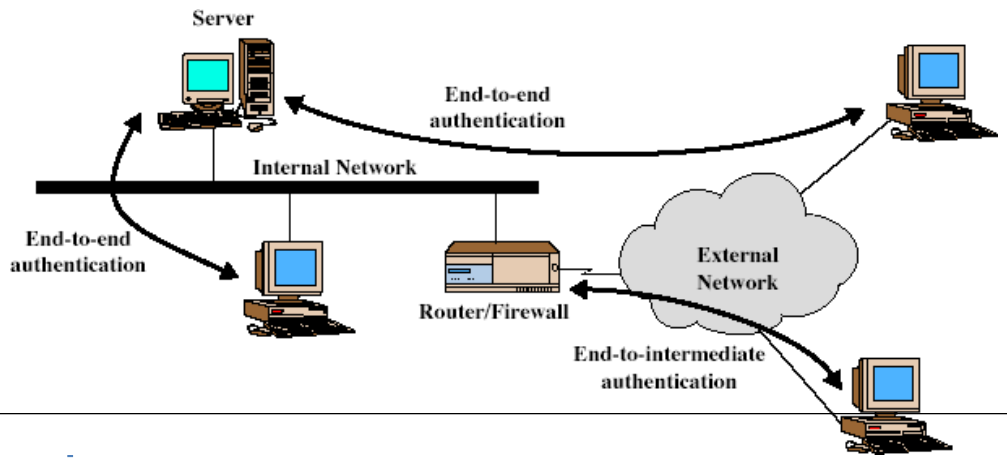
IP sec can be used (both AH packets and ESP packets) in two modes

• **Transport mode**: the IP sec header is inserted just after the IP header –this contains the security information, such as SA identifier, encryption, authentication

- Typically used in end-to-end communication IP header not protected

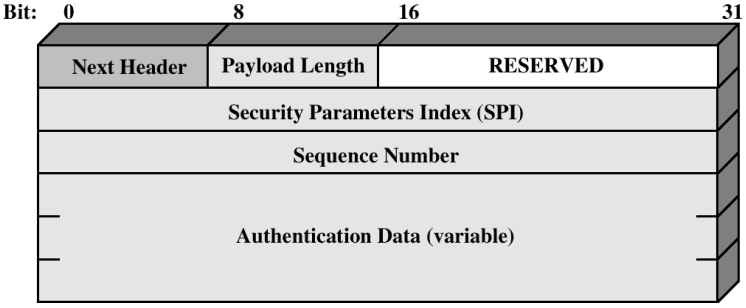
• **Tunnel mode**: the entire IP packet, header and all, is encapsulated in the body of a new IP packet with a completely new IP header

- Typically used in firewall-to-firewall communication Provides protection for the whole IP packet
- No routers along the way will be able (and will not need) to check the content of the packets



AUTHENTICATION HEADER

The Authentication Header provides support for data integrity and authentication of IP packets. The data integrity feature ensures that undetected modification to a packet's content in transit is not possible. The authentication feature enables an end system or network device to authenticate the user or application and filter traffic accordingly; it also prevents the address spoofing attacks observed in today's Internet. The AH also guards against the replay attack. Authentication is based on the use of a message authentication code (MAC), hence the two parties must share a secret key. The Authentication Header consists of the following fields:



IPSec Authentication Header

- **Next Header (8 bits):** Identifies the type of header immediately following this header.
- **Payload Length (8 bits):** Length of Authentication Header in 32-bit words, minus 2. For example, the default length of the authentication data field is 96 bits, or three 32-bit words. With a three-word fixed header, there are a total of six words in the header, and the Payload Length field has a value of 4.
- **Reserved (16 bits):** For future use.
- **Security Parameters Index (32 bits):** Identifies a security association.
- **Sequence Number (32 bits):** A monotonically increasing counter value, discussed later.
- **Authentication Data (variable):** A variable-length field (must be an integral number of 32-bit words) that contains the Integrity Check Value (ICV), or MAC, for this packet.

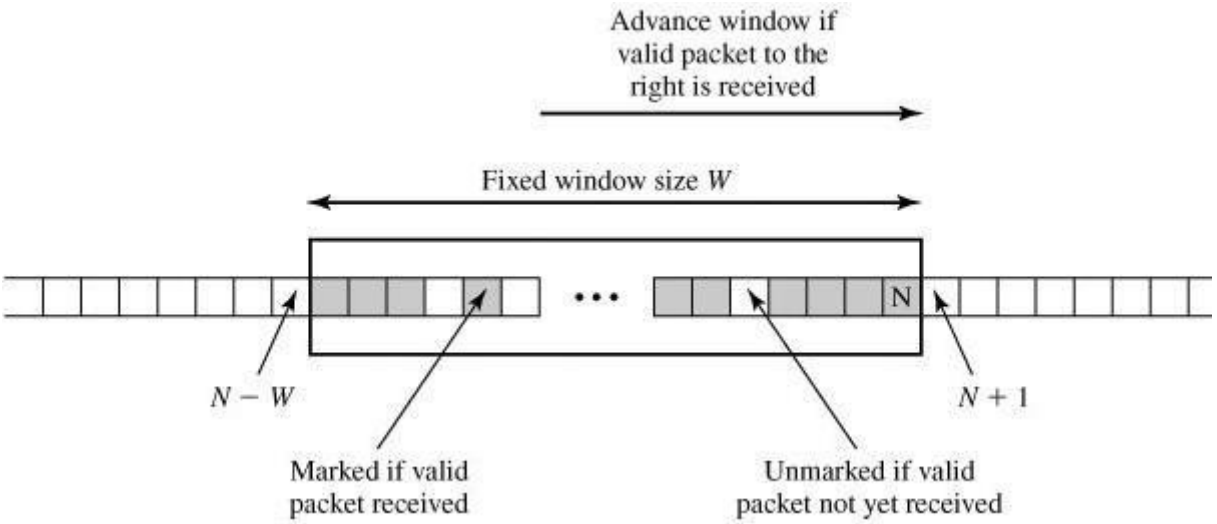
Anti-Replay Service

Anti-replay service is designed to overcome the problems faced due to replay attacks in which an intruder intervenes the packet being transferred, make one or more duplicate copies of that authenticated packet and then sends the packets to the desired destination, thereby causing inconvenient processing at the destination node. The Sequence Number field is designed to thwart such attacks.

When a new SA is established, the sender initializes a sequence number counter to 0. Each time that a packet is sent on this SA, the sender increments the counter and places the value in the Sequence Number field. Thus, the first value to be used is 1. This value goes on increasing with respect to the number of packets being transmitted. The sequence number field in each packet represents the value of this counter. The maximum value of the sequence number field can go up to 2_{32}-1. If the limit of 2_{32}-1 is reached, the sender should terminate this SA and negotiate a new SA with a new key.

The IPSec authentication document dictates that the receiver should implement a

window of size W, with a default of W = 64. The right edge of the window represents the highest sequence number, N, so far received for a valid packet. For any packet with a sequence number in the range from N-W+1 to N that has been correctly received (i.e., properly authenticated), the corresponding slot in the window is marked as shown. Inbound processing proceeds as follows when a packet is received:



Antireplay Mechanism

1. If the received packet falls within the window and is new, the MAC is checked. If the packet is authenticated, the corresponding slot in the window is marked.
2. If the received packet is to the right of the window and is new, the MAC is checked. If the packet is authenticated, the window is advanced so that this sequence number is the right edge of the window, and the corresponding slot in the window is marked.
3. If the received packet is to the left of the window, or if authentication fails, the packet is discarded; this is an auditable event.

Integrity Check Value

ICV is the value present in the authenticated data field of ESP/AH, which is used to determine any undesired modifications made to the data during its transit. ICV can also be referred as MAC or part of MAC algorithm. MD5 hash code and SHA-1 hash code are implemented along with HMAC algorithms i.e.,

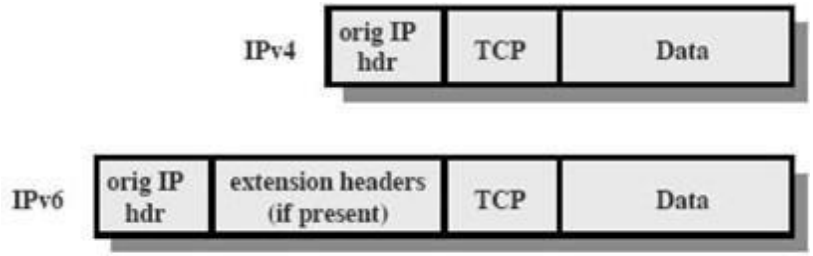
- HMAC-MD5-96
- HMAC-SHA-1-96

In both cases, the full HMAC value is calculated but then truncated by using the first 96 bits, which is the default length for the Authentication Data field. The MAC is calculated over

- IP header fields that either do not change in transit (immutable) or that are predictable in value upon arrival at the endpoint for the AH SA. Fields that may change in transit and whose value on arrival is unpredictable are set to zero for purposes of calculation at both source and destination.
- The AH header other than the Authentication Data field. The Authentication Data field is set to zero for purposes of calculation at both source and destination.
- The entire upper-level protocol data, which is assumed to be immutable in transit (e.g., a TCP segment or an inner IP packet in tunnel mode).

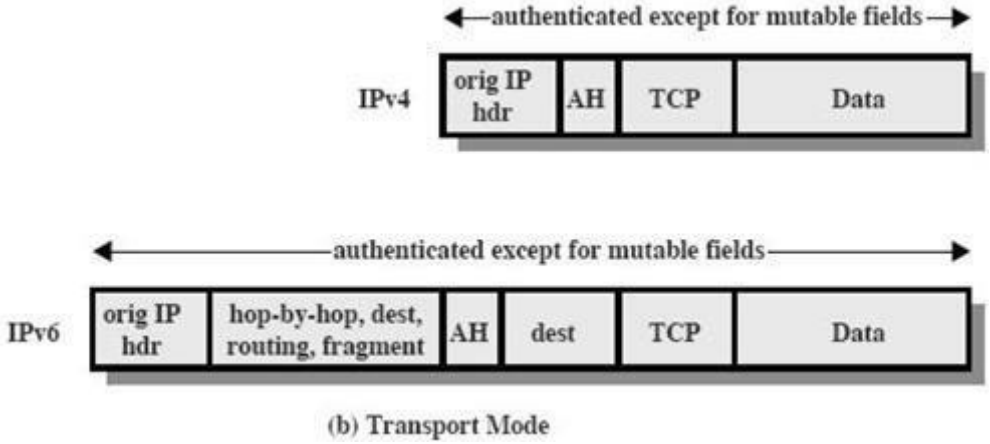
Transport and Tunnel Modes

The following figure shows typical IPv4 and IPv6 packets. In this case, the IP payload is a TCP segment; it could also be a data unit for any other protocol that uses IP, such as UDP or ICMP.

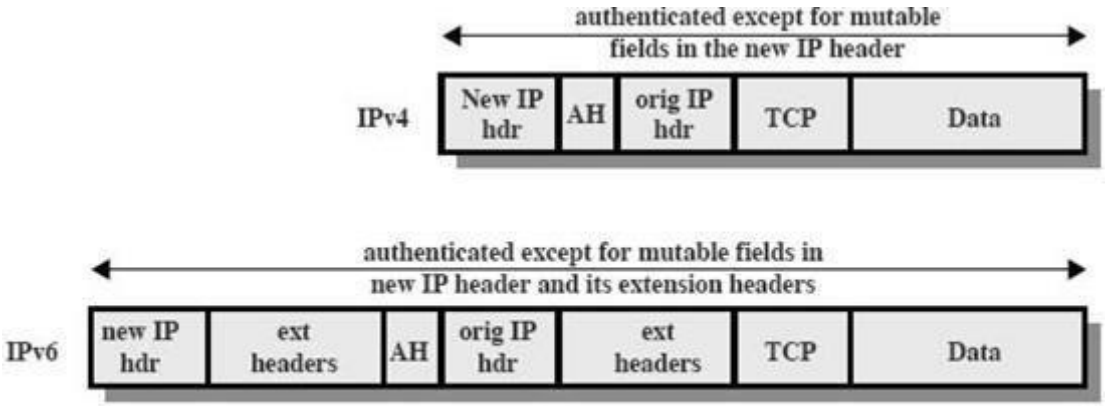


(a) Before Applying AH

For transport mode AH using IPv4, the AH is inserted after the original IP header and before the IP payload (e.g., a TCP segment) shown below. Authentication covers the entire packet, excluding mutable fields in the IPv4 header that are set to zero for MAC calculation. In the context of IPv6, AH is viewed as an end-to-end payload; that is, it is not examined or processed by intermediate routers. Therefore, the AH appears after the IPv6 base header and the hop-by-hop, routing, and fragment extension headers. The destination options extension header could appear before or after the AH header, depending on the semantics desired. Again, authentication covers the entire packet, excluding mutable fields that are set to zero for MAC calculation.



(b) Transport Mode



(c) Tunnel Mode

For tunnel mode AH, the entire original IP packet is authenticated, and the AH is inserted between the original IP header and a new outer IP header. The inner IP header carries the ultimate source and destination addresses, while an outer IP header may contain different IP addresses (e.g., addresses of firewalls or other security gateways). With tunnel mode, the entire inner IP packet, including the entire

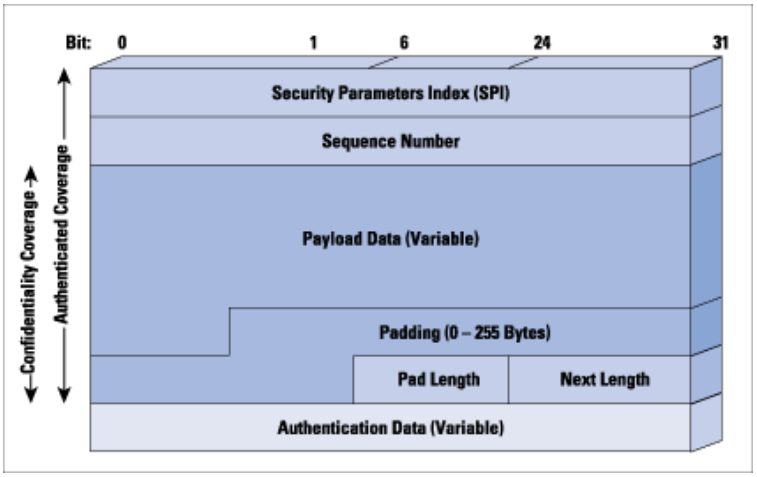
inner IP header is protected by AH. The outer IP header (and in the case of IPv6, the outer IP extension headers) is protected except for mutable and unpredictable fields.

ENCAPSULATING SECURITY PAYLOAD

The Encapsulating Security Payload provides confidentiality services, including confidentiality of message contents and limited traffic flow confidentiality. As an optional feature, ESP can also provide an authentication service.

ESP Format

The following figure shows the format of an ESP packet. It contains the following fields:



- **Security Parameters Index** (32 bits): Identifies a security association.

• **Sequence Number** (32 bits): A monotonically increasing counter value; this provides an anti-replay function, as discussed for AH.

• **Payload Data (variable):** This is a transport-level segment (transport mode) or IP packet (tunnel mode) that is protected by encryption.

• **Padding (0-255 bytes):** This field is used to make the length of the plaintext to be a multiple of some desired number of bytes. It is also added to provideconfidentiality.

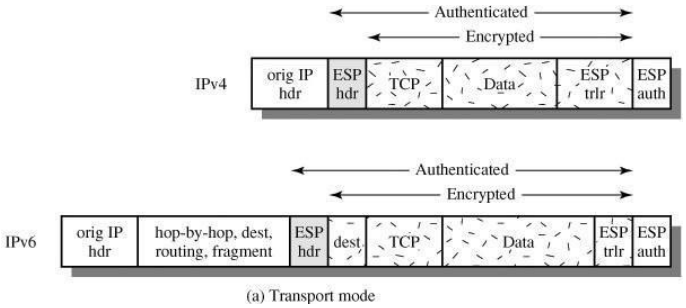
• **Pad Length (8 bits):** Indicates the number of pad bytes immediately preceding this field.

• **Next Header (8 bits):** Identifies the type of data contained in the payload data field by identifying the first header in that payload (for example, an extension header in IPv6, or an upper-layer protocol such as TCP).

• **Authentication Data (variable):** A variable-length field (must be an integral number of 32-bit words) that contains the Integrity Check Value computed over the ESP packet minus the Authentication Data field.

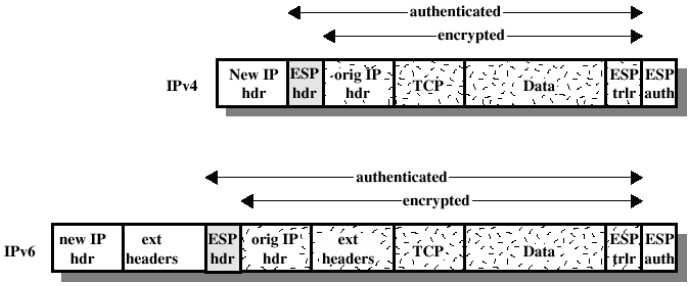
Adding encryption makes ESP a bit more complicated because the encapsulation *surrounds* the payload rather than *precedes* it as with AH: ESP includes header and trailer

Transport Mode ESP



(a) Transport mode

Tunnel Mode ESP

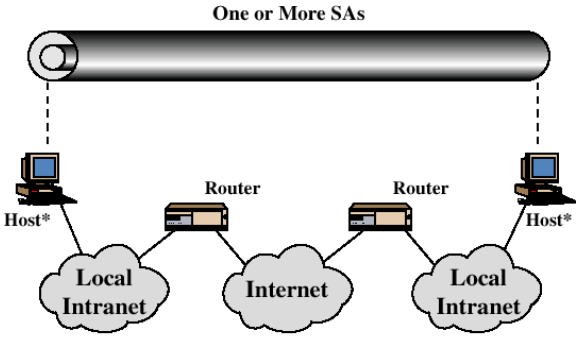


(b) Tunnel Mode

Basic Combinations of Security Associations

The IPSec Architecture document lists four examples of combinations of SAs that must be supported by compliant IPSec hosts (e.g., workstation, server) or security gateways (e.g. firewall, router).

case:-1



(a) Case 1

All security is provided between end systems that implement IPSec. For any two end systems to communicate via an SA, they must share the appropriate secret keys. Among the possible combinations:

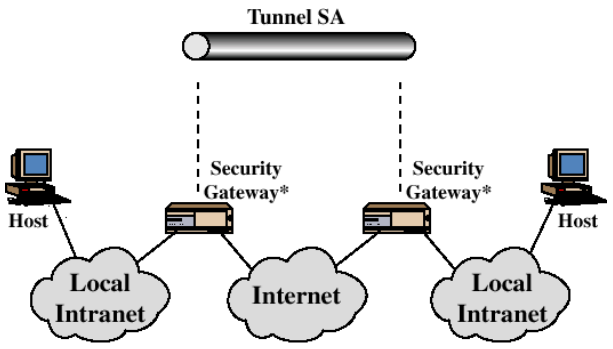
a) AH in transport mode

b) ESP in transport mode

c) ESP followed by AH in transport mode (an ESP SA inside an AH SA)

d) Any one of a, b, or c inside an AH or ESP in tunnel mode

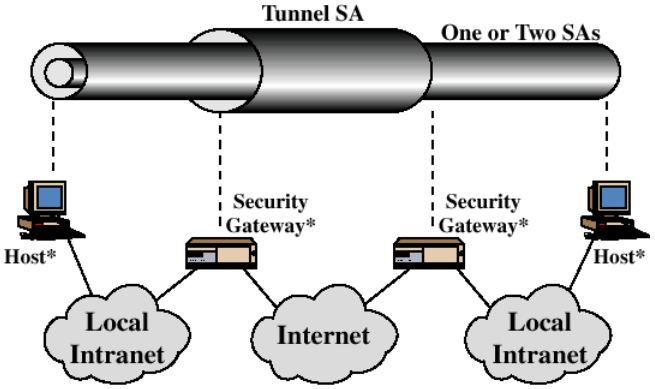
Case:-2



(b) Case 2

Security is provided only between gateways (routers, firewalls, etc.) and no hosts implement IPSec. This case illustrates simple virtual private network support. The security architecture document specifies that only a single tunnel SA is needed for this case. The tunnel could support AH, ESP, or ESP with the authentication option. Nested tunnels are not required because the IPSec services apply to the entire inner packet.

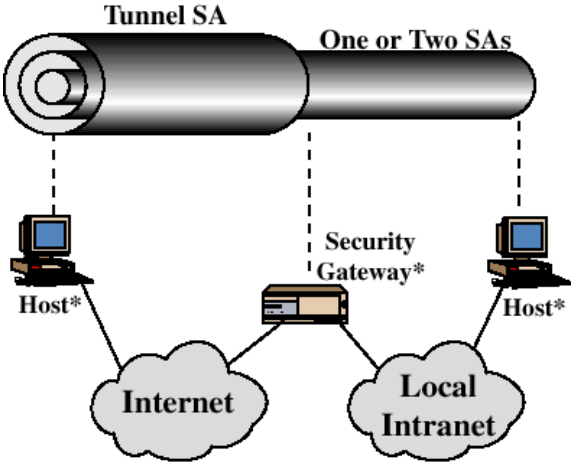
Case-3:-



(c) Case 3

The third combination is similar to the second, but in addition provides security even to nodes. This combination makes use of two tunnels first for gateway to gateway and second for node to node. Either authentication or the encryption or both can be provided by using gateway to gateway tunnel. An additional IPSec service is provided to the individual nodes by using node to node tunnel.

Case:-4



(d) Case 4

This combination is suitable for serving remote users i.e., the end user sitting anywhere in the world can use the internet to access the organizational workstations via the firewall. This combination states that only one tunnel is needed for communication between a remote user and an organizational firewall.

KEY MANAGEMENT

The key management portion of IPSec involves the determination and distribution of secret keys. The IPSec Architecture document mandates support for two types of key management:

• **Manual:** A system administrator manually configures each system with its own keys and with the keys of other communicating systems. This is practical for small, relatively static environments.

• **Automated:** An automated system enables the on-demand creation of keys for SAs and facilitates the use of keys in a large distributed system with an evolving configuration.

The default automated key management protocol for IPSec is referred to as ISAKMP/Oakley and consists of the following elements:

• **Oakley Key Determination Protocol**: Oakley is a key exchange protocol based on the Diffie-Hellman algorithm but providing added security. Oakley is generic in that it does not dictate specific formats.

• **Internet Security Association and Key Management Protocol (ISAKMP):** ISAKMP provides a framework for Internet key management and provides the specific protocol support, including formats, for negotiation of security attributes.

Oakley Key Determination Protocol

Oakley is a refinement of the Diffie-Hellman key exchange algorithm. The Diffie-Hellman algorithm has two attractive features:

• Secret keys are created only when needed. There is no need to store secret keys for a long period of time, exposing them to increased vulnerability.

• The exchange requires no pre-existing infrastructure other than an agreement on the global parameters.

However, Diffie-Hellman has got some weaknesses:

• No identity information about the parties is provided.

• It is possible for a man-in-the-middle attack

• It is computationally intensive. As a result, it is vulnerable to a clogging attack, in which an opponent requests a high number of keys.

Oakley is designed to retain the advantages of Diffie-Hellman while countering its weaknesses.

Features of Oakley

The Oakley algorithm is characterized by five important features:

1. It employs a mechanism known as cookies to thwart clogging attacks.
2. It enables the two parties to negotiate a group; this, in essence, specifies the global parameters of the Diffie-Hellman key exchange.

3. It uses nonces to ensure against replay attacks.
4. It enables the exchange of Diffie-Hellman public key values.
5. It authenticates the Diffie-Hellman exchange to thwart man-in-the-middle attacks.
In clogging attacks, an opponent forges the source address of a legitimate user and sends a public Diffie-Hellman key to the victim. The victim then performs a modular exponentiation to compute the secret key. Repeated messages of this type can clog the victim's system with useless work. The **cookie exchange** requires that each side send a pseudorandom number, the cookie, in the initial message, which the other side acknowledges. This acknowledgment must be repeated in the first message of the Diffie- Hellman key exchange. The recommended method for creating the cookie is to perform a
fast hash (e.g., MD5) over the IP Source and Destination addresses, the UDP Source and Destination ports, and a locally generated secret value. Oakley supports the use of different **groups** for the Diffie-Hellman key exchange. Each group includes the definition of the two global parameters and the identity of the algorithm. Oakley employs **nonces** to ensure against replay attacks. Each nonce is a locally generated pseudorandom number. Nonces appear in responses and are encrypted during certain portions of the exchange to secure their use. Three different authentication methods can be used with Oakley are digital signatures, public-key encryption and Symmetric-key encryption.

Aggressive Oakley Key Exchange

Aggressive key exchange is a technique used for exchanging the message keys and is so called because only three messages are allowed to be exchanged at any time.

Example of Aggressive Oakley Key Exchange

$\mathbf{I \rightarrow R:}$ $CKY_I, OK\_KEYX, GRP, g^x, EHAO, NIDP, ID_I, ID_R, N_I, S_{KI}[ID_I \| ID_R \| N_I \| GRP \| g^x \| EHAO]$

$\mathbf{R \rightarrow I:}$ $CKY_R, CKY_I, OK\_KEYX, GRP, g^y, EHAS, NIDP, ID_R, ID_I, N_R, N_I, S_{KR}[ID_R \| ID_I \| N_R \| N_I \| GRP \| g^y \| g^x \| EHAS]$

$\mathbf{I \rightarrow R:}$ $CKY_I, CKY_R, OK\_KEYX, GRP, g^x, EHAS, NIDP, ID_I, ID_R, N_I, N_R, S_{KI}[ID_I \| ID_R \| N_I \| N_R \| GRP \| g^x \| g^y \| EHAS]$

Notation:

I	=	Initiator
R	=	Responder
CKY_I, CKY_R	=	Initiator, responder cookies
OK_KEYX	=	Key exchange message type
GRP	=	Name of Diffie-Hellman group for this exchange
g^x, g^y	=	Public key of initiator, responder; g^{xy} = session key from this exchange
EHAO, EHAS	=	Encryption, hash authentication functions, offered and selected
NIDP	=	Indicates encryption is not used for remainder of this message
ID_I, ID_R	=	Identifier for initiator, responder
N_I, N_R	=	Random nonce supplied by initiator, responder for this exchange
$S_{KI}[X], S_{KR}[X]$	=	Indicates the signature over X using the private key (signing key) of intiator, responder

In the first step, the initiator (I) transmits a cookie, the group to be used, and I's public Diffie-Hellman key for this exchange. I also indicates the offered public-key encryption, hash, and authentication algorithms to be used in this exchange. Also included in this message are the identifiers of I and the responder (R) and I's nonce for this exchange. Finally, I appends a signature using I's private key that signs the two identifiers, the nonce, the group, the Diffie-Hellman public key, and the offered algorithms. When R receives the message, R verifies the signature using I's public signing key. R acknowledges the message by echoing back I's cookie, identifier, and nonce, as well as the group. R also includes in the message a cookie, R's Diffie-Hellman public key, the selected algorithms (which must be among the offered algorithms), R's identifier, and R's nonce for this exchange. Finally, R appends a

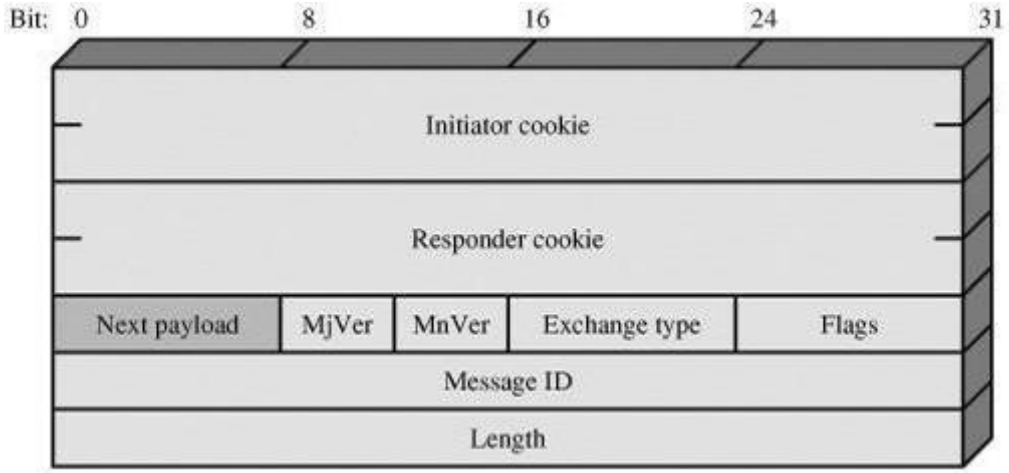
signature using R's private key that signs the two identifiers, the two nonces, the group, the two Diffie-Hellman public keys, and the selected algorithms.

When I receives the second message, I verifies the signature using R's public key. The nonce values in the message assure that this is not a replay of an old message. To complete the exchange, I must send a message back to R to verify that I has received R's public key.

ISAKMP

ISAKMP defines procedures and packet formats to establish, negotiate, modify, and delete security associations. As part of SA establishment, ISAKMP defines payloads for exchanging key generation and authentication data.

ISAKMP Header Format



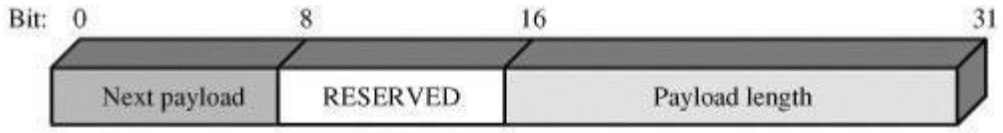
(a) ISAKMP header

An ISAKMP message consists of an ISAKMP header followed by one or more payloads and must follow UDP transport layer protocol for its implementation. The header format of an ISAKMP header is shown below:

- Initiator Cookie (64 bits): Cookie of entity that initiated SA establishment, SA notification, or SA deletion.
- Responder Cookie (64 bits): Cookie of responding entity; null in first message from initiator.
- Next Payload (8 bits): Indicates the type of the first payload in the message
- Major Version (4 bits): Indicates major version of ISAKMP in use.
- Minor Version (4 bits): Indicates minor version in use.
- Exchange Type (8 bits): Indicates the type of exchange. Can be informational, aggressive, authentication only, identity protection or base exchange (S).
- Flags (8 bits): Indicates specific options set for this ISAKMP exchange. Two bits so far defined: The Encryption bit is set if all payloads following the header are encrypted using the encryption algorithm for this SA. The Commit bit is used to ensure that encrypted material is not received prior to completion of SA establishment.
- Message ID (32 bits): Unique ID for this message.
- Length (32 bits): Length of total message (header plus all payloads) in octets.

ISAKMP Payload Types

All ISAKMP payloads begin with the same generic payload header shown below.



(b) Generic payload header

The Next Payload field has a value of 0 if this is the last payload in the message; otherwise its value is the type of the next payload. The Payload Length field indicates the length in octets of this payload, including the generic payload header. There are many different ISAKMP payload types. They are:

a. The SA payload is used to begin the establishment of an SA. The Domain of Interpretation parameter identifies the DOI under which negotiation is taking place. The Situation parameter defines the security policy for this negotiation; in essence, the levels of security required for encryption and confidentiality are specified (e.g., sensitivity level, security compartment).

b. The Proposal payload contains information used during SA negotiation. The payload indicates the protocol for this SA (ESP or AH) for which services and mechanisms are being negotiated. The payload also includes the sending entity's SPI and the number of transforms. Each transform is contained in a transform payload.

c. The Transform payload defines a security transform to be used to secure the communications channel for the designated protocol. The Transform # parameter serves to identify this particular payload so that the responder may use it to indicate acceptance

of this transform. The Transform-ID and Attributes fields identify a specific transform (e.g., 3DES for ESP, HMAC-SHA-1-96 for AH) with its associated attributes (e.g., hash length).

d. The Key Exchange payload can be used for a variety of key exchange techniques, including Oakley, Diffie-Hellman, and the RSA-based key exchange used by PGP. The Key Exchange data field contains the data required to generate a session key and is dependent on the key exchange algorithm used.

e. The Identification payload is used to determine the identity of communicating peers and may be used for determining authenticity of information. Typically the ID Data field will contain an IPv4 or IPv6 address.

f. The Certificate payload transfers a public-key certificate. The Certificate Encoding field indicates the type of certificate or certificate-related information, which may include SPKI, ARL, CRL, PGP info etc. At any point in an ISAKMP exchange, the sender may include a Certificate Request payload to request the certificate of the other communicating entity.

g. The Hash payload contains data generated by a hash function over some part of the message and/or ISAKMP state. This payload may be used to verify the integrity of the data in a message or to authenticate negotiating entities.

h. The Signature payload contains data generated by a digital signature function over some part of the message and/or ISAKMP state. This payload is used to verify the

integrity of the data in a message and may be used for nonrepudiation services.

i.

The Nonce payload contains random data used to guarantee liveness during an exchange and protect against replay attacks.

j. The Notification payload contains either error or status information associated with this SA or this SA negotiation. Some of the ISAKMP error messages that have been defined are Invalid Flags, Invalid Cookie, Payload Malformed etc

k. The Delete payload indicates one or more SAs that the sender has deleted from its database and that therefore are no longer valid.

ISAKMP Exchanges

ISAKMP provides a framework for message exchange, with the payload types serving as the building blocks. The specification identifies five default exchange types that should be supported.

1. Base Exchange: allows key exchange and authentication material to be transmitted together. This minimizes the number of exchanges at the expense of not providing identity protection.

The first two messages provide cookies and establish an SA with agreed protocol and

(a) Base Exchange

(1)**I**⟶**R**: SA; NONCE	Begin ISAKMP-SA negotiation
(2)**R**⟶**E**: SA; NONCE	Basic SA agreed upon
(3)**I**⟶**R**: KE; ID_I AUTH	Key generated; Initiator identity verified by responder
(4)**R**⟶**E**: KE; ID_R AUTH	Responder identity verified by initiator; Key generated; SA established

transforms; both sides use a nonce to ensure against replay attacks. The last two messages exchange the key material and user IDs, with an authentication mechanism used to authenticate keys, identities, and the nonces from the first two messages.

2. Identity Protection Exchange: expands the Base Exchange to protect the users' identities.

(b) Identity Protection Exchange

(1)**I**⟶**R**: SA	Begin ISAKMP-SA negotiation
(2)**R**⟶**E**: SA	Basic SA agreed upon
(3)**I**⟶**R**: KE; NONCE	Key generated
(4)**R**⟶**E**: KE; NONCE	Key generated
(5)*__I__⟶**R**: ID_I; AUTH	Initiator identity verified by responder
(6)*__R__⟶**E**: ID_R; AUTH	Responder identity verified by initiator; SA established

The first two messages establish the SA. The next two messages perform key exchange, with nonces for replay protection. Once the session key has been computed, the two parties

exchange encrypted messages that contain authentication information, such as digital signatures and optionally certificates validating the public keys.

3. Authentication Only Exchange: used to perform mutual authentication, without a key exchange

The first two messages establish the SA. In addition, the responder uses the second message to convey its ID and uses authentication to protect the message. The initiator sends the third message to transmit its authenticated ID.

4. Aggressive Exchange: minimizes the number of exchanges at the expense of not providing identity protection.

(d) Aggressive Exchange

(1)**I**⟶**R**: SA; KE; NONCE; ID_I;	Begin ISAKMP-SA negotiation and key exchange
(2)**R**⟶**E**: SA; KE; NONCE; ID_R; AUTH	Initiator identity verified by responder; Key generated; Basic SA agreed upon
(3)*I⟶**R**: AUTH	Responder identity verified by initiator; SA established

In the first message, the initiator proposes an SA with associated offered protocol and transform options. The initiator also begins the key exchange and provides its ID. In the second message, the responder indicates its acceptance of the SA with a particular protocol and transform, completes the key exchange, and authenticates the transmitted information. In the third message, the initiator transmits an authentication result that covers the previous information, encrypted using the shared secret session key.

5. Informational Exchange: used for one-way transmittal of information for SA management.

(e) Informational Exchange

(1)*I⟶**R**: N/D	Error or status notification, or deletion