Topics to Cover

- Message Authentication Requirements
- Message Authentication Functions
- Requirements for Message Authentication Codes (MACs)
- MACs based on hash functions

MESSAGE AUTHENTICATION REQUIREMENTS

Some of the attacks in a Communication Network

1) Disclosure
2) Traffic Analysis
3) Masquerade
4) Content Modification
5) Sequence Modification
6) Timing Modification
7) Source Repudiation
8) Destination Repudiation

Measures to resist different attacks in a Communication Network

- Disclosure and Traffic Analysis attacks can be countered by *Message Confidentiality*.
- Masquerading and the Modification attacks can be prevented by *Message Authentication*.
- Source Repudiation can be mitigated by *Digital Signatures*.
- Destination Repudiation can be countered by *Digital Signatures + Specific Protocol*.

- Message Authentication ensures that received messages originate from the correct source, have not been altered, and that their order and timing are accurate.
- A Digital Signature is an authentication method that also helps prevents Sender Repudiation.

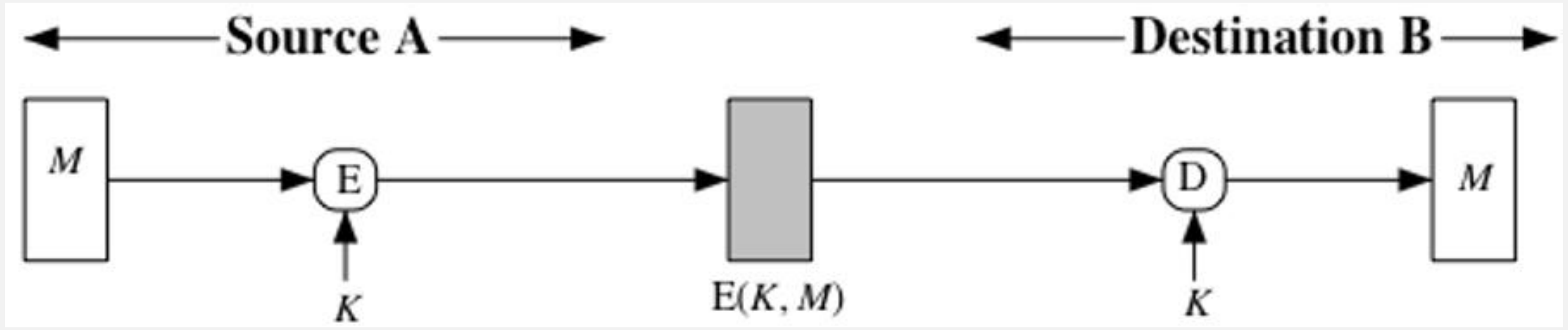
MESSAGE AUTHENTICATION FUNCTIONS

Message Authentication Functions

- ➢Message Authentication or Digital Signature mechanism has two levels:
  - Lower-level function: Produces an authenticator (value used to authenticate a message).
  - Higher-level protocol: Uses the authenticator to verify the message's authenticity.

- ➢Types of functions to produce an authenticator:
  - Hash Function: Maps a message of any length to a fixed-length hash value as the authenticator.
  - Message Encryption: The ciphertext of the entire message serves as the authenticator.
  - MAC: Uses a secret key and message to produce a fixed-length value as the authenticator.

Symmetric Encryption to provide Confidentiality and Authentication



- 'A' message 'M' is encrypted using a secret key 'K' shared between Source 'A' and Destination 'B'.
- If no one else knows the key, only ‘A’ and ‘B’ can read the message, ensuring confidentiality.

Symmetric Encryption to provide Confidentiality and Authentication (Contd..)

- When 'B' receives the message, 'B' knows it came from 'A' because 'A' is the only one who has the key 'K'.
- If the message 'M' is received correctly, 'B' can be sure it hasn’t been altered since only 'A' could have encrypted it in the first place.

- Symmetric Encryption uses the same key for both encrypting and decrypting, providing both confidentiality and authenticity.

Symmetric Encryption to provide Confidentiality and Authentication (Contd..)

- 'B' decrypts the message using the key 'K'.
- If the input to the decryption function is valid ciphertext from 'A', 'B' will get a meaningful message.
- If the input is random or altered ciphertext, 'B' will likely get a non-readable message.

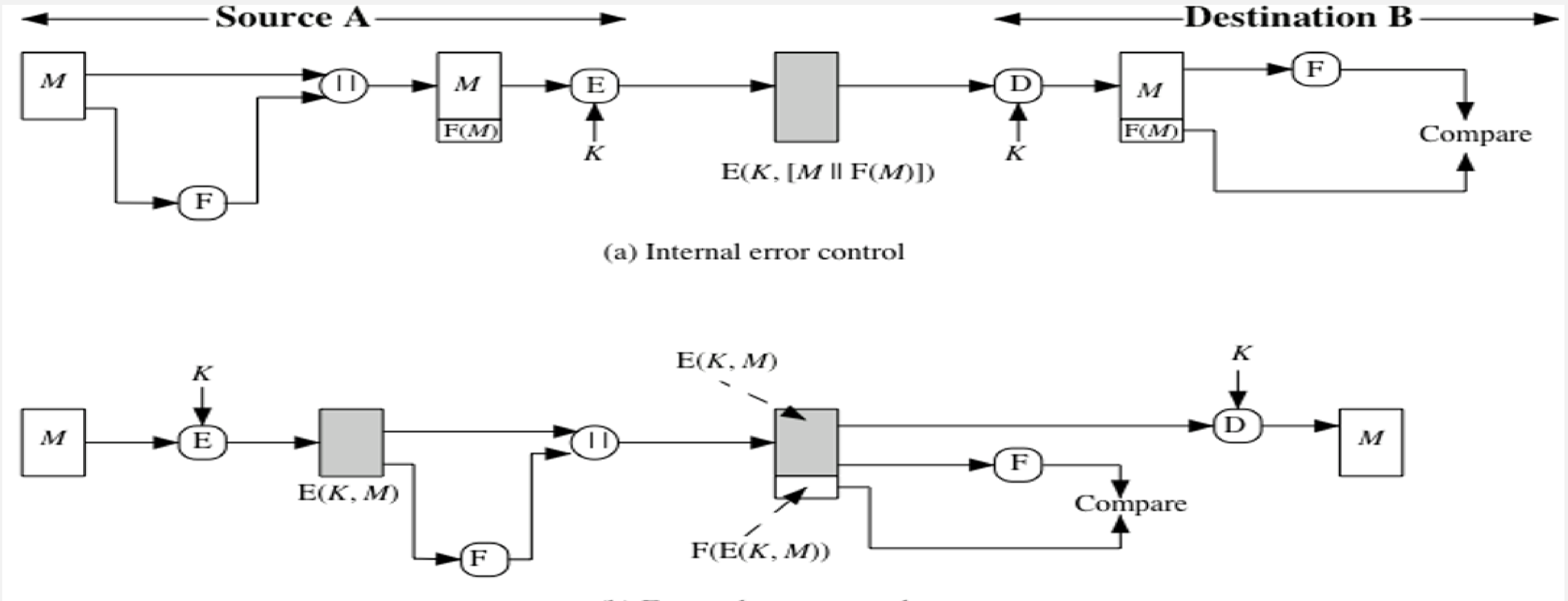
Symmetric Encryption to provide Confidentiality and Authentication (Contd..)

- If the message M can be any random bit pattern, there’s no way to know for sure if a decrypted message is legitimate.
- In this case, any bit pattern could be accepted as authentic plaintext, which is a problem.
- To prevent this, we need to restrict the possible bit patterns for legitimate plaintext.
- For example, if only 1 out of 10^6 bit patterns is legitimate, the chance of a random ciphertext producing a valid plaintext is very low.
- It can be difficult to automatically tell if decrypted ciphertext results in meaningful or valid plaintext, especially for complex files like binary objects or X-rays.
- An attacker could send random, fake messages that appear to come from a legitimate user, causing disruption.
- To ensure valid plaintext, a recognizable structure (like an error-detecting code or checksum) can be added to the message before encryption, making it easier to identify legitimate content.

Frequency Check Sequence (FCS)

- ‘A’ creates a plaintext message ‘M’ and generates an FCS using function ‘F’.
- The FCS is appended to ‘M’, and the entire block is then encrypted.
- ‘B’ decrypts the block and separates the message from the FCS.
- ‘B’ applies the same function ‘F’ to the message to recompute the FCS.
- If the computed FCS matches the incoming one, the message is considered authentic.
- Internal error control ensures authentication by making it difficult for an opponent to generate valid ciphertext with correct error control bits.
- If the FCS is the outer code, an opponent could still create messages with valid error-control codes, potentially causing confusion or disruption despite not knowing the decrypted plaintext.

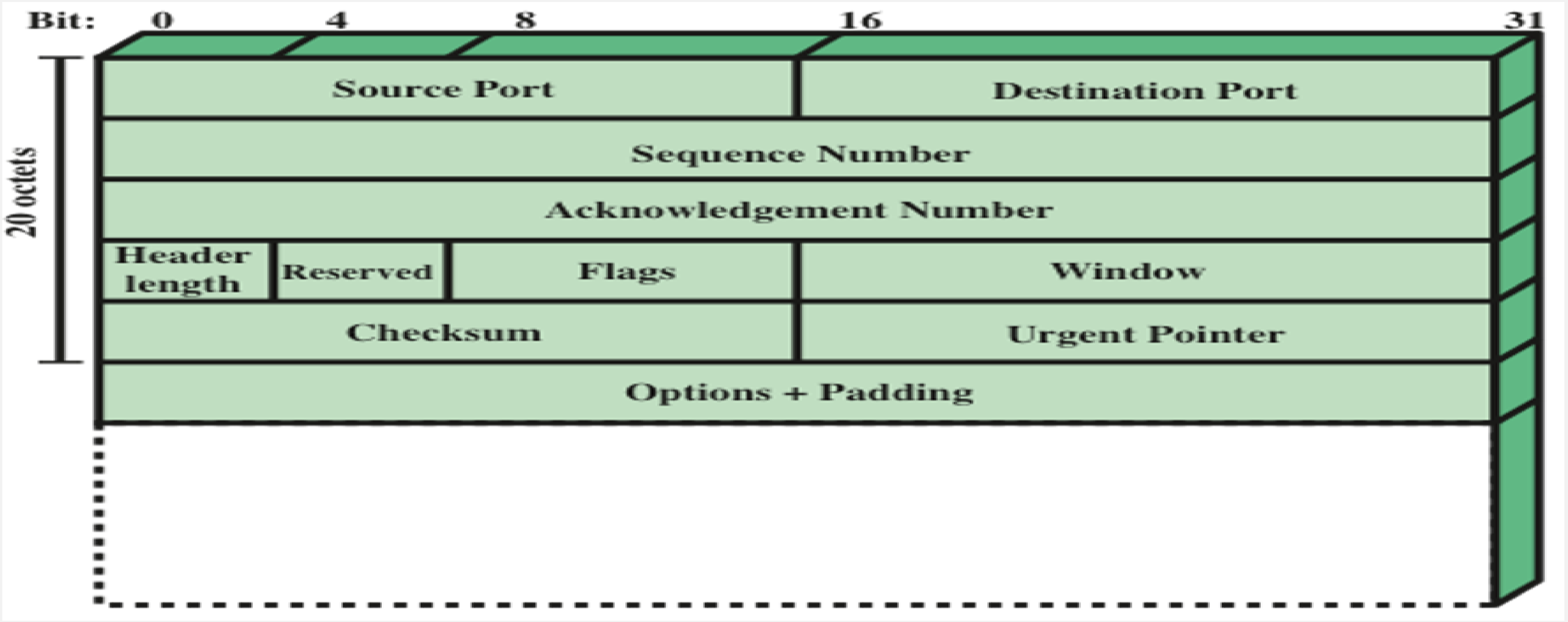
Internal and External Error Control



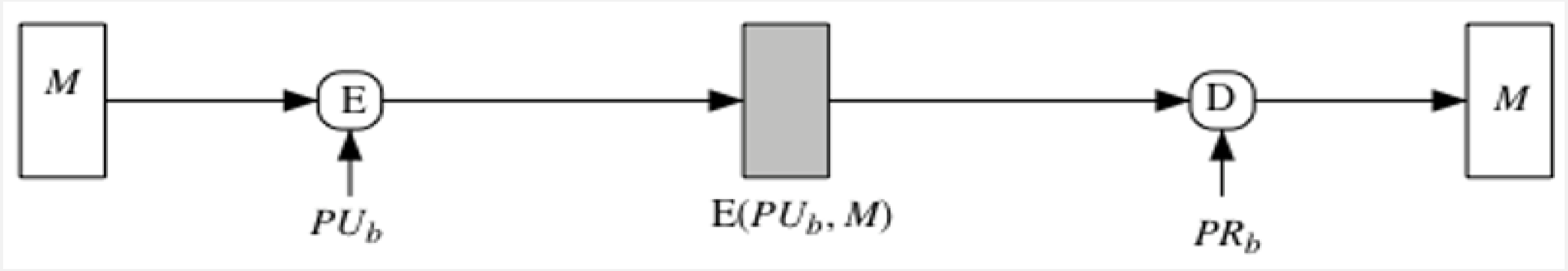
(a) Internal error control

(b) External error control

TCP Segment

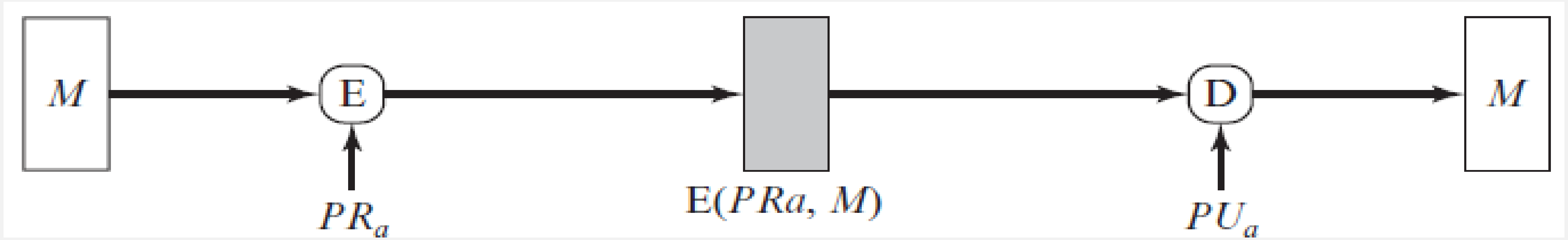

Bit: 0
4
8
16
31
20 octets
Source Port
Destination Port
Sequence Number
Acknowledgement Number
Header length
Reserved
Flags
Window
Checksum
Urgent Pointer
Options + Padding

Public Key Encryption to provide Confidentiality



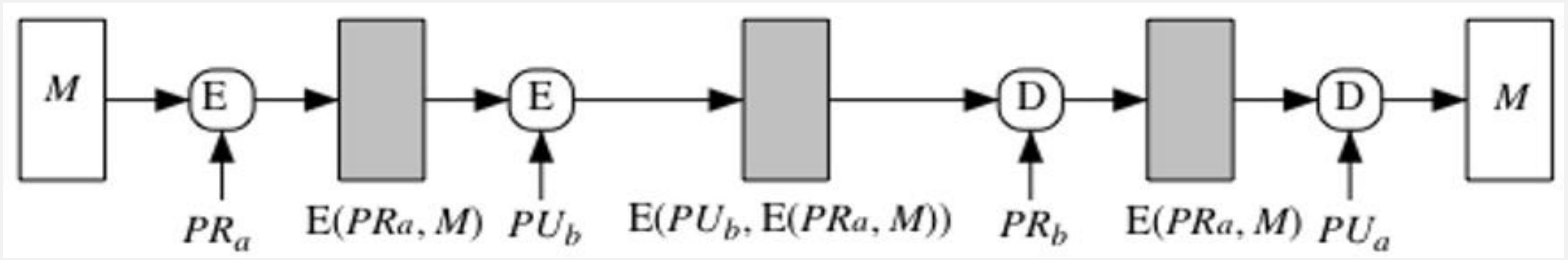
- Public-Key encryption ensures confidentiality, as only the recipient (B) can decrypt the message using their private key.
- It doesn't provide authentication because an attacker could also encrypt a message with B’s public key and impersonate A.

Public Key Encryption to provide Authentication and Signature



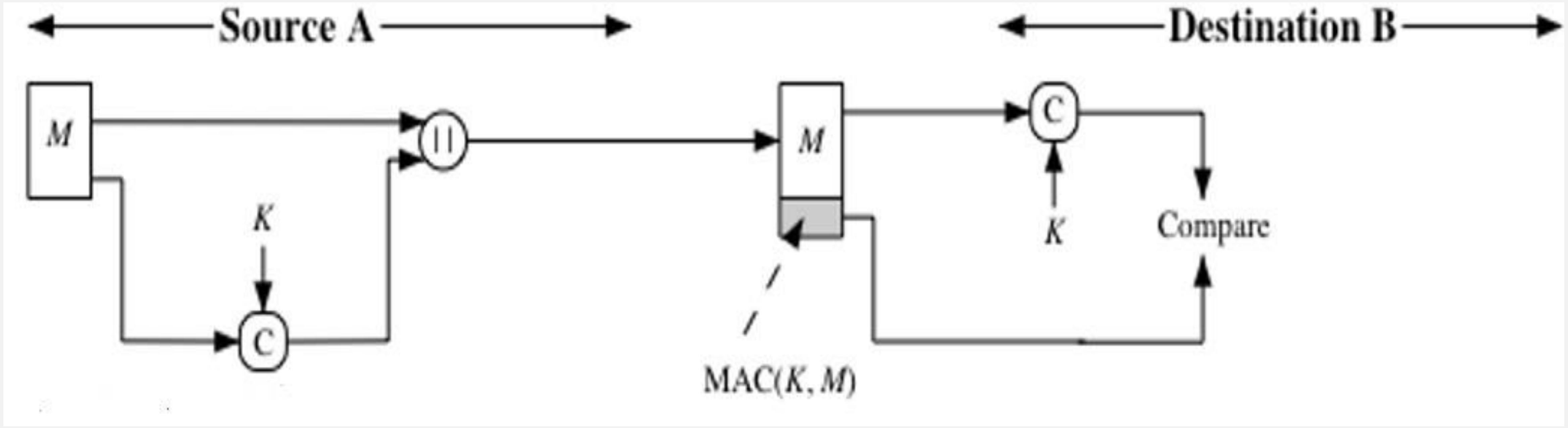
- 'A' encrypts the message with its private key, and 'B' decrypts it with A’s public key, proving the message came from 'A' (providing Authentication).
- The ciphertext can only be created by ‘A’, so ‘B’ can verify that ‘A’ "signed" the message using its private key. (providing Digital Signature)
- However, the scheme does not provide Confidentiality.

Public Key Encryption to provide Confidentiality, Authentication, and Signature



- A encrypts the message (M) first with its private key for authentication (digital signature), then with B's public key for confidentiality.
- The disadvantage is that the public-key process is more complex and needs to be done four times instead of two in each communication.

MAC



- Alternative authentication method uses a secret key to create a small block of data called a cryptographic checksum or MAC.
- ‘A’ and ‘B’ are two parties who share a secret key ‘K’.

MAC (Contd..)

- When ‘A’ wants to send a message ‘M’ to ‘B’, ‘A’ calculates the MAC using the message and the shared key: MAC = C(K, M); where ‘C’ is the MAC function.
- ‘A’ sends the message along with the MAC to ‘B’.
- ‘B’ receives the message and calculates its own MAC using the same secret key.
- ‘B’ compares the calculated MAC with the received MAC to check if the message is authentic.
- Assuming only the sender and receiver know the secret key, and the received MAC matches the calculated one, then following security features can be assured: Message Integrity, Sender Authentication, and Sequence Number Integrity.

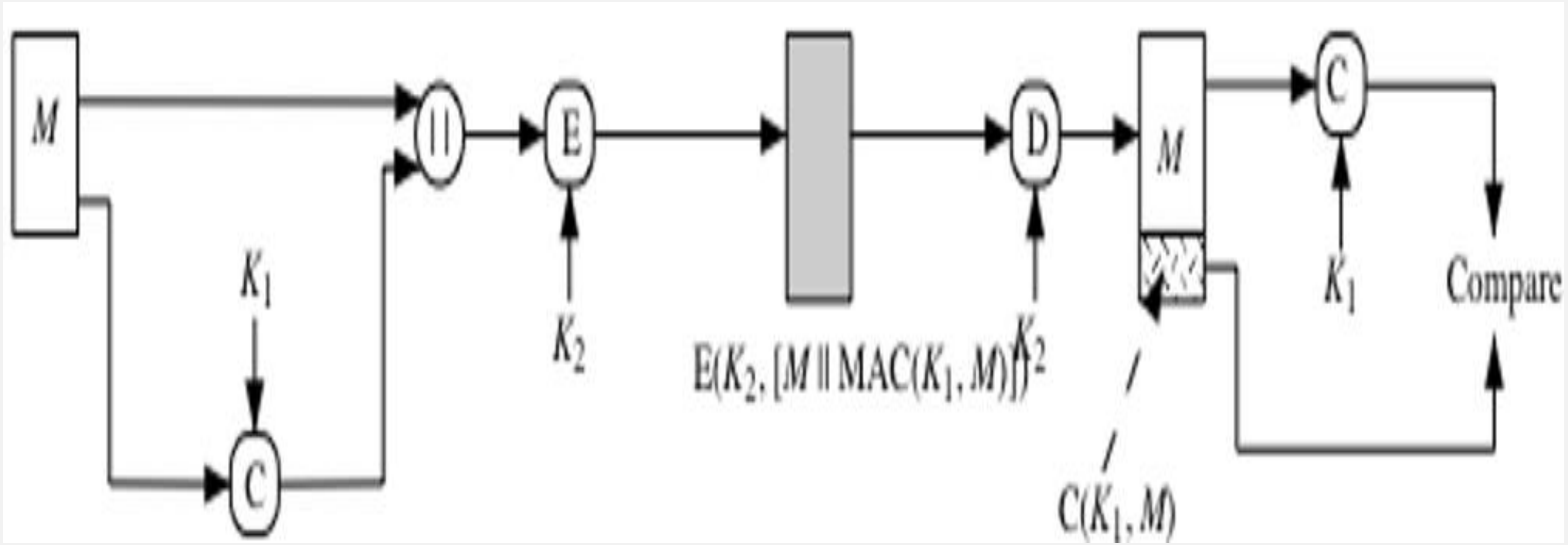
MAC (Contd..)

- A MAC function is similar to encryption but doesn't need to be reversible, and it maps many messages to the same MAC value.
- The number of possible MACs depends on the bit size, with 2^n possible MACs and 2^k possible keys, while the number of messages (N) is much larger than the number of MACs.

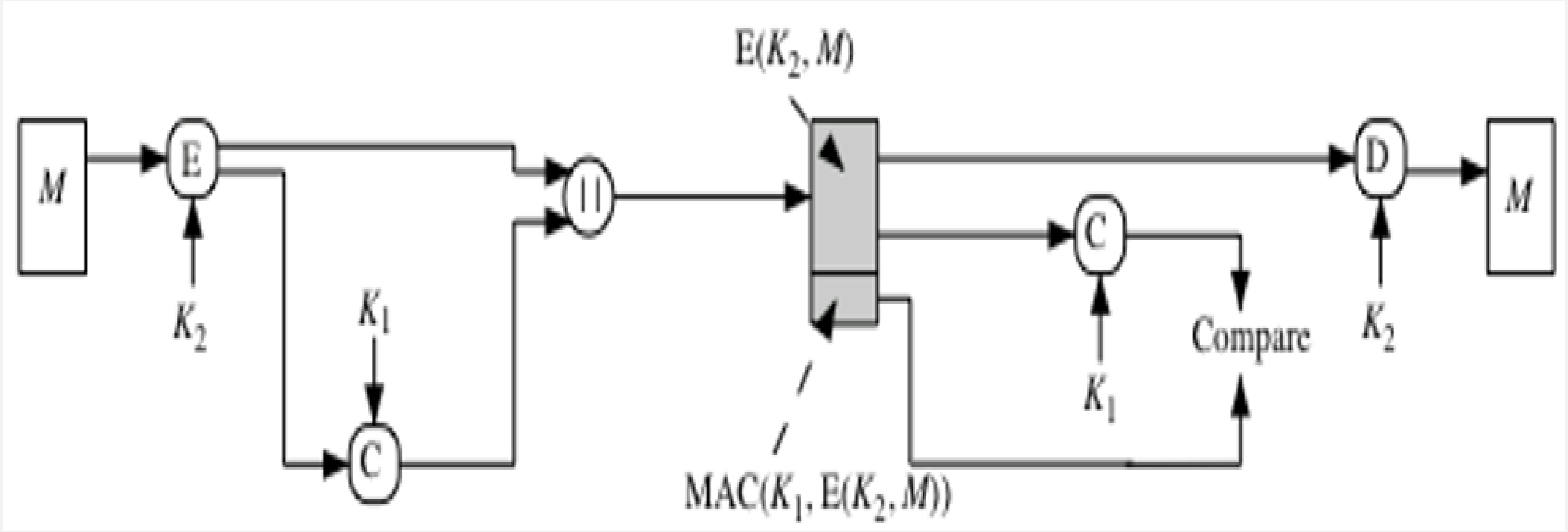
➢ For example:-

- With 100-bit messages and a 10-bit MAC, there are 2^{100} possible messages but only 2^{10} possible MACs.
- On average, each MAC corresponds to 2^{90} different messages.
- If a 5-bit key is used, there are only 32 possible mappings between messages and MACs.

MAC for Confidentiality and Authentication: Authentication tied to Plaintext



MAC for Confidentiality and Authentication: Authentication tied to Ciphertext



Rationales of using MACs instead of just using Symmetric Encryption for Authentication

- In applications where the same message is broadcast to multiple destinations, it is more cost-effective and reliable to have a single system monitor authenticity using a message authentication code, with violations triggering an alert to other destinations.
- In a scenario where one communicating entity with a heavy load selectively authenticates randomly chosen messages due to the time constraints of decrypting all incoming messages.
- Authentication of a computer program in plaintext is efficient as it allows execution without decryption, and attaching a message authentication code ensures program integrity can be verified when needed.

Rationales of using MACs instead of just using Symmetric Encryption for Authentication (Contd..)

- In some applications, such as SNMPv3, authenticating messages is important, especially when they contain commands to change system parameters, while confidentiality may not be necessary.
- Separation of authentication and confidentiality functions allows architectural flexibility, enabling authentication at the application level and confidentiality at a lower level, such as the transport layer.
- A user may want to extend protection beyond message reception while still allowing content processing, but with message encryption, the protection is lost upon decryption, leaving the message vulnerable to fraudulent modifications within the target system.

REQUIREMENTS FOR MACs

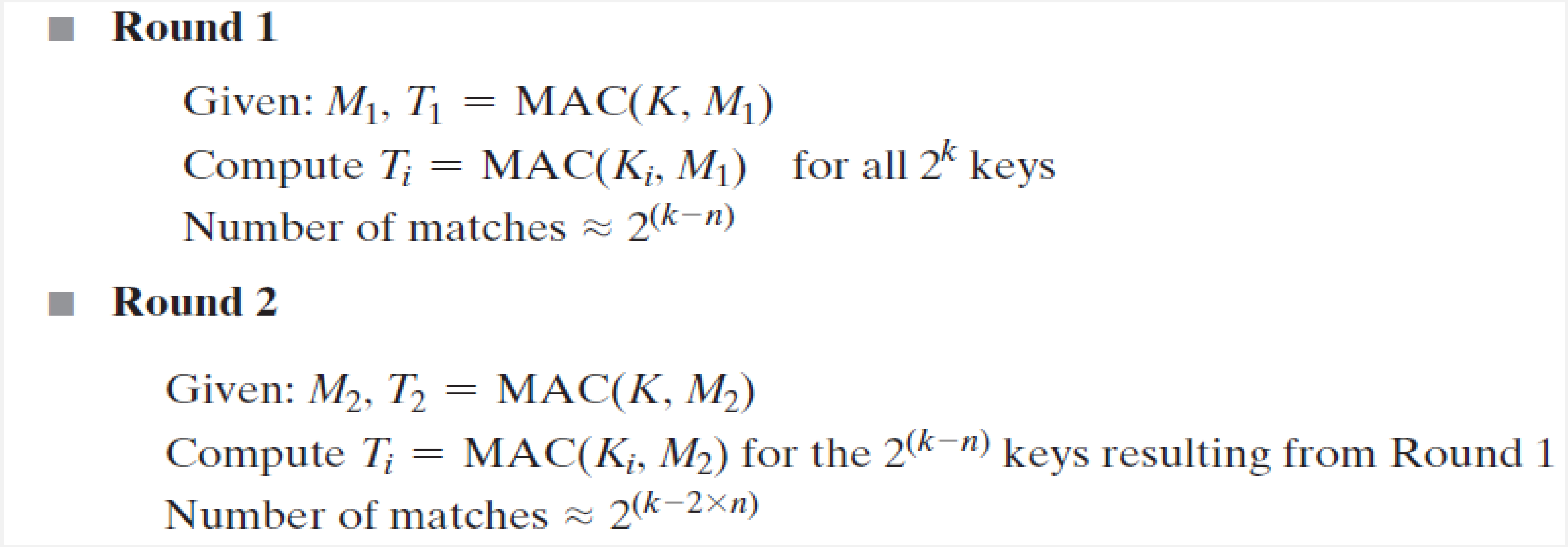
Cryptographic Message Authentication and Key-Based Security

- A MAC is a fixed-length tag created using a secret key (K) and a message (M) to verify message authenticity: T = MAC(K, M).
- The tag is added to the message by the sender, and the receiver uses the same secret key to verify it.
- Encryption protects the message's confidentiality, and its security depends on the strength (length) of the encryption key.
- A BFA tries all possible keys to break the encryption, with the average number of attempts being $2^{(k-1)}$ for a key of length k.
- In a ciphertext-only attack, the attacker tries different keys to decrypt the message until they find the correct one.

Brute-Force Key Recovery in MAC Systems: Key Size vs Tag Size Considerations

- A MAC is a type of function where many inputs (messages) map to one output (MAC value).
- Assume that the key size (k) is greater than the MAC size (n), meaning the key is longer than the MAC tag.
- If an attacker knows a message (M_1) and its MAC (T_1 = MAC(K, M_1)), they can try different keys (K_i) to see which one produces the same MAC tag.
- The attacker will test 2^k different keys, but there are only 2^n possible unique MAC tags.
- Since there are fewer possible MAC tags than possible keys, many keys will generate the same MAC tag.
- On average, $2^{(k-n)}$ keys will produce the same tag.
- Since there's no way to know which key is correct, the attacker must keep trying different keys until they find the right one.

Brute-Force Key Recovery in MAC Systems: Key Size vs Tag Size Considerations (Contd..)



- **Round 1**

  Given: $M_1, T_1 = \text{MAC}(K, M_1)$

  Compute $T_i = \text{MAC}(K_i, M_1)$ for all 2^k keys

  Number of matches $\approx 2^{(k-n)}$

- **Round 2**

  Given: $M_2, T_2 = \text{MAC}(K, M_2)$

  Compute $T_i = \text{MAC}(K_i, M_2)$ for the $2^{(k-n)}$ keys resulting from Round 1

  Number of matches $\approx 2^{(k-2\times n)}$

Brute-Force Key Recovery in MAC Systems: Key Size vs Tag Size Considerations (Contd..)

- k rounds are needed, where $k = \alpha * n$.

- For example, consider a 80-bit key, 32-bit tag:
  - Round 1 → 2^{48} possible keys
  - Round 2 → 2^{16} possibilities
  - Round 3 → 1 key (correct one).

- If Key length ≤ tag length:
  - The first round may produce a single match, but if not, additional tests with different (message, tag) pairs are required.

- Brute-Force key discovery is as hard or harder than finding the decryption key.

Vulnerability Analysis of a MAC Algorithm Using DES and XOR Operations

- The message ‘M’ is made up of blocks X_1, X_2, X_3, …………………., X_m, each of 64 bits long.
- $\Delta(M) = X_1 \oplus X_2 \oplus X_3 \oplus \ldots\ldots\ldots\ldots \oplus X_m$
- MAC(K, M) = E(K, Δ(M)); where DES is the encryption algorithm used in ECB mode.
- The key ‘K’ is 56 bits long, and the MAC is 64 bits long.
- An attacker trying to find the key 'K' through brute force would need to perform at least 2^{56} DES encryptions to guess the correct key.

Vulnerability Analysis of a MAC Algorithm Using DES and XOR Operations (Contd..)

- The attacker can replace first m-1 blocks of message X_1 through X_{m-1} with new blocks Y1 through Y_{m-1}.
- $Y_m = Y_1 \oplus Y_2 \oplus Y_3 \oplus \ldots\ldots\ldots\ldots \oplus \Delta(M)$
- The attacker can then create a new message with these blocks, using the original MAC tag to make the message appear authentic to the receiver.
- This attack allows the attacker to insert a fraudulent message of length 64*(m−1) bits without knowing the secret key ‘K’.

Security Requirements for MAC Functions Against Potential Attacks

- Even if an attacker knows the MAC function, they should not be able to create a new message M′ that has the same MAC tag as a valid message M (i.e., MAC(K, M') = MAC(K, M)).
- The MAC tag should be random for different messages. If you pick two random messages, the chance that their MAC tags are the same should be very low (specifically, 2^{-n} , where 'n' is the number of bits in the tag).
- If the attacker knows a way to modify a message M to create a new message M' (e.g., by flipping certain bits), the probability that the MAC tags for M and M' are the same should still be very low (again, 2^{-n}).

MACs BASED ON HASH FUNCTIONS

Overview of Hash-based MACs (HMAC)

- There has been an increased interest in hash-based MACs in recent years.
- Cryptographic hash functions are faster in software than block ciphers.
- Hash function libraries are widely available.
- AES and encryption code availability make these benefits less significant, but hash-based MACs are still popular.
- SHA was not designed for use as a MAC because it doesn’t use a secret key.
- HMAC is a popular solution, adding a secret key to hash functions.
- HMAC is standardized in RFC 2104 and required for IPsec and SSL.
- HMAC is also a NIST standard (FIPS 198).

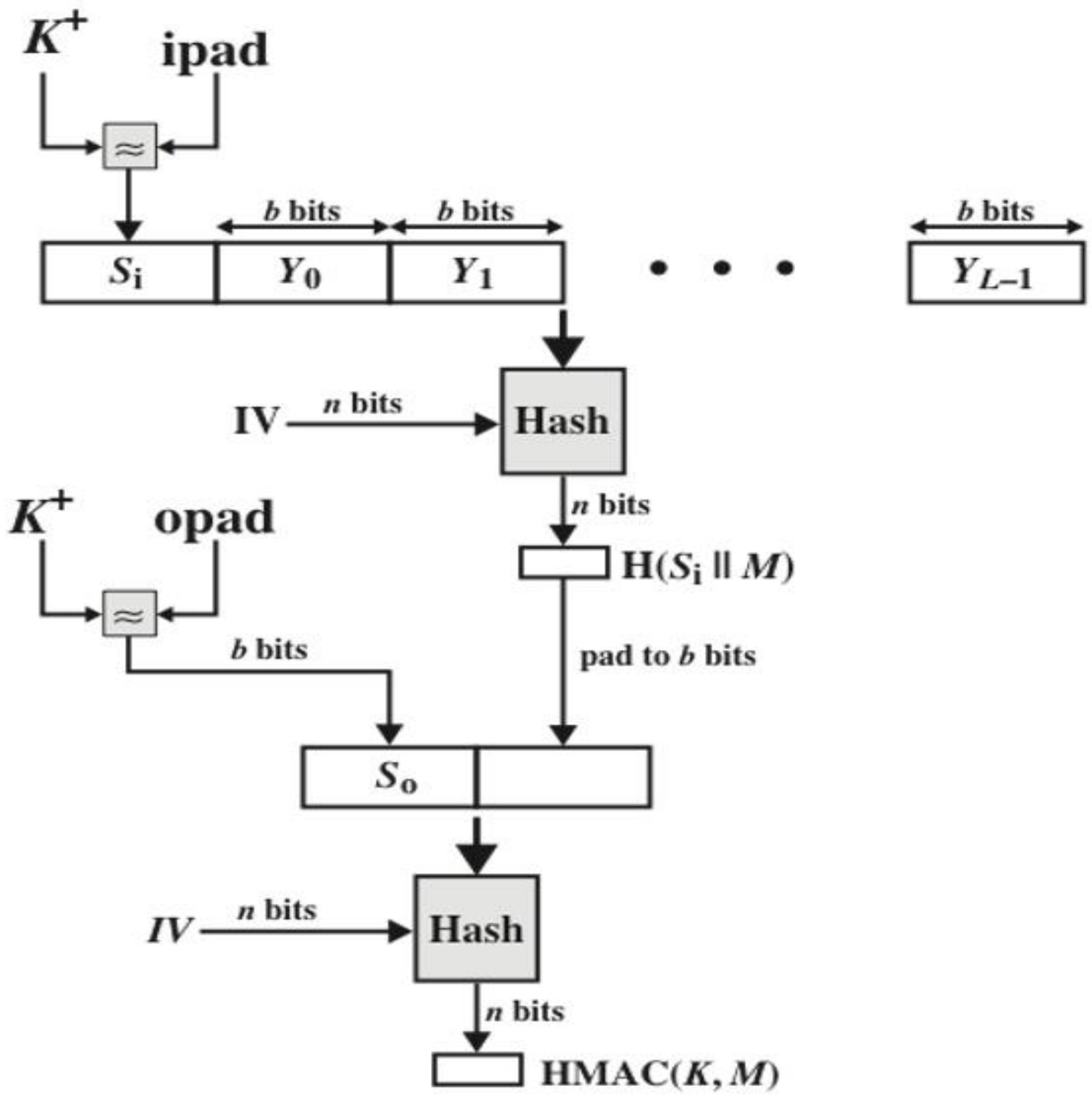
HMAC design objectives listed by RFC 2104

- Use hash functions that work well in software and have freely available code.
- Allow easy replacement of the hash function if a faster or more secure one is needed.
- Preserve the original performance of the hash function without significant slowdown.
- Handle keys in a simple and straightforward manner.
- Ensure there is a clear understanding of the system's security, based on reasonable assumptions about the hash function.

Benefits of following the HMAC design objectives

- An existing hash function implementation can be used as a module in HMAC, so most of the HMAC code is ready to use without changes.
- If a hash function must be replaced, the old one can be replaced by a new one, without major changes to the code.
- This allows us to replace a slower hash function with a faster one or upgrade to a more secure one (e.g., switching from SHA-2 to SHA-3) if the security is compromised.
- HMAC can be proven secure if the embedded hash function has strong cryptographic properties.

HMAC Structure



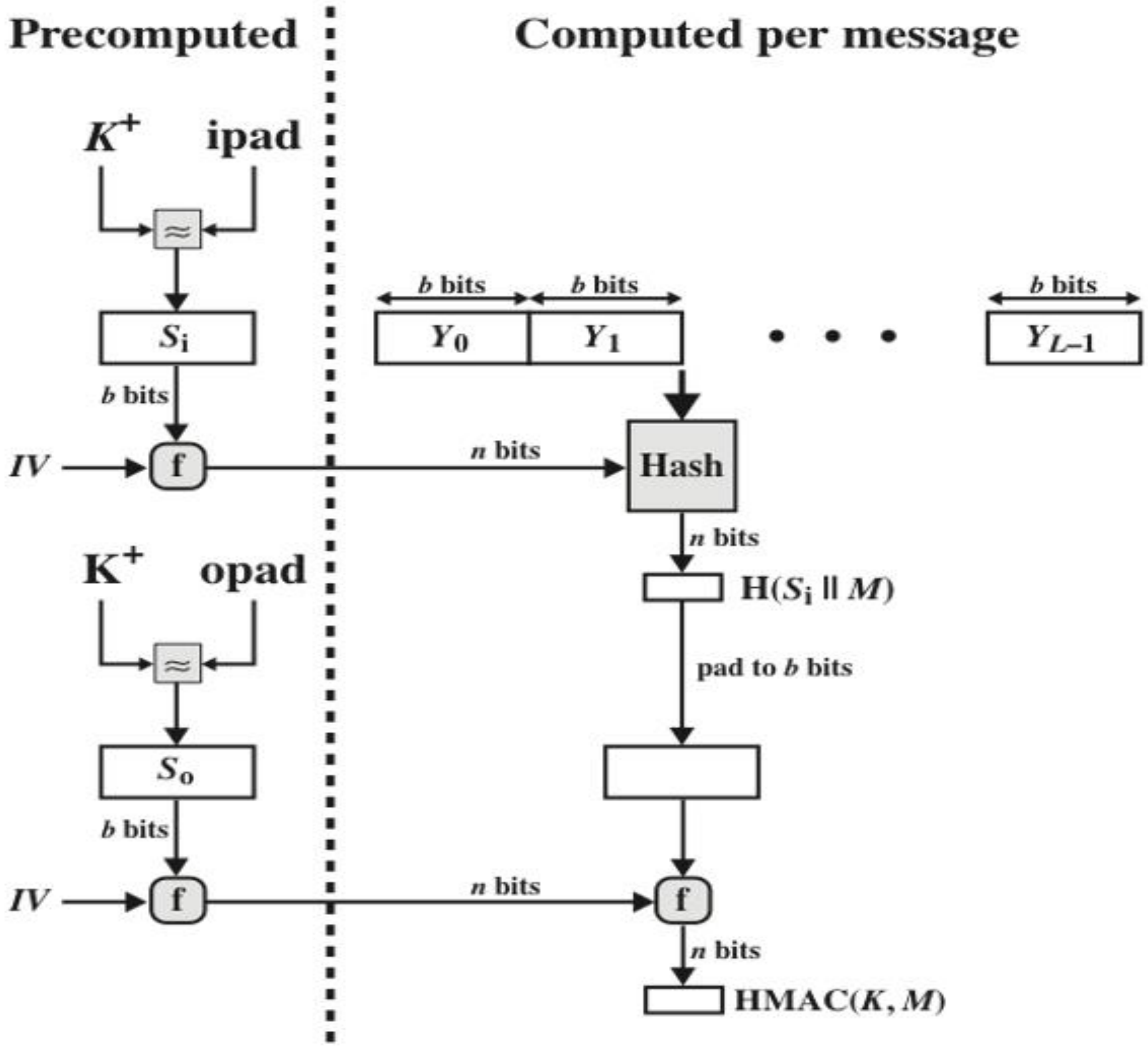
HMAC Structure Notations

- H = embedded hash function (e.g., MD5, SHA-1, RIPEMD-160)
- IV = initial value input to hash function
- M = message input to HMAC (including the padding specified in the embedded hash function)
- Y_i = ith block of M; $0 \leq i \leq (L - 1)$
- 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; recommended length is ≥ n; if key length is greater than b, the key is input to the hash function to produce an n-bit key
- K^+ = K padded with zeros on the left so that the result is ‘b’ bits in length
- ipad = 0x36, repeated b/8 times.
- opad = 0x5C, repeated b/8 times.

HMAC Algorithm

- ➢ $\mathrm{HMAC}(K, M) = H[(K^{+} \oplus \mathrm{opad}) \| H[(K^{+} \oplus \mathrm{ipad}) \| M]]$
- Append zeros to the left end of K to create a b-bit string K^{+} (e.g., if K is of length 160 bits and b = 512, then K will be appended with 44 zeroes).
- XOR K^{+} with ipad to produce the b-bit block S_i.
- Append M to S_i.
- Apply H to the stream generated in step 3.
- XOR K^{+} with opad to produce the b-bit block S_o.
- Append the hash result from step 4 to S_o.
- Apply H to the stream generated in step 6 and output the result.

Efficient Implementation of HMAC



Efficient Implementation of HMAC (Contd..)

- Two precomputed quantities are involved: f(IV, ($K^+ \oplus$ ipad)), f(IV, ($K^+ \oplus$ opad))
- f(cv, block) is the compression function used in the hash algorithm.
- Inputs to f(cv, block): a chaining variable (‘n’ bits) and a block of data (‘b’ bits).
- Output of f(cv, block): a new chaining variable (‘n’ bits).
- Only one extra instance of the compression function is used compared to that of the normal hash process.
- This is especially useful for short messages in MAC computations.

Security of HMAC

- The security of HMAC depends on the strength of the hash function used in it.
- HMAC’s designers have proven a direct link between the strength of the hash function and HMAC's security.
- The security of a MAC is measured by how hard it is for an attacker to forge a valid tag.
- The risk of forgery depends on how much time the attacker spends and how many message-tag pairs they've seen.

Compression Function attack on HMAC

- Attacker tries to compute the hash output even if the IV (initial value) is random, secret, and unknown.

- The attacker tries to break the hash function by guessing the secret IV (initial value), which is ‘n’ bits long.

- The attacker could either:
- Use a BFA, which would take time proportional to 2^n.
- Use a birthday attack (a faster method to find collisions).

Collision attack on HMAC

- The attacker tries to find two different messages (M and M′) that give the same hash.
- This requires an effort of $2^{n/2}$ for an n-bit hash function (e.g., for 128-bit hash, 2^{64} attempts).
- Since the attacker doesn't know the secret key (K) used in HMAC, they must observe enough message-tag pairs generated with the same key to try to find a collision.
- For a 128-bit hash (like MD5), the attacker would need to observe 2^{64} blocks of data (about 150,000 years on a 1 Gbps link).