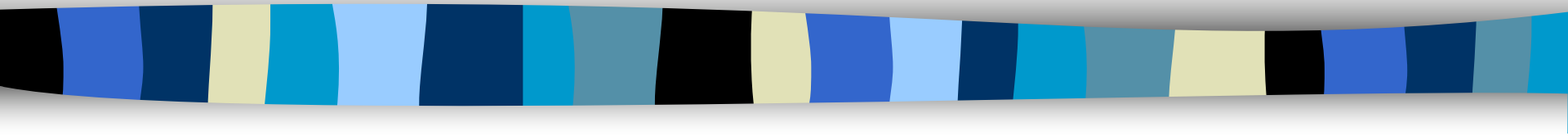
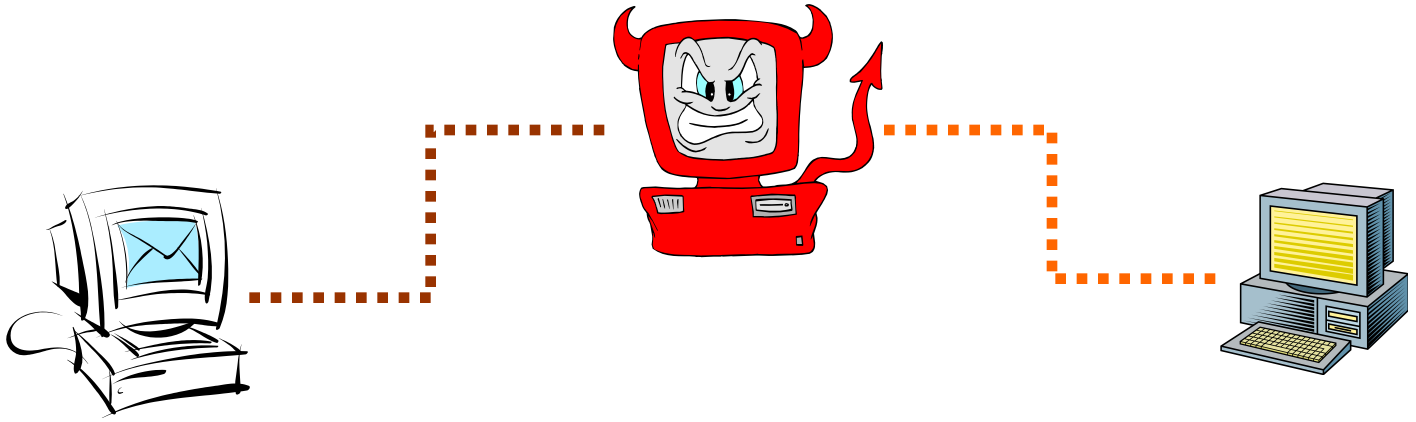
Information Security
CENG418
week-3



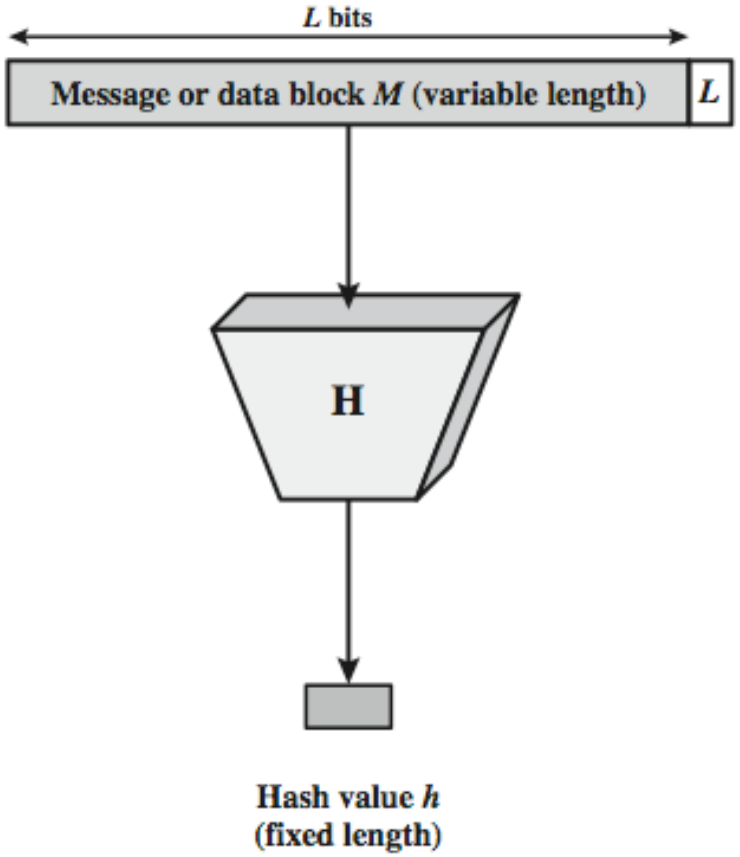
Cryptography: Cryptographic Hash Function and Message Authentication Code and Secure Channel

Data Integrity and Source Authentication



- Encryption does not protect data from modification by another party. Encryption only satisfies the secrecy requirement.
- **Need a way to ensure that data arrives at destination in its original form as sent by the sender and it is coming from an authenticated source.**

Cryptographic Hash Function



- A hash function maps a message of an arbitrary length to a m-bit output
  - output known as the fingerprint or the message digest
  - if the message digest is transmitted **securely**, then changes to the message can be detected
- A hash function is a many-to-one function, so collisions can happen.

Ref: *W.Stallings"Cryptography and Network Security, Chapter 11, Chapter 13, Fifth Edition", Lecture slides by Lawrie Brown.*

Hashing

- is based on the idea of one way function:

$$f(x) = g^x \, mod \, p$$

This is called *modular exponentiation*.

Read:
https://en.wikipedia.org/wiki/Modular_exponentiation

Security Requirements for Cryptographic Hash Functions

Given a function h:X $\rightarrow$Y, then we say that h is (X is the message set of class and Y is the set of hash class) :

- preimage resistant (one-way):

  if given y $\in$Y it is computationally infeasible to find a value x $\in$X s.t. h(x) = y

- 2-nd preimage resistant (weak collision resistant):

  if given x $\in$ X it is computationally infeasible to find a value x' $\in$ X, s.t. x'$\neq$x and h(x') = h(x)

- collision resistant (strong collision resistant):

  if it is computationally infeasible to find two distinct values x',x $\in$ X, s.t. h(x') = h(x)

Bruteforce Attacks on Hash Functions

- Attacking one-wayness
  - Goal: given h:X$\rightarrow$Y, y$\in$Y, find x such that h(x)=y
  - Algorithm:
    - pick a random value x in X, check if h(x)=y, if h(x)=y, returns x; otherwise iterate
    - after failing q iterations, return fail
  - The average-case success probability is

$$\varepsilon = 1 - \left(1 - 1/|Y|\right)^q \approx \frac{q}{|Y|}$$

  - Let $|Y|=2^m$, to get ε to be close to 0.5, $q \approx 2^{m-1}$

Bruteforce Attacks on Hash Functions

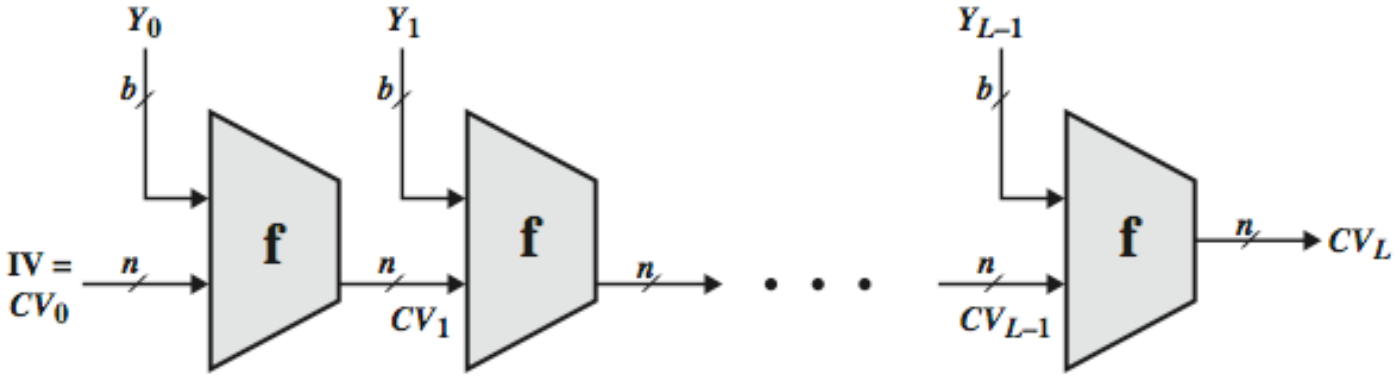
- Attacking collision resistance
  - Goal: given h, find x, x' such that h(x)=h(x')
  - Algorithm: pick a random set X_0 of q values in X
    for each $x \in X_0$, computes $y_x=h(x)$
    if $y_x=y_{x'}$ for some $x' \neq x$ then return (x,x') else fail
  - The average success probability is

$$1-e^{-\frac{q(q-1)}{2|Y|}}$$

  - Let $|Y|=2^m$, to get ε to be close to 0.5, $q \approx 2^{m/2}$
  - This is known as the birthday attack.

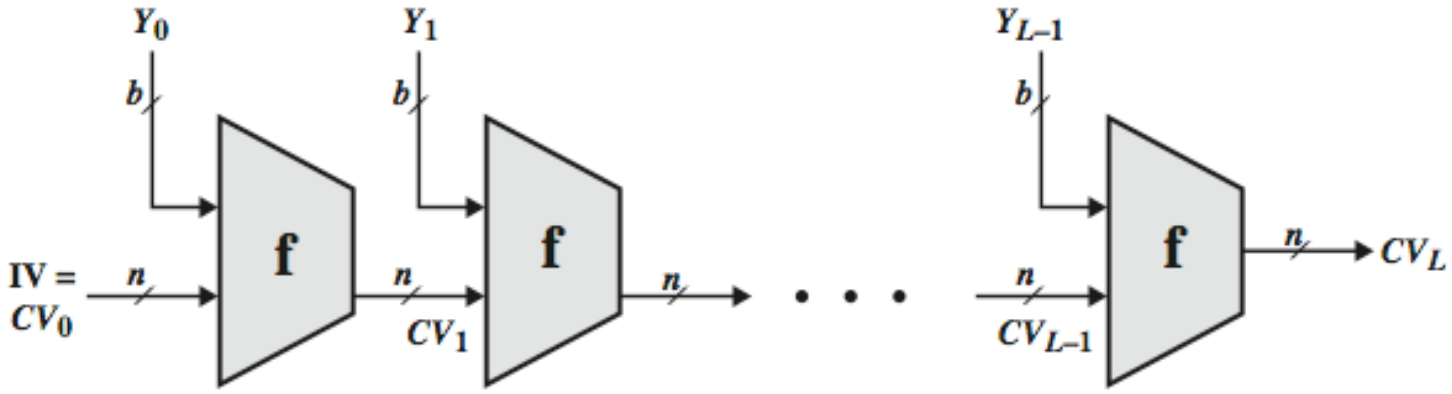
Hash Function Cryptanalysis

- Cryptanalytic attacks exploit some property of algorithms so faster than exhaustive search.
- Hash functions use iterative structure
  - process message in blocks (include length)
- The hash algorithm involves repeated use of a compression function, *f*, that takes two input blocks of *b* bits each and produces an *n*-bit output.
- The final block includes the value of the total length of the input to the hash function. The inclusion of the length makes the job of the attacker more difficult.
- Attacks focus on collisions in function **f.**



Hash Function Cryptanalysis

- This was proposed by Merkle and is the structure of most hash functions in use today.
- Cryptanalysis of hash functions focuses on the internal structure of **f** and is based on attempts to find efficient techniques for producing collisions for a single execution of **f**.
- Once that is done, the attack must take into account the fixed value of IV.
- The attack on **f** depends on exploiting its internal structure.
- The attacks that have been mounted on hash functions are rather complex and beyond our scope here.



Secure Hash Algorithm

- MD5
  - output 128 bits
  - collision resistance completely broken by researchers in China
- SHA originally designed by NIST & NSA in 1993
- was revised in 1995 as SHA-1
- US standard for use with DSA signature scheme
  - standard is FIPS 180-1 1995, also Internet RFC3174
  - nb. the algorithm is SHA, the standard is SHS
- based on design of MD4 with key differences
- produces 160-bit hash values
- recent 2005 results on security of SHA-1 have raised concerns on its use in future applications

Revised Secure Hash Standard

- NIST issued revision FIPS 180-2 in 2002
- adds 3 additional versions of SHA-2
  - SHA-256, SHA-384, SHA-512
- designed for compatibility with increased security provided by the AES cipher
- structure & detail is similar to SHA-1
- hence analysis should be similar
- but security levels are rather higher

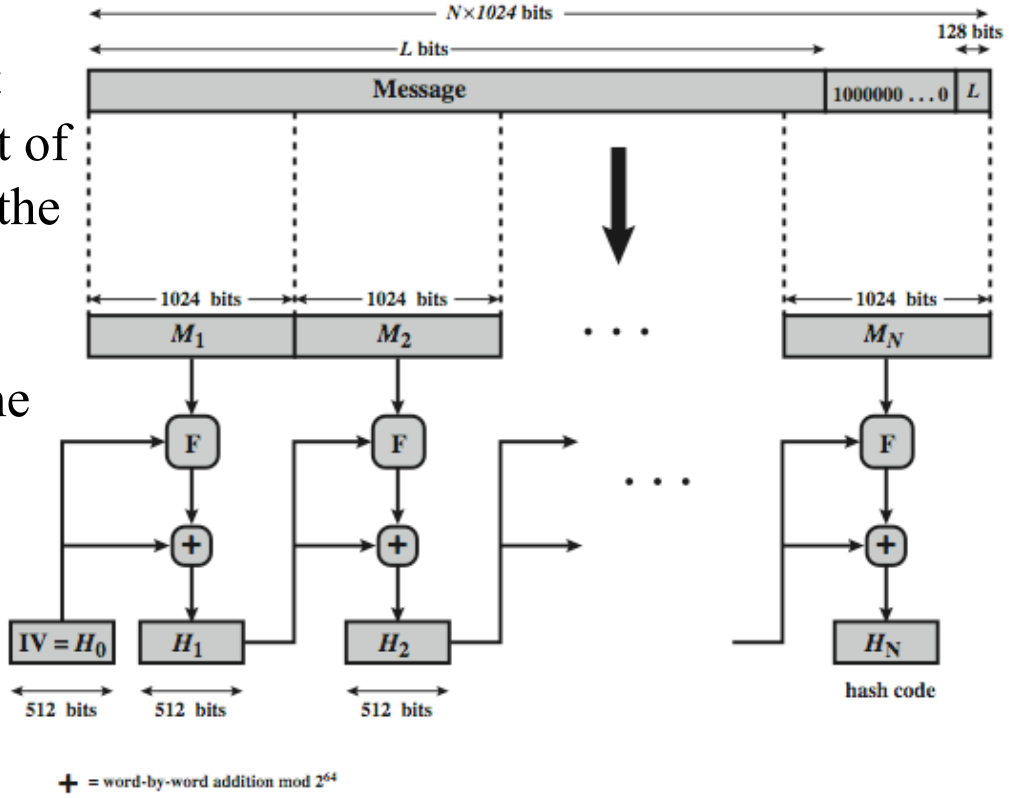
SHA Versions

		SHA-2		
	SHA-1	**SHA-256**	**SHA-384**	**SHA-512**
Message digest size	160	256	384	512
Message size	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Block size	512	512	1024	1024
Word size	32	32	64	64
Number of steps	80	64	80	80

SHA-512 Overview

Step 1: Append padding bits, consists of a single 1-bit followed by the necessary number of 0-bits, so that its length is congruent to 896 modulo 1024

- Step 2: Append length as an (big-endian) unsigned 128-bit integer
- Step 3: Initialize hash buffer to a set of 64-bit integer constants
- Step 4: Process the message in 1024-bit (128-word) blocks, which forms the heart of the algorithm. Each round takes as input the 512-bit buffer value H_i, and updates the contents of that buffer.
- Step 5: Output the final state value as the resulting hash



Simple Python Implementation of Sha-512

- Read and try the following Python implementation to understand the SHA algorithm:

- https://gist.github.com/illia-v/7883be942da5d416521375004cecb68f

SHA-3

- SHA-1 not yet "broken"
  - but similar to broken MD5 & SHA-0
  - so considered insecure
- SHA-2 (esp. SHA-512) seems secure
  - shares same structure and mathematical operations as predecessors so have concern
- NIST announced in 2007 a competition for the SHA-3 next gen NIST hash function
  - goal to have in place by 2012 and then fixed
  - SHA-3 NIST Standard: http://csrc.nist.gov/groups/ST/hash/sha-3/sha-3_standardization.html
  - (FIPS) 180-3, Secure Hash Standard for SHA-3.

SHA-3 Requirements

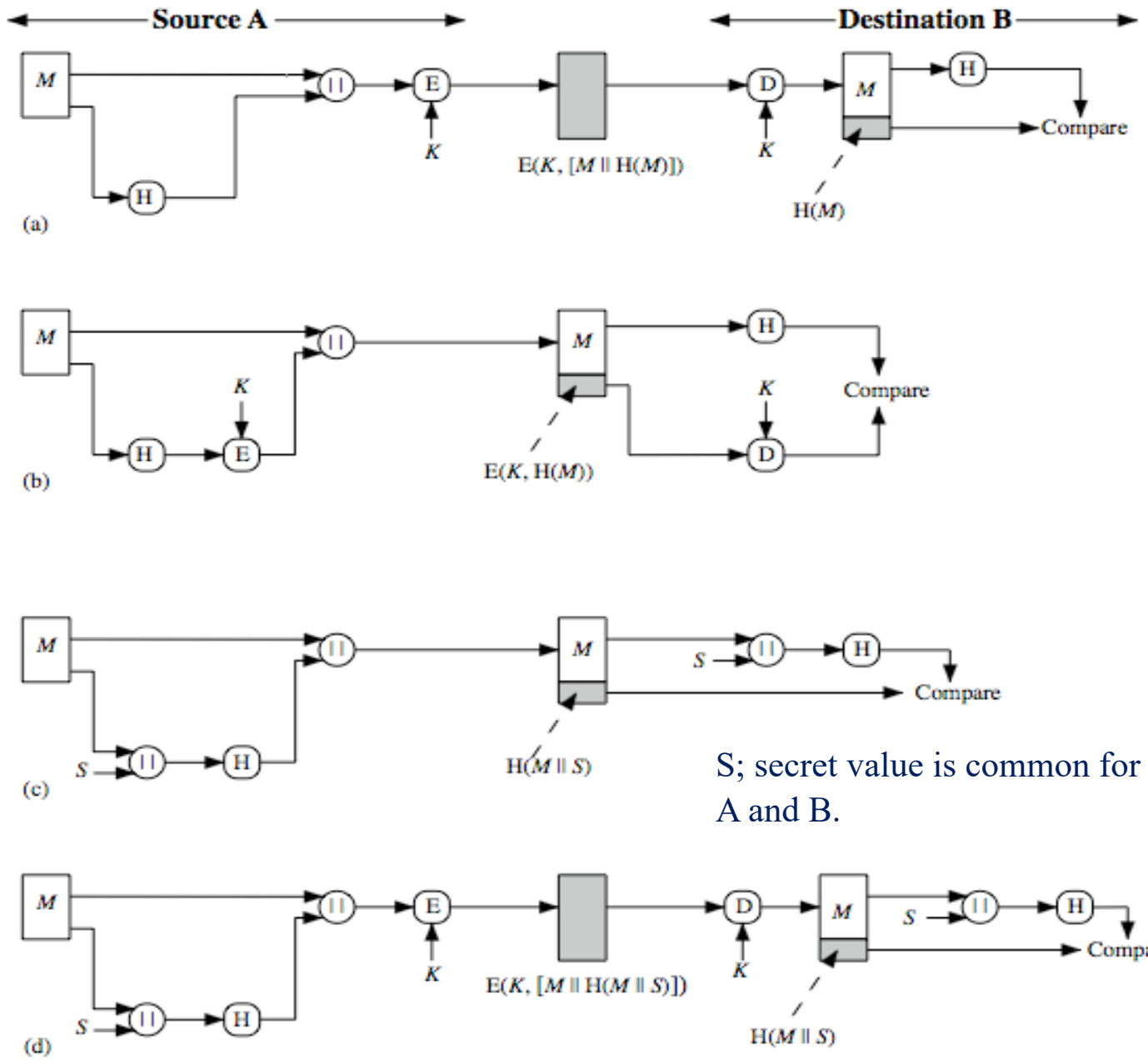
- replace SHA-2 with SHA-3 in any use
  - so use same hash sizes; 224, 256, 384, and 512 bits
- preserve the online nature of SHA-2
  - so must process small blocks (512 / 1024 bits)
- evaluation criteria
  - security close to theoretical max for hash sizes
  - cost in time & memory efficient over a range of hardware platforms
  - Algorithm and implementation characteristics: such as flexibility & simplicity

Uses of hash functions

- Software integrity
  - E.g., tripwire
- Timestamping
  - How?

- Message authentication
- One-time Passwords
- Digital signature

Hash Functions & Message Authentication

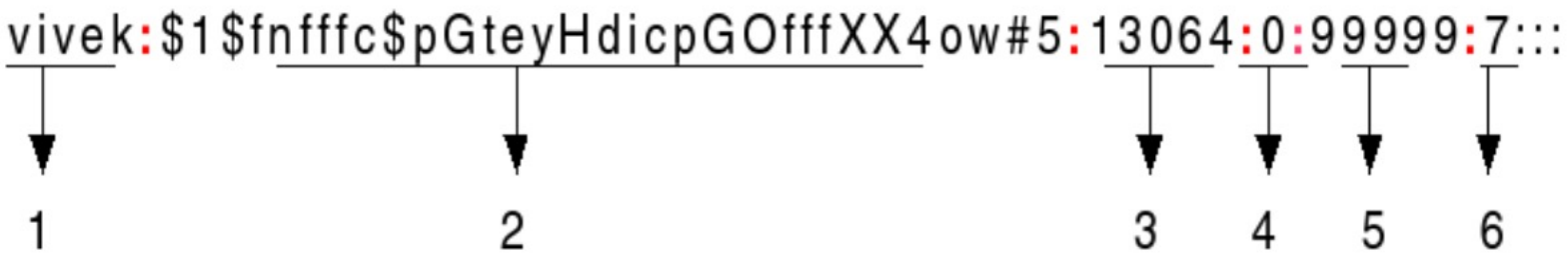


S; secret value is common for A and B.

Other Hash Function Uses

- to create a one-way password file
  - store hash of password not actual password
- for intrusion detection and virus detection
  - keep & check hash of files on system
- pseudorandom function (PRF) or pseudorandom number generator (PRNG)
  - a common application for a hash-based PRF is for the generation of symmetric keys.

/etc/shadow file in Linux



1. **Username** : A valid account name, which exist on the system.
2. **Password** : Your encrypted password is in hash format. The password should be minimum 15-20 characters long including special characters, digits, lower case alphabetic and more. Usually password format is set to `$id$salt$hashed`, The `$id` is the algorithm used On GNU/Linux as follows:
    1. `$1$` is MD5
    2. `$2a$` is Blowfish
    3. `$2y$` is Blowfish
    4. `$5$` is SHA-256
    5. `$6$` is SHA-512

Coming Attractions …

- **Cryptography:** Message Authentication Code.

Limitation of Using Hash Functions for Authentication

- Require an authentic channel to transmit the hash of a message
  - anyone can compute the hash value of a message, as the hash function is public
  - not always possible
- How to address this?
  - use more than one hash functions
  - use a key to select which one to use
- Suppose receiver “Bob”, also with the key K, receives message m’ and tag T. Bob uses the MAC verification algorithm to verify that T is a valid MAC under key K for message m’.

Hash Family - MAC

- A hash family is a four-tuple (X, Y, K, H), where
  - X is a set of possible messages
  - Y is a finite set of possible message digests
  - K is the keyspace
  - For each $K \in K$, there is a hash function $h_K \in H$. Each h_K: $X \rightarrow Y$
- Alternatively, one can think of H as a function $K \times X \rightarrow Y$

Message Authentication Code

- A MAC scheme is a hash family, used for message authentication
- MAC = $C_K(M)$
- The sender and the receiver share K
- The sender sends (M, $C_k(M)$)
- The receiver receives (X,Y) and verifies that $C_K(X)=Y$, if so, then accepts the message as from the sender
- To be secure, an adversary shouldn't be able to come up with (X',Y) such that $C_K(X)=Y$.

The Ideal MAC and MAC Security

- The notion of an ideal MAC function, which is very similar to the notion of an ideal block cipher.
- This definition is preferred because it encompasses a broad range of attacks; weak key attacks, related-key attacks and more.
- **Definition:** *An ideal MAC function is a random mapping from all possible inputs to n-bit output.*
  - The key K, is not known by the attacker. There could be a weakness in the rest of the system that provides partial information about K to the attacker.

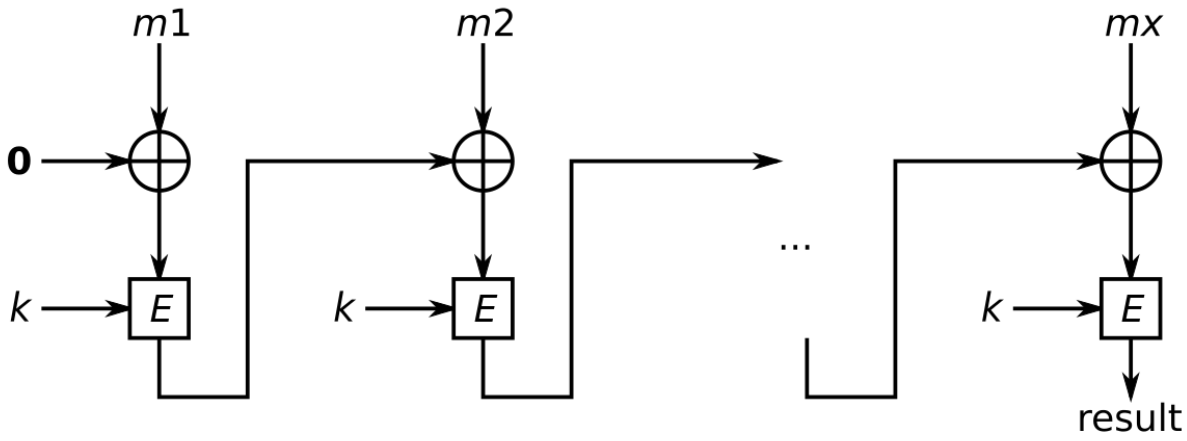
Example of Insecure Hash Families

- Let h be a one-way hash function
- H(K,M) = h(K || M), where || denote concatenation
  - Insecure as MAC
  - Given M and a=h(K || M), can compute M'=M||… and a', such that h(K||M') = a'
- H(K,M) = h(M || M),
  - Also insecure as MAC

CBC-MAC

- CBC-MAC:

$$H_0 = IV$$
$$H_i = E_K(P_i \oplus H_{i-1})$$
$$MAC = H_k$$



  - The most common definition of CBC-MAC requires the IV to be fixed at 0.
  - In general, one should never use the same key for both encryption and authentication. Using the same key for both can lead to privacy compromises for CBC encryption and authenticity compromises for CBC-MAC.
- There are number of different collision attacks on CBC-MAC that effectively limit the security to half the length of the block size.

CBC-MAC

- A simple collision attack example:
- Let M be a CBC-MAC function.
- If we know M(a)=M(b) then we also know that M(a||c)=M(b||c).
- This due to the structure of CBC-MAC. Let's illustrate this with a simple case: c consist of a single block:

$$M(a \| c) = E_K(c \oplus M(a))$$
$$M(b \| c) = E_K(c \oplus M(b))$$
$$and \qquad M(a) = M(b)$$

CBC-MAC and CMAC

Attack proceeds in two stages:

1. The attacker collects the MAC values of a large number of messages until a collision occurs.
   - This takes 2^{64} steps for 128-bit block cipher because the birthday paradox.
   - This provides a and b for which M(a)=M(b).
2. If the attacker can now get the sender to authenticate a||c, he can replace the message with b||c without changing the MAC value.

The receiver will check the MAC and accept the bogus message b||c.

- CBC-MAC would be fine if we could use a block cipher with a 256-bit block.

CBC-MAC and CMAC

If you wish to use CBC-MAC, you should instead do following:

1. Construct a string *s* from the concatenation of *l* and *m*, where *l* is the length of message *m* encoded in a fixed length format.
2. Pad *s* until the length is a multiple of the block size.
3. Apply CBC-MAC to the padded string *s*.
4. Output the last cipher text block, or part of the block. Do not output any of the intermediate values.

The **advantages of CBC-MAC** is that it uses the same type of computations as the block cipher encryption modes. In many system encryption and MAC are applied on the bulk data. These are **speed critical areas**. Due to the same primitive functions makes efficient implementations easier, especially in hardware.

CBC-MAC and CMAC

- CMAC is based on CBC-MAC and was recently standardized by NIST.
- CMAC works almost exactly like CBC-MAC, except it treats the last block differently.
  - CMAC, XORs one of two special values into the last block prior to the last block cipher encryption.
  - The special values are derived from the CMAC key.

HMAC

- HMAC computes $h(K \oplus a \parallel K \oplus b \parallel m)$, where a and b are specified constants.
- The message itself is only hashed once, and the output hashed again with the key.
- HMAC-SHA-256 to 128 bits should be safe.

HMAC: Constructing MAC from Cryptographic Hash Functions

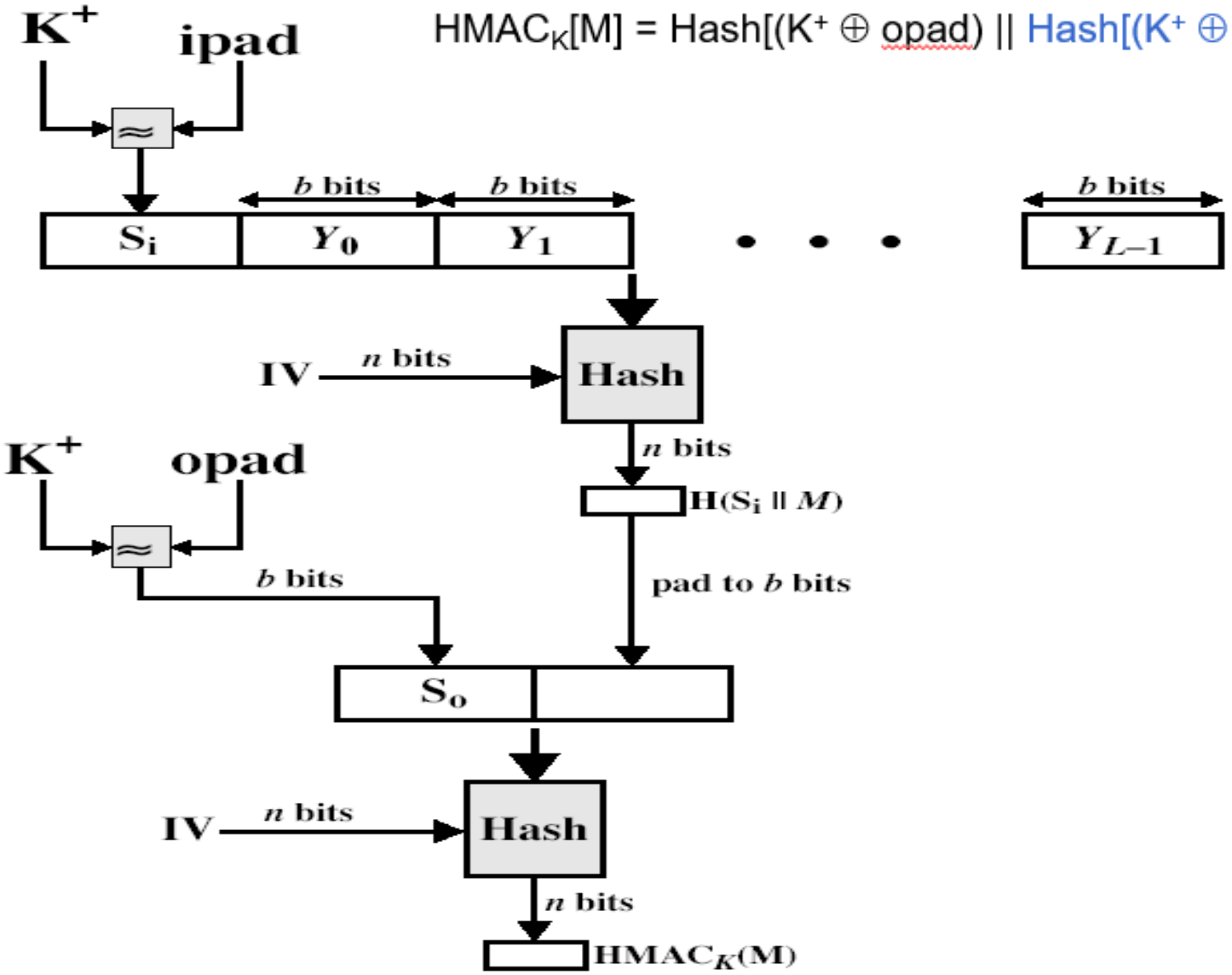
$$h(K \oplus a \parallel K \oplus b \parallel m),$$

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

- K^+ is the key padded (with 0) to B bytes, the input block size of the hash function
- a = opad = the byte 0x5C repeated B times.
- b= ipad = the byte 0x36 repeated B times.

HMAC Overview

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



HMAC Security

- If used with a secure hash functions (e.g., SHA-256) and according to the specification (key size, and use correct output), no known practical attacks against HMAC

SECURE CHANNEL

Properties of a Secure Channel

- The problem is defined as creating a secure connection between Alice and Bob.

1. Roles:

- The most connections are bidirectional.
  - In real systems, one party can be a client and the other is server. (one initiator and one responder).
  - Alice and Bob might be the same person, and the transmission medium could be a backup device or a USB stick.
- There is always Eve, who tries to attack the secure channel in any way possible.
  - Eve can read all of the communications and arbitrarily manipulate these communications. Eve can delete, insert or modify data that is being transmitted.

Properties of a Secure Channel

2. Key:

- To implement a secure channel we need a shared secret key.
  - The key is K known only to Alice and Bob.
  - Every time the secure channel is initialized, a new value is generated for the key K. Here a key negotiation protocol is necessary to refresh the secret key K periodically.
  - Alice and Bob assume that knowledge of K is restricted preferably to their selves.

Properties of a Secure Channel

3. Message or Stream

- The communication between parties can be as a sequence of discrete messages (such as emails) or as a continuous stream of bytes (such as streaming media).
- We will consider only discrete messages. These can be converted to handle a stream of bytes by cutting the data stream into separate messages and reassembling the stream at receiver's end.

Properties of a Secure Channel

4.Security Properties

- Alice sends a sequence of messages $m_1, m_2, \ldots$
- There are processed by the secure channel algorithms
- Bob processes the received message $m'_1, m'_2, \ldots$

The following properties are hold:

i. Secrecy; Eve should not learn anything about mesages.

   i. Eve can see the size and timing of messages over channel

   ii. Eve can find out who is communicating with whom, how much and when. (Traffic Analysis; as well known problem for SSL/TLS, IPsec, and SSH).

Properties of a Secure Channel

ii. Bob only gets proper messages with in their correct order. There are no duplicates, modified messages and no bogus messages sent by someone other then Alice.

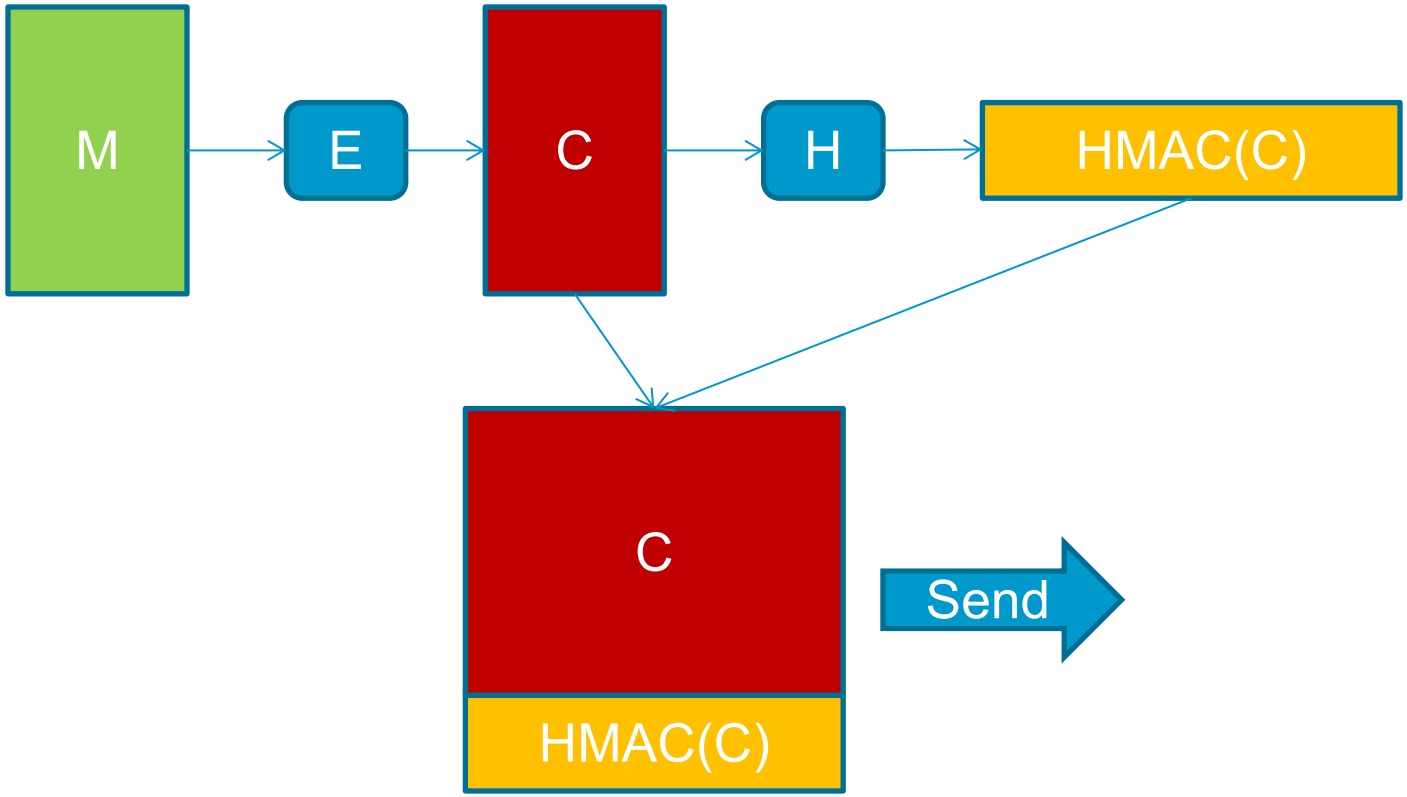
iii. Bob should exactly learn which messages are missed. Alice wants to ensure that Bob gets all the messages she sent him.

Secure Channel description do not cover this third requirement. Message acknowledgment and resend are standard communication protocol techniques.

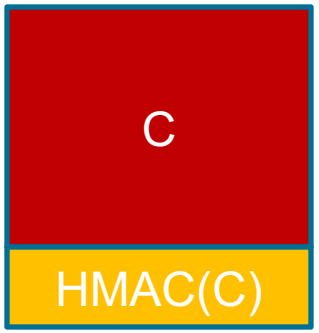
Encryption and Authentication

- Three ways for encryption and authentication
  - **Authenticate-then-encrypt** (AtE), Authenticate first and then encrypt both the message and the MAC value. Used in SSL
    - a = MAC(x), C=E(x,a), transmit C
  - **Encrypt-then-authenticate** (EtA), We can encrypt first then authenticate the cipher text. Used in IPSec
    - C=E(x), a=MAC(C), transmit (C,a)
  - **Encrypt-and-authenticate** (E&A), Both encrypt the message and authenticate the message and then combine (concatenate) the two results. Used in SSH
    - C=E(x), a=MAC(x), transmit (C,a)
- Which way provides secure communications when embedded in a protocol that runs in a real adversarial network setting?
- There is no simple answer for which method is the best.

Encrypt then Authenticate



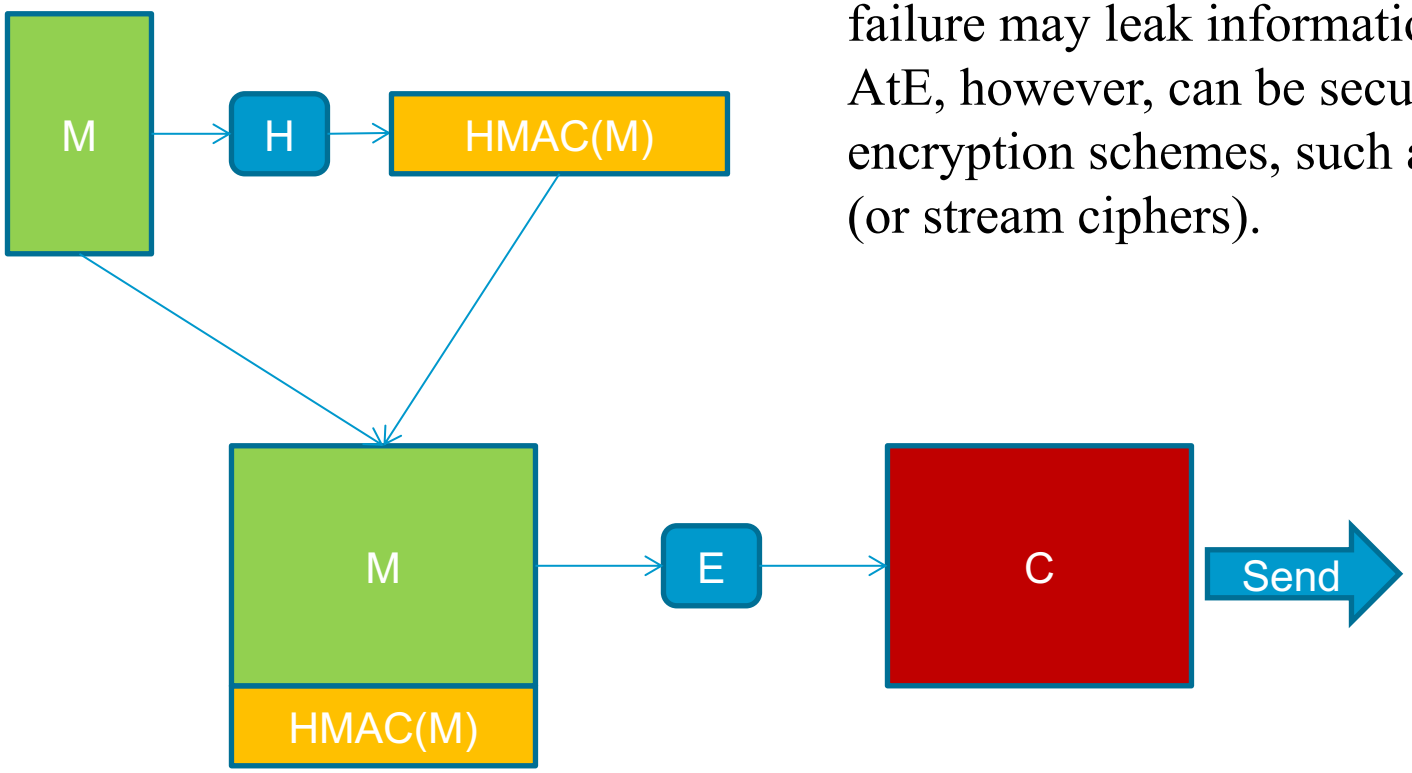
Encrypt Then Authenticate



- There are two arguments in favor encryption first:

1. Applying the MAC to the cipher-text of such a weak encryption scheme fixes it and makes it secure.
2. It is more efficient in discarding bogus messages.
   - Normally Bob has to both decrypt the message and check the authentication. With encrypt first, the decryption is done last on the receiver side, and Bob never has to decrypt bogus messages, since he can identify and discard them before decryption.

Authenticate then Encrypt



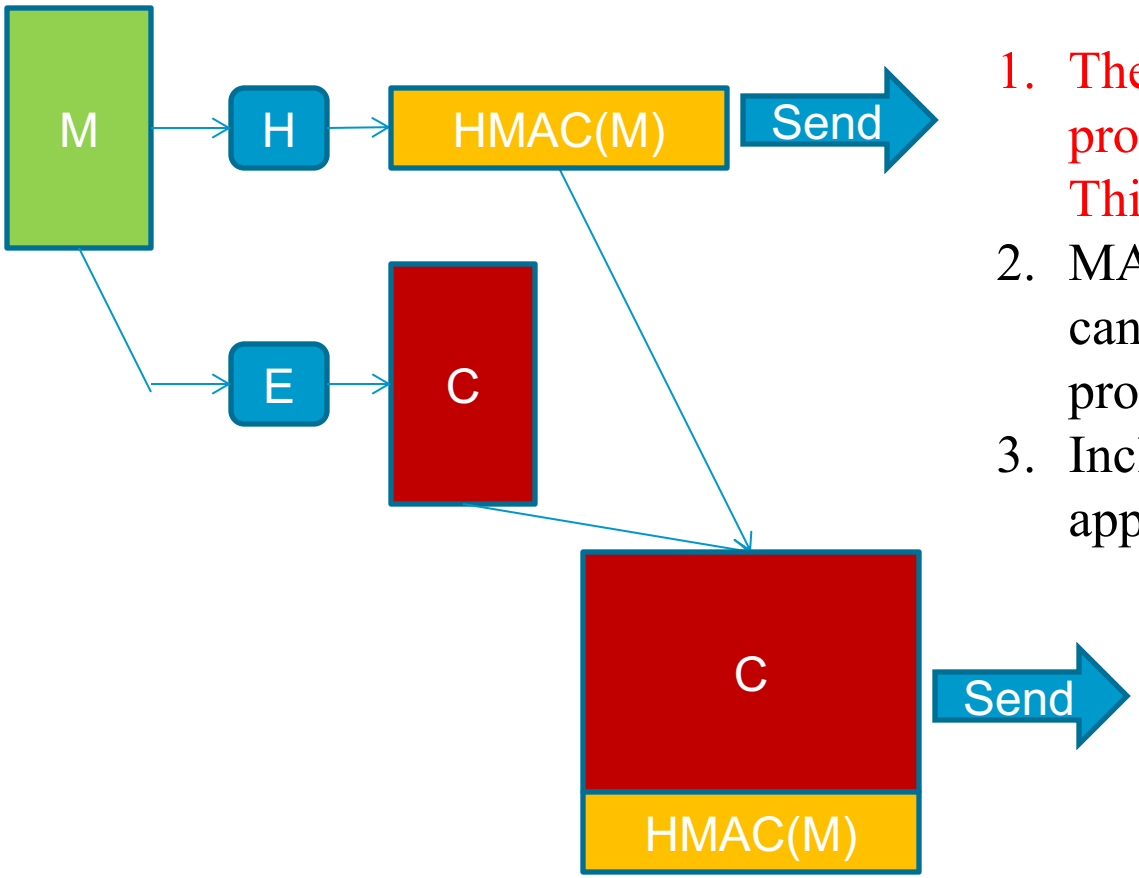
Authenticate-then-encrypt (AtE) is not always secure:

a = MAC(M), C=E(M,a), transmit C

As first step is decryption, its success or failure may leak information.

AtE, however, can be secure for some encryption schemes, such as CBC or OTP (or stream ciphers).

Encrypt and Authenticate



1. The encryption and authentication processes can happen in parallel. This can increase performance.
2. MAC is not encrypted and attacker can view it. MACs are designed to protect authenticity not privacy.
3. Including nonce value, this approach can be more secure.

Important Question!

- Imagine, how much damage Eve could do if she could read all the traffic. Then think about how much damage Eve could do if she could modify the data being communicated.
- Which one is more important; encryption or authentication?

Important Question!

- In many cases, one can argue that authentication is more important than encryption.
- We therefore prefer to expose the encryption function to Eve's direct attacks, and protect the MAC as much as possible.
- In most cases, modifying data is a devastating attack, and does more damage than merely reading it.

Next Week

- Key Exchange on Unsecure Networks - DHKE
- Asymmetrical Cryptosystems – RSA and ELGAMAL
- Identification and Non-Repudiation
  - PKI and Digital Signature (DSA – Digital Signature Algorithm examples)

Homework

- Design a scheme based on SHA-512 and timestamp information to verify a password that has expiration.
- Initially H(password,t0)=a is stored. a also includes time information t0.
- verify(password) checks whether H(password,t1)==a. If passwords are same and |t1-t0|<3 months, hash values should be the same.