

# Lecture 6: Symmetric Cryptography

---

CS 181S

September 24, 2018

# The Big Picture Thus Far...

Attacks  
are perpetrated by  
threats  
that inflict  
harm  
by exploiting  
vulnerabilities  
which are controlled by  
countermeasures.

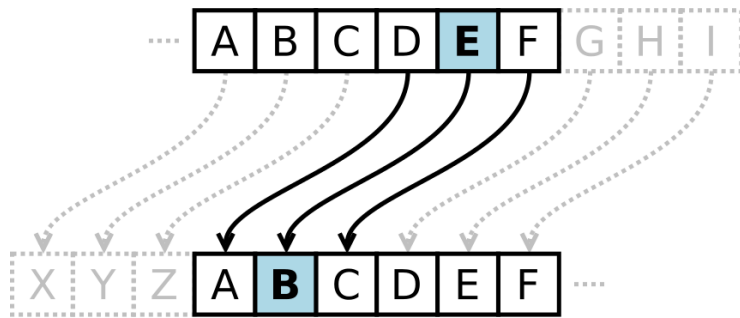
# Purpose of Encryption

- **Threat:** attacker who controls the network
  - can read, modify, delete messages
  - in essence, the attacker *is* the network
  - *Dolev-Yao model* [1983]

# Purpose of encryption

- **Threat:** attacker who controls the network
  - can read, modify, delete messages
  - in essence, the attacker *is* the network
  - *Dolev-Yao model* [1983]
- **Harm:** messages containing secret information disclosed to attacker (violating confidentiality)
- **Vulnerability:** communication channel between sender and receiver can be read by other principals
- **Countermeasure:** encryption

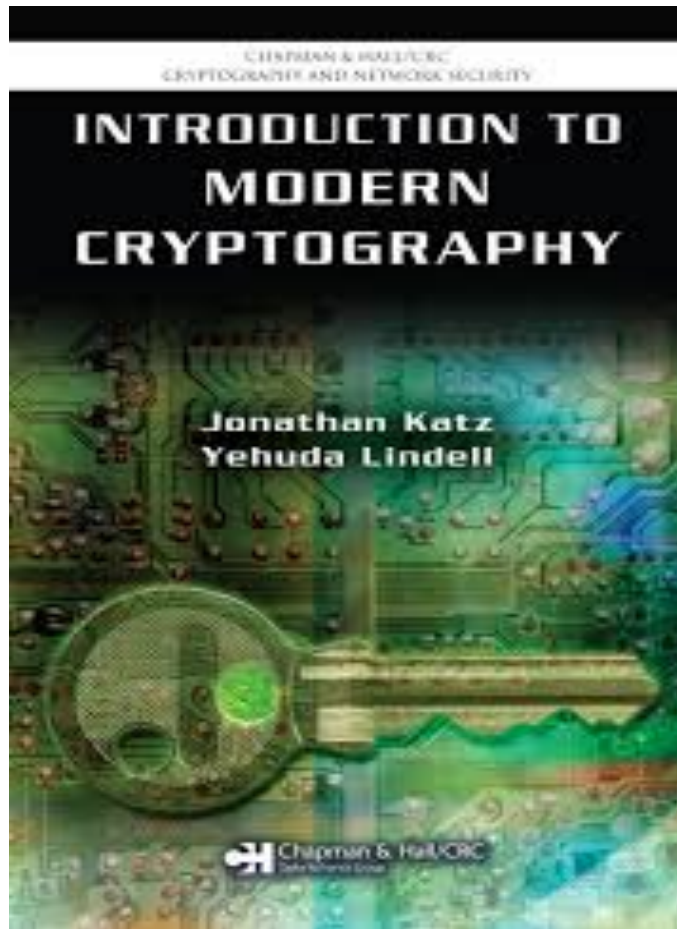
# Classical Cryptography



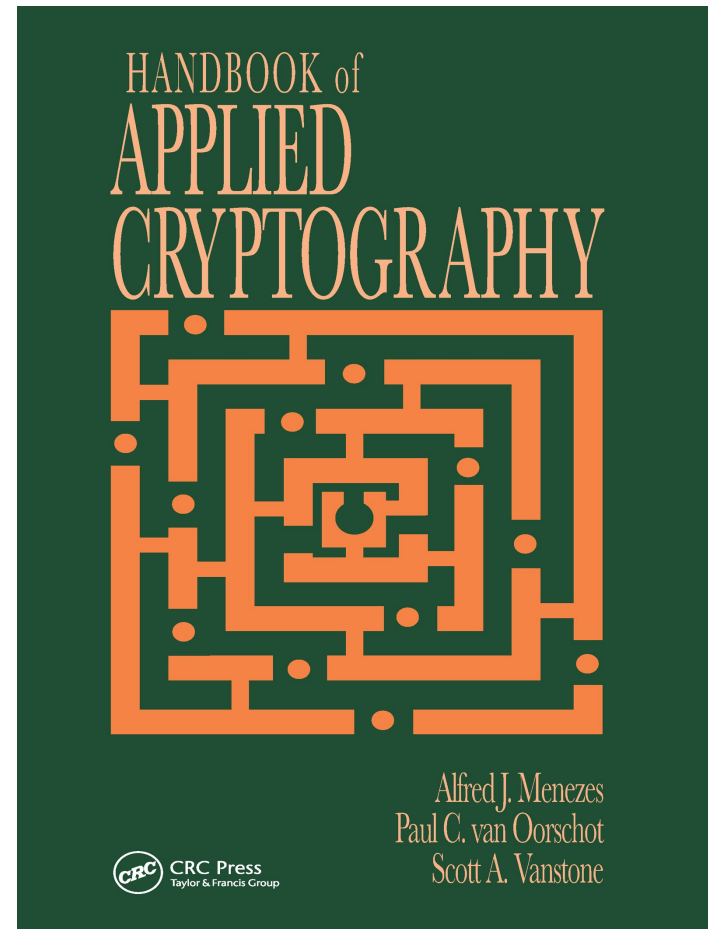
# Kerckhoffs' Principle

- Secrecy should depend upon the key remaining secret
- Secrecy should **not** depend upon the algorithm remaining secret
- Instance of Open Design
- Proprietary encryption schemes are to be avoided
  - Just google "proprietary encryption broken"

# Cryptography



cf. CS 4830/6830



cf. CS 6832

# Tenants of modern cryptography

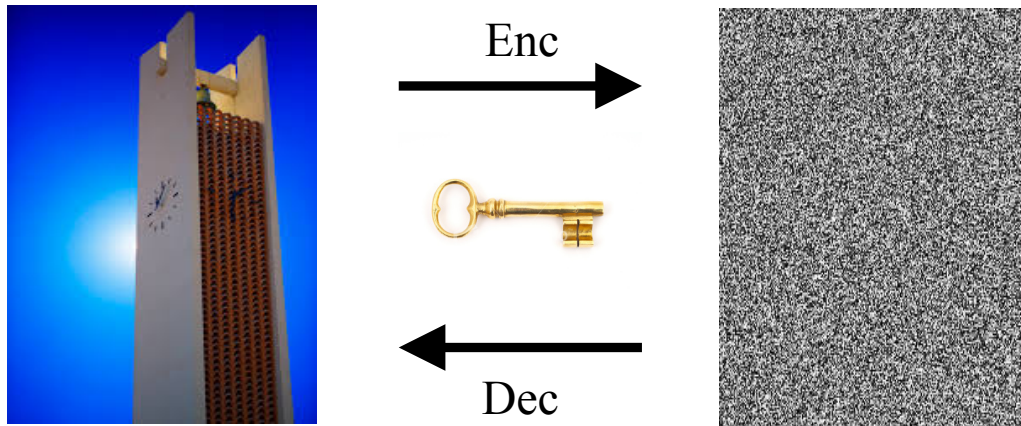
When inventing a cryptographic algorithm/protocol:

- Formulate a precise **definition of security**
- Provide a rigorous **mathematical proof** that the cryptographic algorithm/protocol satisfies the definition of security
- State any **required assumptions** in the proof, keeping them as minimal as possible



# (Symmetric) Encryption algorithms

- $\text{Gen}(1^n)$ : generate a **key** of length  $n$
- $\text{Enc}(m; sk)$ : encrypt **message** (aka **plaintext** or **cleartext**)  $m$  under key  $sk$
- $\text{Dec}(c; sk)$ : decrypt **ciphertext**  $c$  with key  $sk$



$(\text{Gen}, \text{Enc}, \text{Dec})$  is a symmetric-key **encryption scheme** aka **cryptosystem**

# Shared key

- How did Alice and Bob come to share key  $k$ ?
  - maybe they met way in advance
  - maybe a trusted third party distributed the same key to both of them
  - better answers to come...
- But at some point, it was generated and shared

# "Secure" encryption scheme?

Given ciphertext, cannot...

- **Determine key?**
  - Defined as "bits of security"
  - Misses the point: we want to protect message secrecy
- **Determine plaintext?**
  - What if you could get 90% of plaintext?
- **Determine any character of plaintext?**
  - What if you could determine it's greater than 1000?
- **Determine any function of the plaintext!**
  - "Right" definition, but must be formulated carefully, and is stronger than some (many) real-world practical encryption schemes

# Bits of Security

- Assume that attack of concern is determining the key, given many ciphertext/plaintext pairs
- **Brute-force attack**: recover key by trying every possible key
  - e.g., AES-128, try all  $2^{128}$  keys
- **Break** is an attack that recovers key in less work than brute-force
- Suppose best-known attack requires  $2^X$  operations....then  $X$  is the **strength** aka **security level** of the encryption scheme
  - Best case is that strength = key length
  - As attacks are discovered, strength degrades
    - e.g., 3DES-168 has known attack that requires  $2^{112}$  operations, reducing strength from 168 to 112

# Perfect encryption

## One-time pad:

- $\text{Gen}(\text{len})$  = uniformly random sequence of bits of length  $\text{len}$
- $\text{Enc}(m; k) = \text{Dec}(m; k) = m \text{ XOR } k$ 
  - $\text{length}(m) = \text{length}(k)$

## Security:

- Does reveal length of plaintext
- But nothing else!

## Practicality:

- Keys must be long (as long as messages)
- Keys can never be reused, would reveal relationships
  - e.g.,  $(m1 \text{ XOR } k) \text{ XOR } (m2 \text{ XOR } k) = m1 \text{ XOR } m2$
- Distributing one-time use long keys is hard

# Stream Ciphers

# Block Ciphers

- Encryption schemes that operate on fixed-size messages
- The fixed-size is a *block*
- Well-known examples:
  - DES
  - 3DES
  - AES

# DES

- **DES (Data Encryption Standard)**
  - Block size: 64 bits
  - Key size: 56 bits
  - Designed by IBM in 1973-4, tweaked by the NSA, then became the US standard for encryption. International adoption followed.
- **3DES (Triple DES)**
  - Block size: 64 bits
  - Key size: 112 or 168 bits
  - Introduced in 1998, because 56 bit keys had become feasible to brute force.
  - 3DES is simply three DES encryptions with two different keys, for an effective 112 bit key; or with three different keys, for an effective 168 bit key.



# AES

## **AES (Advanced Encryption Standard)**

- Block size: 128 bits
- Key size: 128, 192, or 256 bits
- Public competition held by NIST, ending in 2001
- Now the US standard, approved by the NSA for Top Secret information
- Currently no practical attacks known

# Key lengths

- Various recommendations for strength summarized at <https://www.keylength.com/en/>
- Based on:
  - known attacks
  - hardware capabilities
  - predicted advances
- Why not use highest strength possible? Performance.

# Key lengths

Security	Symmetric	NIST Rec.
$\leq 80$	2TDEA	No
112	3TDEA	until 2030
128	AES-128	Yes
$\geq 256$	AES-256	Yes

# The obvious idea...

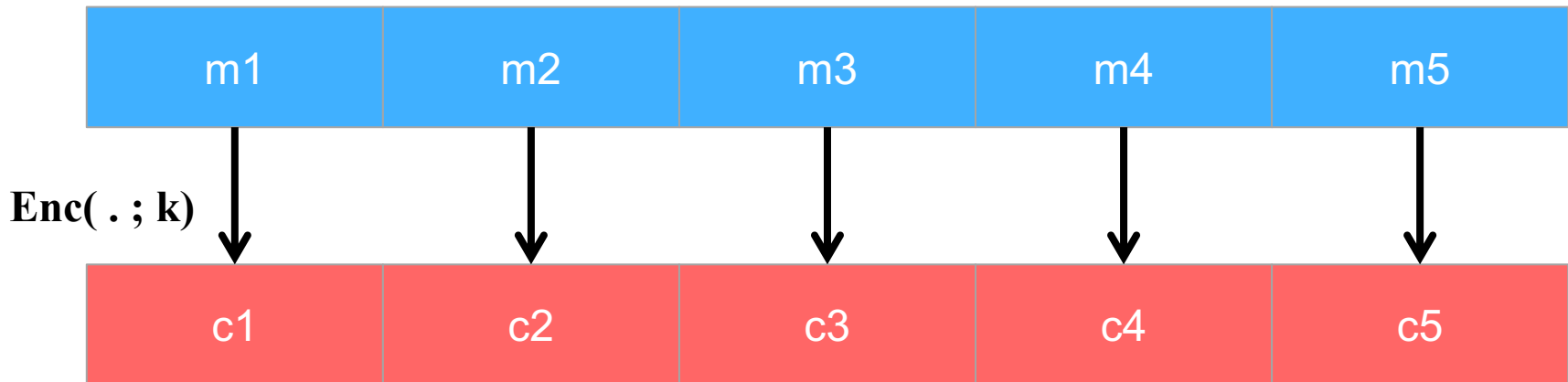
- Divide long message into short chunks, each the size of a block
- Encrypt each block with the block cipher



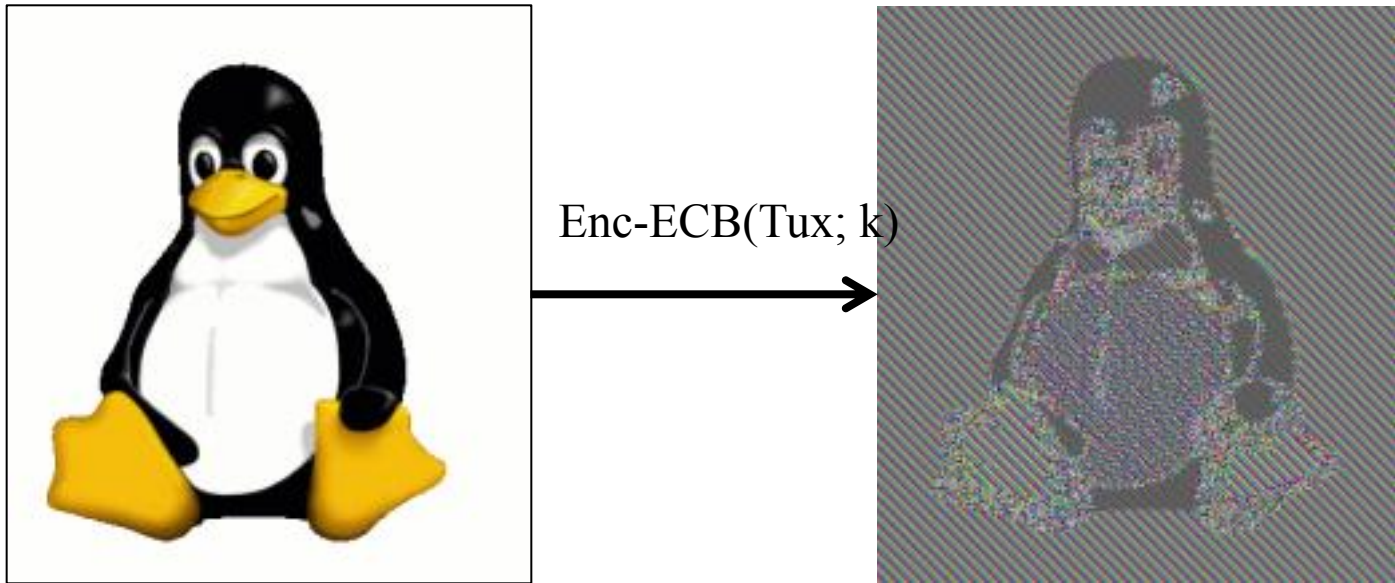
m

# The obvious idea...

- Divide long message into short chunks, each the size of a block
- Encrypt each block with the block cipher



...is a bad idea

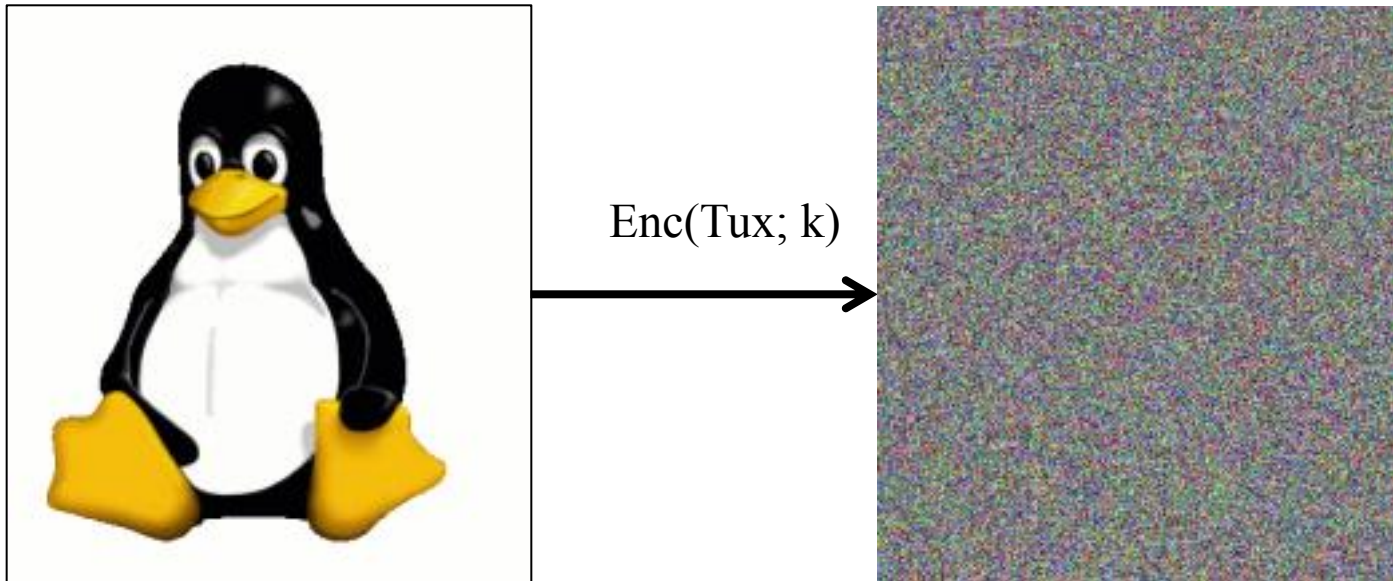


Called *electronic code book*  
(ECB) mode

# Good modes

- Cipher Block Chaining (CBC) mode
  - idea: XOR previous ciphertext block into current plaintext block
- Counter (CTR) mode
  - idea: derive one-time pad from increasing counter
- (and others)
- With both:
  - every ciphertext block depends in some way upon previous plaintext or ciphertext blocks
  - so even if plaintext blocks repeat, ciphertext blocks don't
  - so *intra-message* repetition doesn't disclose information

# Good modes



but what if you encrypt Tux twice under the same key?



# Good modes

- Problem: block ciphers are *deterministic*: inter-message repetition is visible to attacker
- Both CBC and CTR modes require an additional parameter: a *nonce*
  - $\text{Enc}(m; \text{nonce}; k)$
  - $\text{Dec}(c; \text{nonce}; k)$
  - CBC calls the nonce an *initialization vector* (IV)
- Different nonces make each encryption different than others
  - Hence inter-message repetition doesn't disclose information

# Nonces

A nonce is a number used once

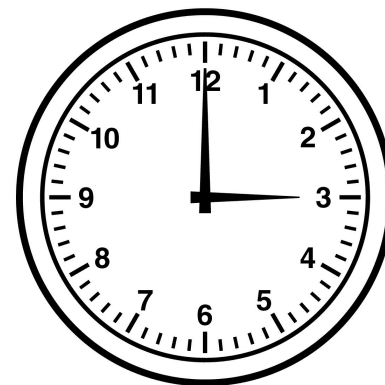
Must be

- **unique:** never used before in lifetime of system and/or (depending on intended usage)
- **unpredictable:** attacker can't guess next nonce given all previous nonces in lifetime of system



# Nonce sources

- **counter**
  - requires state
  - easy to implement
  - can overflow
  - highly predictable
- **clock:** just a counter
- **random number generator**
  - might not be unique, unless drawn from large space
  - might or might not be unpredictable
  - generating randomness:
    - standard library generators often are not cryptographically strong, i.e., unpredictable by attackers
    - cryptographically strong randomness is a black art



# Padding

What if the message length isn't *exactly* a multiple of block length? End up with final block that isn't full:



**Non-solution:** pad out final block with 0's (not reversible)

**Solution:** Let  $B$  be the number of bytes that need to be added to final plaintext block to reach block length. Pad with  $B$  copies of the byte representing  $B$ . Called PKCS #5 or *#7 padding*.

# Protection of integrity

- **Threat:** attacker who controls the network
  - Dolev-Yao model: attacker can read, modify, delete messages
- **Harm:** information contained in messages can be changed by attacker (violating integrity)
- **Vulnerability:** communication channel between sender and receiver can be controlled by other principals
- **Countermeasure:** message authentication codes (MACs)
  - beware: not the same "MAC" as *mandatory access control*

# Encryption and integrity



# Encryption and integrity

## NO!

- Plaintext block might be random number, and recipient has no way to detect change in random number
- Attacker might substitute ciphertext from another execution of same protocol
- In some block modes (e.g., CTR), it's easy to flip individual bits
  - change "admin=0" to "admin=1"
- In some block modes (e.g., CBC), it's easy to truncate blocks from beginning of message
- ...

# MAC algorithms

- $\text{Gen}(\text{len})$ : generate a key of length  $\text{len}$
- $\text{MAC}(m; k)$ : produce a tag for message  $m$  with key  $k$ 
  - message may be arbitrary size
  - tag is typically fixed length
- “Secure MAC”? Must be hard to forge tag for a message without knowledge of key



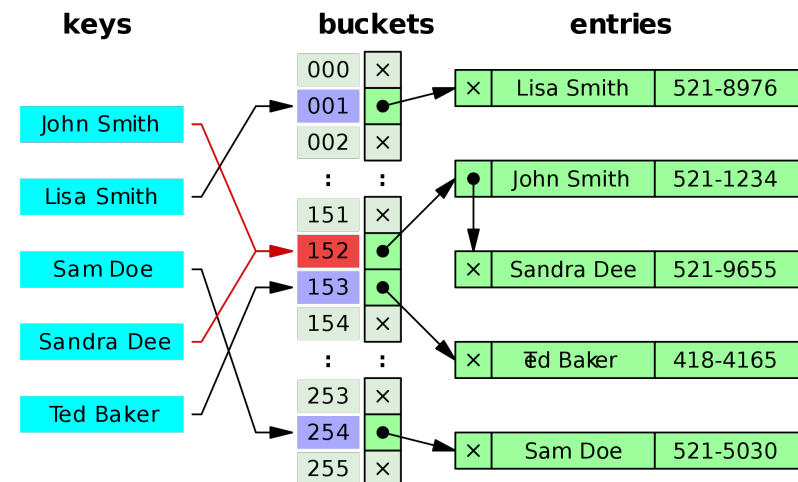


# Real-world MACs

- CBC-MAC
  - Parameterized on a block cipher
  - Core idea: encrypt message with block cipher in CBC mode, use very last ciphertext block as the tag
- HMAC
  - Parameterized on a [hash function](#)
  - Core idea: hash message together with key
  - Your everyday hash function isn't good enough...

# Hash functions

- Input: arbitrary size bit string
- Output: fixed size bit string
  - **compression**: many inputs map to same output, hence creating **collision**
  - for use with hash tables, **diffusion**: minimize collisions (and **clustering**)



# Cryptographic hash functions

- Aka message digest
- Stronger requirements than (plain old) hash functions
- **Goal:** hash is compact representation of original like a
  - Hard to find 2 people with same fingerprint
  - Whether you get to pick pairs of people, or whether you start with one person and find another
    - ...collision-resistant
  - Given person easy to get fingerprint
  - Given fingerprint hard to find person
    - ...one-way



# Real-world hash functions

- **MD5:** Ron Rivest (1991)
  - 128 bit output
  - Collision resistance broken 2004-8
  - Can now find collisions in seconds
  - Don't use it
- **SHA-1:** NSA (1995)
  - 160 bit output
  - Theoretical attacks that reduce strength to less than 80 bits
  - As of 2017, “practical attack” on PDFs: <https://shattered.io/>
  - Industry has been deprecating SHA-1 over the couple years

# Real world hash functions

- **SHA-2:** NSA (2001)
  - Family of algorithms with output sizes {224, 256, 384, 512}
  - In principle, could one day be vulnerable to similar attacks as SHA-1
- **SHA-3:** public competition (won in 2012, standardized by NIST in 2015)
  - Same output sizes as SHA-2
  - Plus a variable-length output called SHAKE

# Encrypt and MAC

0.  $k_E = \text{Gen}_E(\text{len})$   
 $k_M = \text{Gen}_M(\text{len})$
1. A:  $c = \text{Enc}(m; k_E)$   
 $t = \text{MAC}(m; k_M)$
2. A  $\rightarrow$  B:  $c, t$
3. B:  $m' = \text{Dec}(c; k_E)$   
 $t' = \text{MAC}(m'; k_M)$   
if  $t = t'$   
then output  $m'$   
else abort

m



c



# Encrypt and MAC

- **Pro:** can compute Enc and MAC in parallel
- **Con:** MAC must protect confidentiality
  
- Example: `ssh` (Secure Shell) protocol
  - recommends AES-128-CBC for encryption
  - recommends HMAC with SHA-2 for MAC

# Aside: Key reuse

- Never use same key for both encryption and MAC schemes
- **Principle:** every key in system should have unique purpose



# Encrypt then MAC

1. A:  $c = \text{Enc}(m; k_E)$   
 $t = \text{MAC}(c; k_M)$

2. A  $\rightarrow$  B:  $c, t$

3. B:  $t' = \text{MAC}(c; k_M)$   
if  $t = t'$

then output  $\text{Dec}(c; k_E)$   
else abort

m



c



# Encrypt then MAC

- **Pro:** provably most secure of three options [Bellare & Namprepre 2001]
- **Pro:** don't have to decrypt if MAC fails
  - resist DoS
- **Example:** IPsec (Internet Protocol Security)
  - recommends AES-CBC for encryption and HMAC-SHA1 for MAC, among others
  - or AES-GCM

# MAC then encrypt

1. A:  $t = \text{MAC}(m; k_M)$   
 $c = \text{Enc}(m, t; k_E)$
2. A  $\rightarrow$  B:  $c$
3. B:  $m', t' = \text{Dec}(c; k_E)$   
if  $t' = \text{MAC}(m'; k_M)$   
then output  $m'$   
else abort

m



c



# MAC then encrypt

- **Pro:** provably next most secure
  - and just as secure as Encrypt-then-MAC for strong enough MAC schemes
  - HMAC and CBC-MAC are strong enough
- **Example:** SSL (Secure Sockets Layer)
  - Many options for encryption, e.g. AES-128-CBC
  - For MAC, standard is HMAC with many options for hash, e.g. SHA-256

# Authenticated encryption

- Three combinations:
  - Enc and MAC
  - Enc then MAC
  - MAC then Enc
- Let's unify all with a pair of algorithms:
  - $\text{AuthEnc}(m; ke; km)$ : produce an authenticated ciphertext  $x$  of message  $m$  under encryption key  $ke$  and MAC key  $km$
  - $\text{AuthDec}(x; ke; km)$ : recover the plaintext message  $m$  from authenticated ciphertext  $x$ , and verify that the MAC is valid, using  $ke$  and  $km$ 
    - Abort if MAC is invalid

# Authenticated encryption

- Newer block cipher modes designed to provide confidentiality and integrity
  - **OCB**: Offset Codebook Mode
  - **CCM**: Counter with CBC-MAC Mode
  - **GCM**: Galois Counter Mode

