

Crypto for Integrity

CS 138

Spring 2026



Protection of integrity

- **Threat:** attacker who controls the network
 - Dolev-Yao model: attacker can read, modify, delete messages
- **Vulnerability:** communication channel between sender and receiver can be controlled by other principals
- **Harm:** information contained in messages can be changed by attacker (violating integrity)
- **Countermeasure:** [more crypto](#)

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 (replay)
- Adversary can modify encrypted plaintext in predictable ways (malleability)

Malleable Ciphertexts

- AES-CBC
 - Adversary can truncate blocks from end of message
- AES-CTR
 - Flipping bits of plaintext flips bits of ciphertext
- RSA
 - Adversary can multiply message

MAC algorithms

- $\text{Gen}(1^n)$: generate a **key** k of length n
- $\text{MAC}(m; k)$: produce a **tag** t for message m
- $\text{Verify}(m, t; k)$: returns 1 if m was the message used to generate t and 0 otherwise



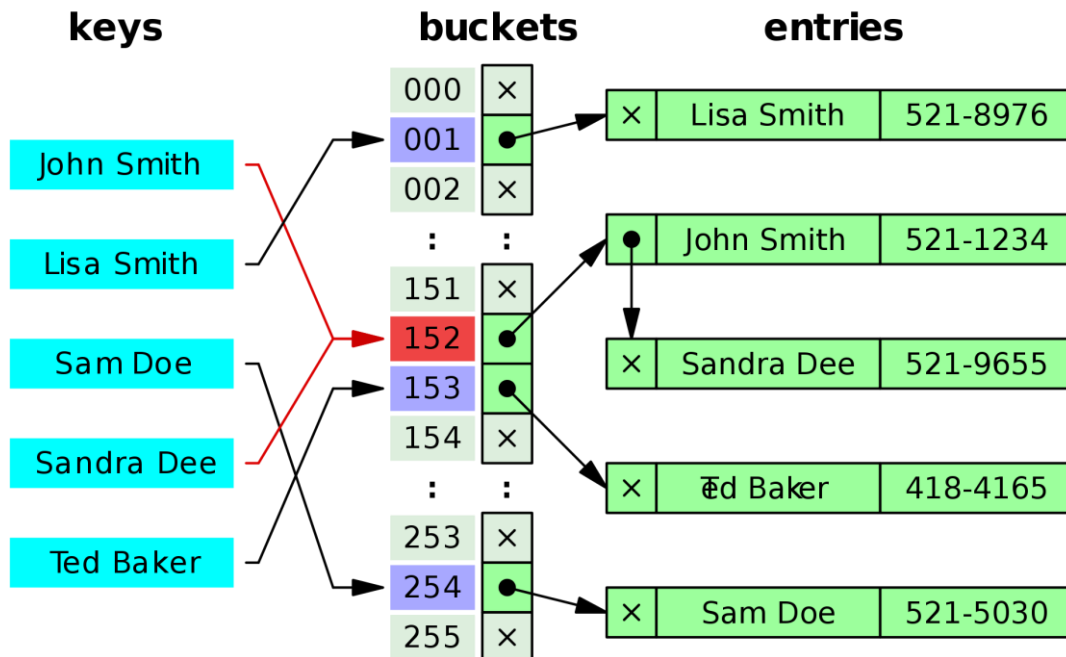
- A MAC is **correct** if the tags produced by MAC are valid, ie, $\text{Verify}(m, \text{MAC}(m, t; k))$ evaluates to 1
- A MAC is **secure** if it is hard for a PPT algorithm to forge a valid tag without the key

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:** size of the output is smaller than the input
 - **diffusion:** minimize collisions (and clustering)



Cryptographic hash functions

- 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**



Exercise: MACs

- Consider a hash function f that breaks a value into 4-byte blocks and returns the xor of these blocks. Would this function make a good HMAC? Why or why not?
1. **compression**
 2. **diffusion**
 3. **collision-resistant**
 4. **one-way**

Historical 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/>
 - Don't use it

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 option 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

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

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

m



c



t



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, `ssh` (Secure Shell) protocol used this
 - default encryption is chacha20
 - default MAC is umac, recommends HMAC-SHA2-512 or 256
- Example: IPsec (Internet Protocol Security)
 - recommends AES-CBC for encryption and HMAC-SHA2-384 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. CHACHA20, AES-256
 - For MAC, standard is HMAC with many options for hash, e.g. SHA-256, SHA-384

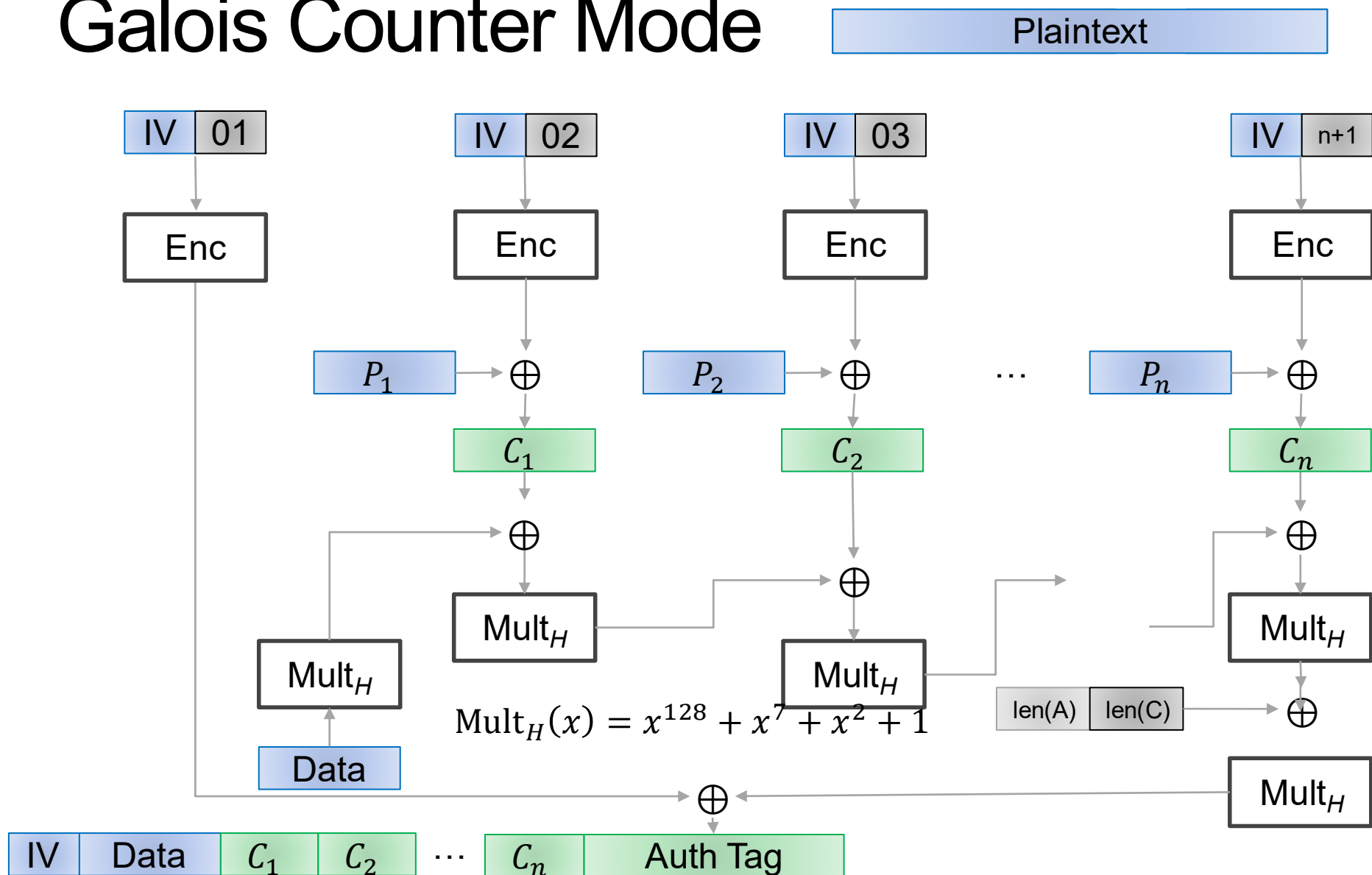
Aside: Key reuse

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

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

Galois Counter Mode



DIGITAL SIGNATURES

Recall: Key pairs

- Instead of sharing a key between pairs of principals...
- ...every principal has a pair of keys
 - **public key:** published for the world to see
 - **private key:** kept secret and never shared



Key pair terminology

	Encryption	Digital Signatures
Public key	Encryption key	Verification key
Private key	Decryption key	Signing key

Digital Signatures

- $\text{Gen}(1^n)$: generate a **keypair** (pk, sk) of length n
- $\text{Sign}(m; sk)$: produce a **signature** σ for message m
- $\text{Verify}(m, \sigma; pk)$: returns 1 if m was the message used to generate σ and 0 otherwise



- A digital signature scheme is **correct** if $\text{Verify}(m, \text{Sign}(m, t; sk); pk)$ evaluates to 1
- A digital signature is **secure** if it is hard for a PPT algorithm to forge a valid signature without sk

RSA

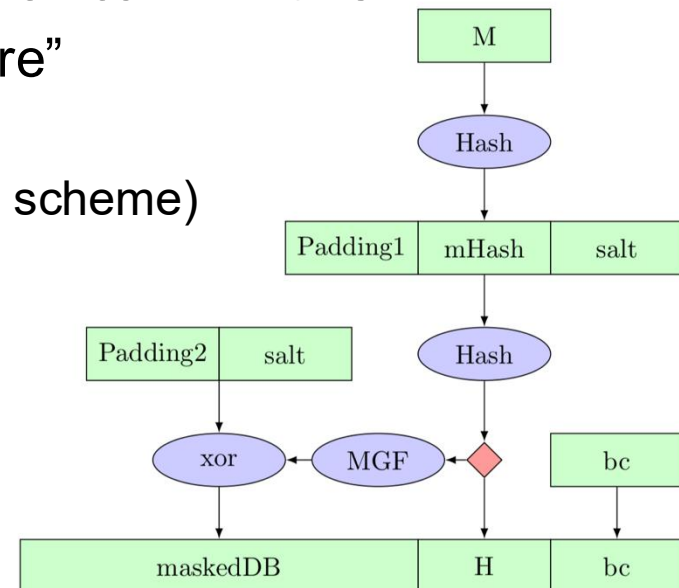
- Core ideas are the same as RSA encryption, but backward
- Intuition: “RSA sign = encrypt with private key”
- Gen(len):
 - Pick primes p, q , define $n = p \cdot q$
 - Choose e, d such that $ed = 1 \pmod{(p-1)(q-1)}$
 - $pk = (n, e)$, $sk = (p, q, d)$
- Sign(m ; sk)
$$\sigma = m^d \pmod n$$
- Verify(m, σ ; pk):
$$m == \sigma^e \pmod n$$

Exercise: Forging Signatures

- Assume that an adversary convinces Alice to sign two messages m_1 and m_2 with the same key, producing signatures σ_1 and σ_2 . How could this adversary forge a signed message with the value m_1m_2 ?

RSA

- Core ideas are the same as RSA encryption
- Intuition: “RSA sign = encrypt with private key”
- Truth (in real world, outside of textbooks):
 - there's a core RSA function R that works with either pk or sk
 - RSA encrypt = do some prep work on m then call R with pk
 - RSA sign = do **different** prep work on m then call R with sk
 - Prep work: recall “textbook RSA is insecure”
 - (For encryption: OAEP)
 - For signatures: PSS (probabilistic signature scheme)
 - Also need to handle long messages...



Signatures with hashing

1. A: $s = \text{Sign}(H(m); k_A)$

2. A \rightarrow B: m, s

3. B: accept if $\text{Ver}(H(m); s; K_A)$

DSA

DSA: Digital Signature Algorithm [Kravitz 1991]

- Standardized by NIST and made available royalty-free in 1991/1993
- Used for decades without any serious attacks
- Closely related to Elgamal encryption
- Usual implemented with elliptic curve (ECDSA, Ed25519)

Blind signatures

[Chaum 1983]

- Purpose: signer doesn't know what they are signing
- Two additional algorithms: Blind and Unblind
- $\text{Unblind}(\text{Sign}(\text{Blind}(m); k)) = \text{Sign}(m; k)$
- Uses: e-cash, e-voting

Group signatures

[Chaum and van Heyst 1991]

- Purpose: one member of group signs anonymously on behalf of group
- Introduces a *group manager* who controls membership
- Two new protocols: Join and Revoke, to manage membership
- One new algorithm: Open, which manager can run to reveal who signed a message

Crypto for Integrity

