Symmetric and Asymmetric Cryptography

CMPSC 403 Fall 2021

October 14, 2021

Outline

Symmetric key cryptography

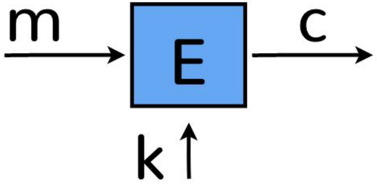
- Encryption
- Hash functions
- Message authentication codes

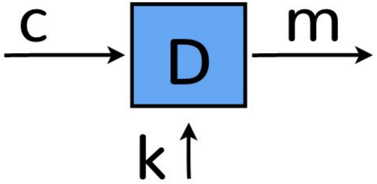
Asymmetric key cryptography

- Key exchange
- Digital signature

Symmetric Key Encryption

- Encryption: (key, plaintext) → ciphertext
  - $E_k(m) = c$
- Decryption: (key, ciphertext) → plaintext
  - $D_k(c) = m$
- Functional property: Where $D_k(E_k(m)) = m$





Symmetric Key Encryption

- One-time key: used to encrypt one message
  - Encrypted email - new key generate per email
- Multi-use key: used to encrypt multiple messages
  - SSL (Secure Sockets Layer)- same key used to encrypt many packets

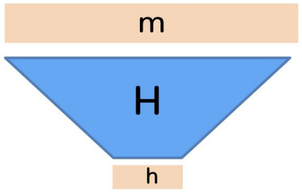
Symmetric Encryption Algorithms

Algorithm	Type	Key Size	Features
DES	Block Cipher	56 bits	Most Common, Not strong enough
TripleDES	Block Cipher	168 bits (112 effective)	Modification of DES, Adequate Security
Blowfish	Block Cipher	Variable (Up to 448 bits)	Excellent Security
AES	Block Cipher	Variable (128, 192, or 256 bits)	Replacement for DES, Excellent Security
RC4	Stream Cipher	Variable (40 or 128 bits)	Fast Stream Cipher, Used in most SSL implementations

Hash Function

- A (cryptographic) hash function maps arbitrary length input into a fixed-size string
  - |m| is arbitrarily large
  - |h| is fixed, usually 128-512 bits

$h = H(m)$



Popular broken hash functions

- MD5: Message Digest
  - Designed by Ron Rivest
  - Output: 128 bits
- SHA-1: Secure Hash Algorithm 1
  - Designed by NSA
  - Output: 160 bits

Hash functions

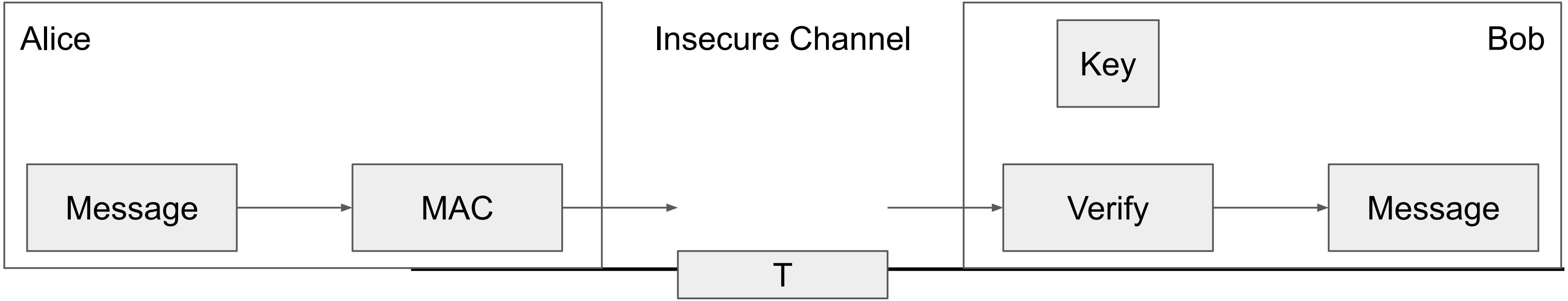
- SHA-2: Secure Hash Algorithm 2
  - Designed by NSA
  - Output: 224, 256, 384, or 512 bits
- SHA-3: Secure Hash Algorithm 3
  - Result of NIST SHA-3 contest
  - Output: arbitrary size
  - Replacement once SHA-2 broken

Do Hash functions provide integrity?

- It depends on your threat model
- If the attacker can modify the hash, hashes don't provide integrity
- **Main issue**: Hashes are *unkeyed* functions
  - There is no secret key being used as input, so any attacker can compute a hash on any value
- **Next**: Use hashes to design schemes that provide integrity

MACs

- Validate message integrity based on shared secret
- **MAC**: Message Authentication Code
  - Keyed function using shared secret
  - Hard to compute function without knowing key

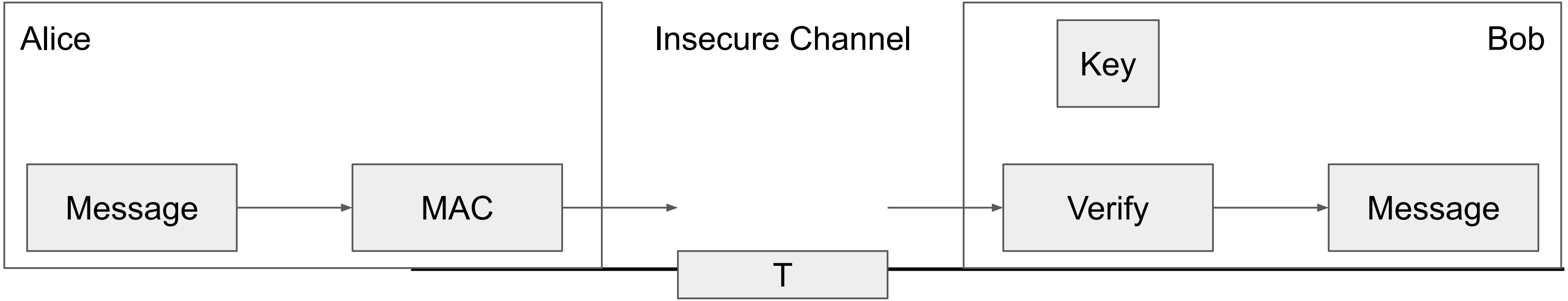


MACs

Two parts:

1. KeyGen() $\rightarrow$ *K*: Generate a key *K*
2. MAC(*K*, *M*) $\rightarrow$ *T*: Generate a tag *T* for the message *M* using key *K*

- Inputs: A secret key and an arbitrary-length message
- Output: A fixed-length **tag** on the message



Think Along!

In 2009, Flickr required API calls to use authentication token that looked like:

MD5(secret || arg1=val1 & arg2=val2 & ...)

- **Is MAC(k, m) = H(k || m) a secure MAC? Why/why not?**

|| - concatenation

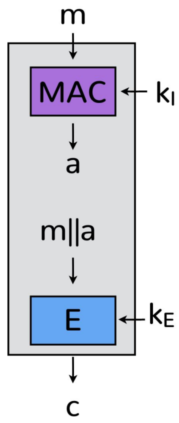
Ref: https://vnhacker.blogspot.com/2009/09/flickrs-api-signature-forgery.html

Example: HMAC

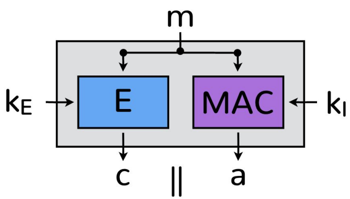
- HMAC(K, M):
  - Output $H((K \oplus opad) \mid\mid H((K \oplus ipad) \mid\mid M))$
- Use K to derive two different keys
  - <u>*opad*</u> (outer pad) is the hard-coded byte **`0x5c`** repeated until it's the same length as K
  - <u>*ipad*</u> (inner pad) is the hard-coded byte **`0x36`** repeated until it's the same length as K
  - As long as *opad* and *ipad* are different, you'll get two different keys
  - For paranoia, the designers chose two very different bit patterns, even though they theoretically need only differ in one bit

Combining MAC with encryption

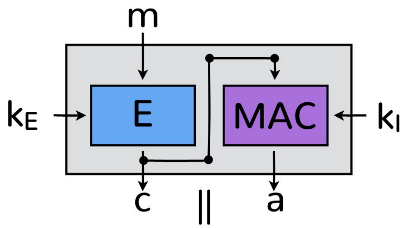
- MAC then Encrypt



- Encrypt and MAC



- Encrypt then MAC



Asymmetric cryptography/public-key cryptography

- **Main insight**: Separate keys for different operations.
- Keys come in pairs, and are related to each other by the specific algorithm:
  - **Public key**: known to everyone, used to encrypt or verify signatures
  - **Private key**: used to decrypt and sign

Public Key Encryption

- <u>Encryption</u>: (public key, plaintext) → ciphertext
  - $E_{pk}(m) = c$
- <u>Decryption</u>: (secret key, ciphertext) → plaintext
  - $D_{sk}(c) = m$
- Encryption and decryption are inverse operations
  - $D_{sk}(E_{pk}(m)) = m$

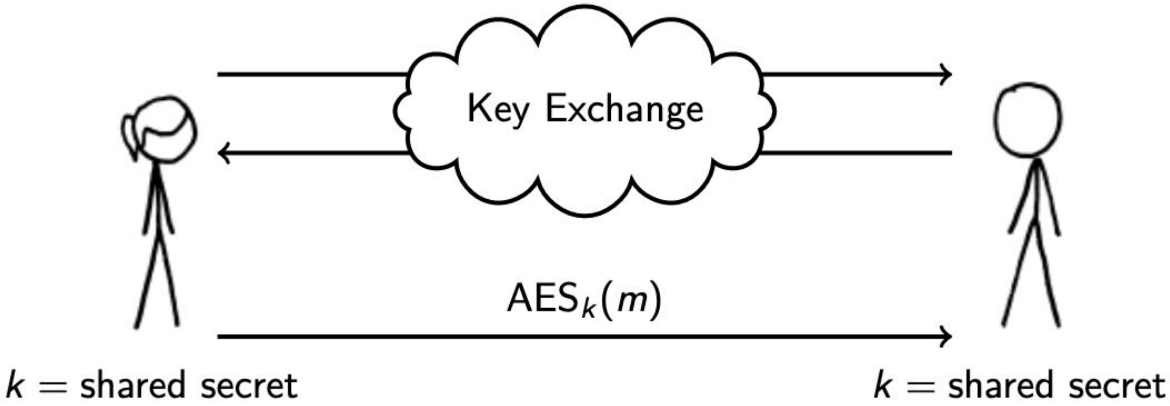
Think Along!

Modular Arithmetic: a mod b

- Add/subtract/multiply
- Division/inverse
- Exponentiation

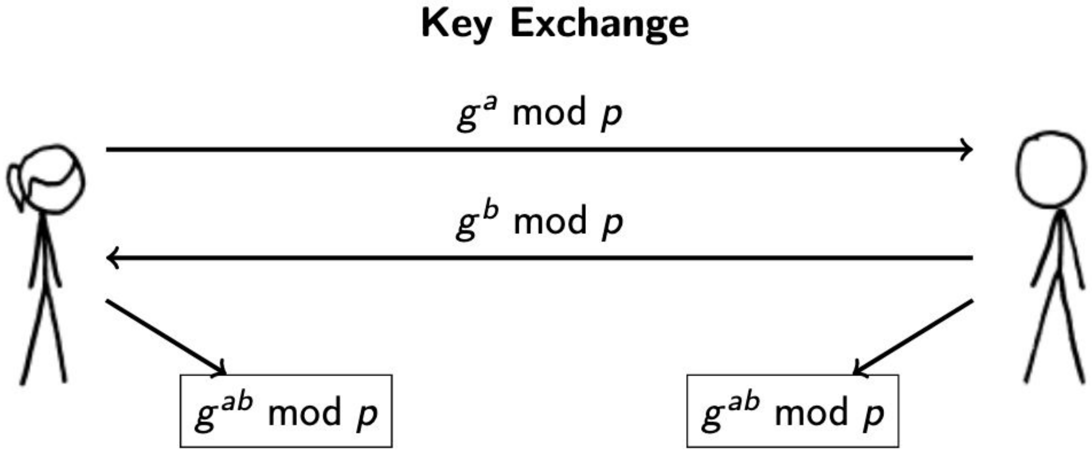
Public Key Crypto Option #1: Key Exchange

- Solving key distribution without trusted third parties



Diffie-Hellman Key Exchange

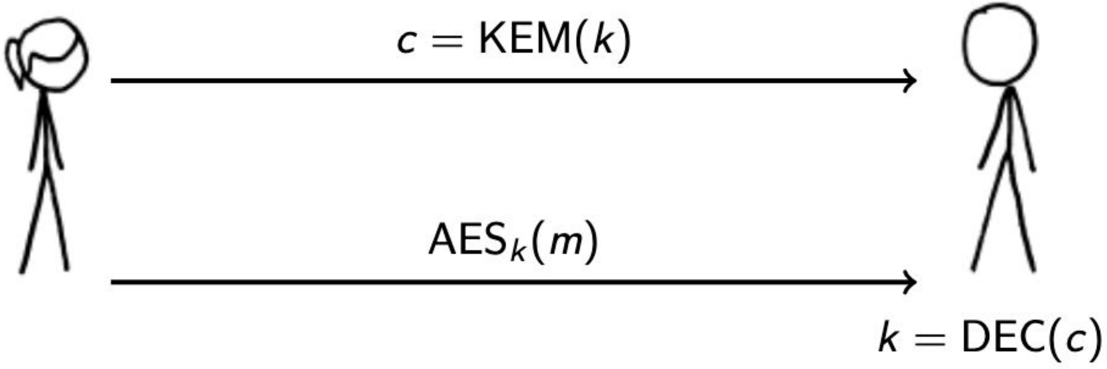
- Parameters:
  - **p**: a prime
  - **g**: an integer mod p



Note: $(g^a)^b \bmod p = g^{ab} \bmod p = g^{ba} \bmod p (g^b)^a \bmod p$.

Public Key Crypto Option #2: Key Encapsulation

- Solving key distribution without trusted third parties



Public-Key Encryption

- **Three parts:**
  - KeyGen() → *PK*, *SK*: Generate a public/private keypair, where *PK* is the public key, and *SK* is the private (secret) key
  - Enc(*PK*, *M*) → *C*: Encrypt a plaintext *M* using public key *PK* to produce ciphertext *C*
  - Dec(*SK*, *C*) → *M*: Decrypt a ciphertext *C* using secret key *SK*

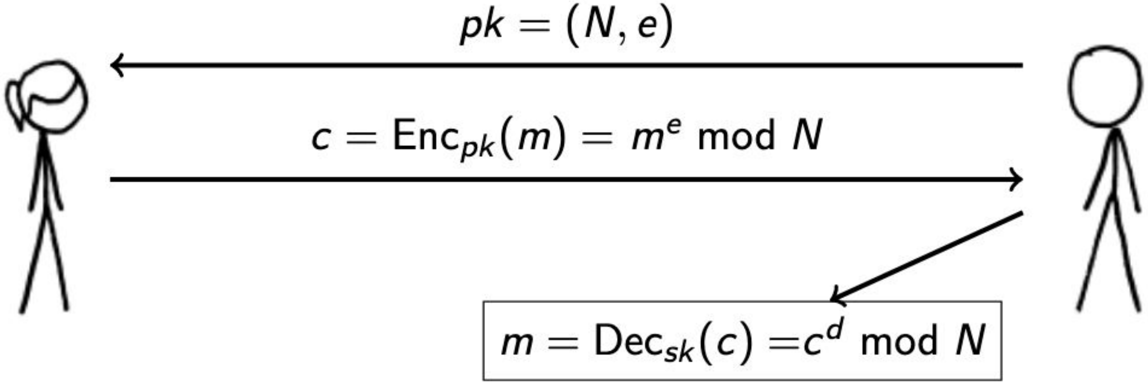
RSA

Public key **pk**

- **N** = pq modulus
- **e** - encryption exponent

Secret key **sk**

- **p, q** - primes
- **d** - decryption exponent

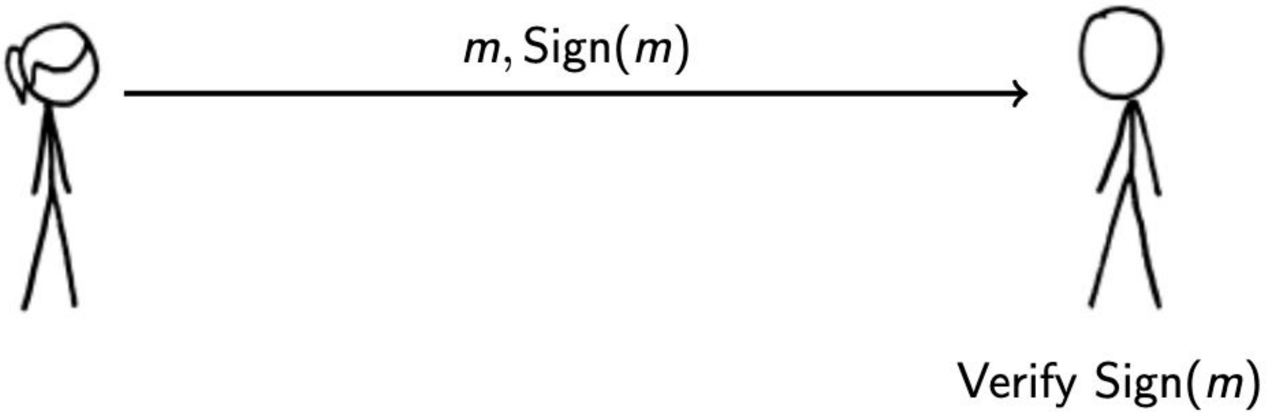


$\mathrm{Dec}(\mathrm{Enc}(m)) = m^{ed} \bmod N \equiv m^{1+k\phi(N)} \equiv m \bmod N$ by Euler's theorem.

Details at https://muirlandoracle.co.uk/2020/01/29/rsa-encryption/

Public Key Idea #3: Digital Signatures

- Bob wants to verify Alice's signature using only a public key.
  - Signature verifies that Alice was the only one who could have sent this message.
  - Signature also verifies that the message hasn't been modified in transit.

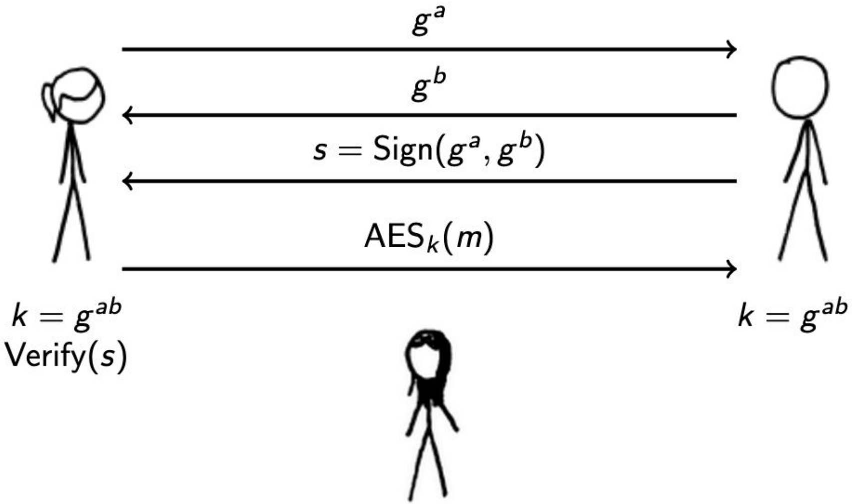


Digital Signatures: Definition

- **Three parts:**
  - KeyGen() → *PK*, *SK*: Generate a public/private keypair, where *PK* is the verify (public) key, and *SK* is the signing (secret) key
  - Sign(*SK*, *M*) → *sig*: Sign the message *M* using the signing key *SK* to produce the signature *sig*
  - Verify(*PK*, *M*, *sig*) → {0, 1}: Verify the signature *sig* on message *M* using the verify key *PK* and output 1 if valid and 0 if invalid

Putting it all together

- Diffie-Hellman used to negotiate shared session key.
- Alice verifies Bob's signature to ensure that key exchange was not man-in-the-middle.
- Shared secret used to symmetrically encrypt data.



Public key cryptography and quantum computers

- All current “hard” problems involved in current public key cryptography can be solved by a general purpose quantum computer.
- Efforts to develop replacements:
  - Lattice-based cryptography
  - Multivariate cryptography
  - Hash-based signatures
  - Supersingular isogeny Diffie-Hellman

For more: book "Cryptography Apocalypse" By Roger A. Grimes