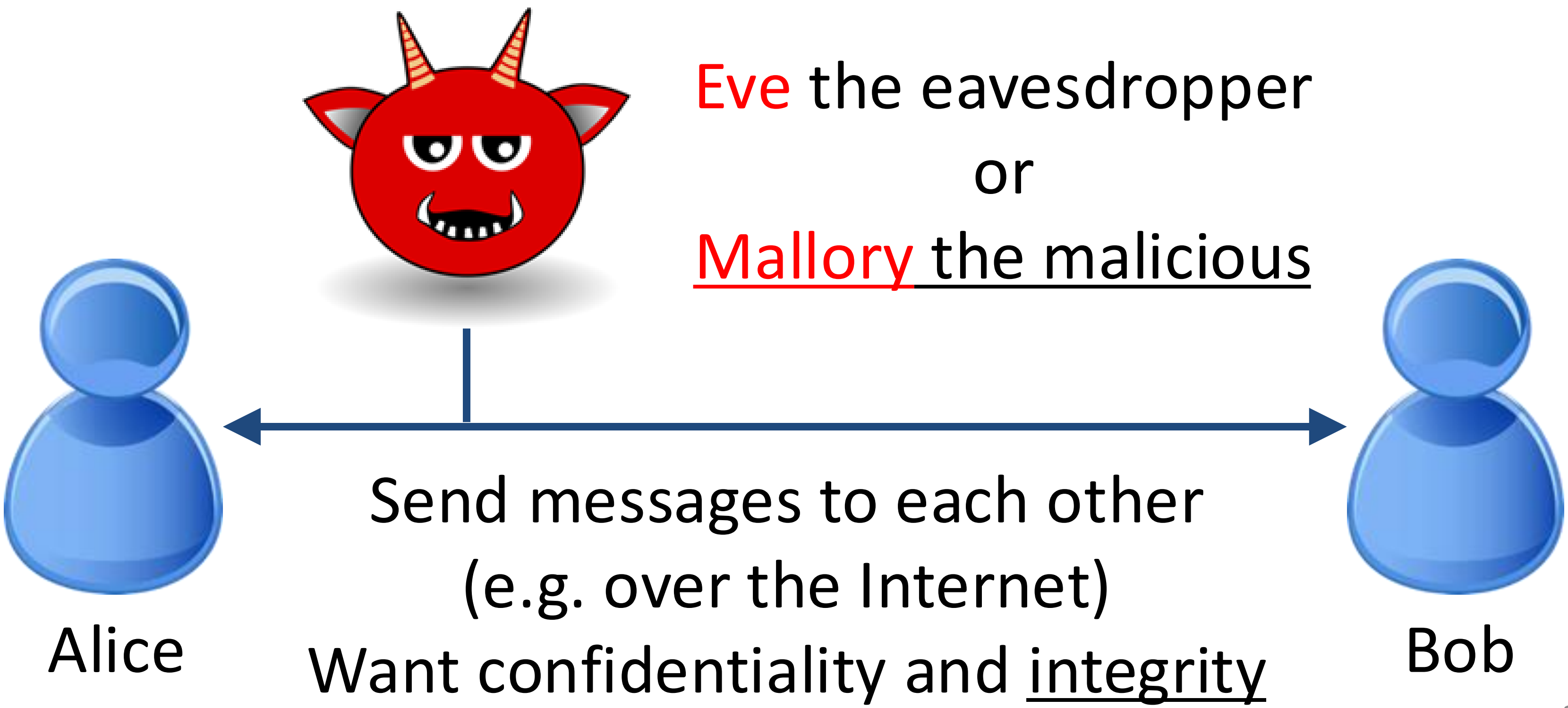
# Lecture 18 – Message Integrity

University of Illinois
ECE 422/CS 461

# Cryptography (or Cryptology)

- Studies techniques for secure communication in the presence an adversary who has control over the communication channel



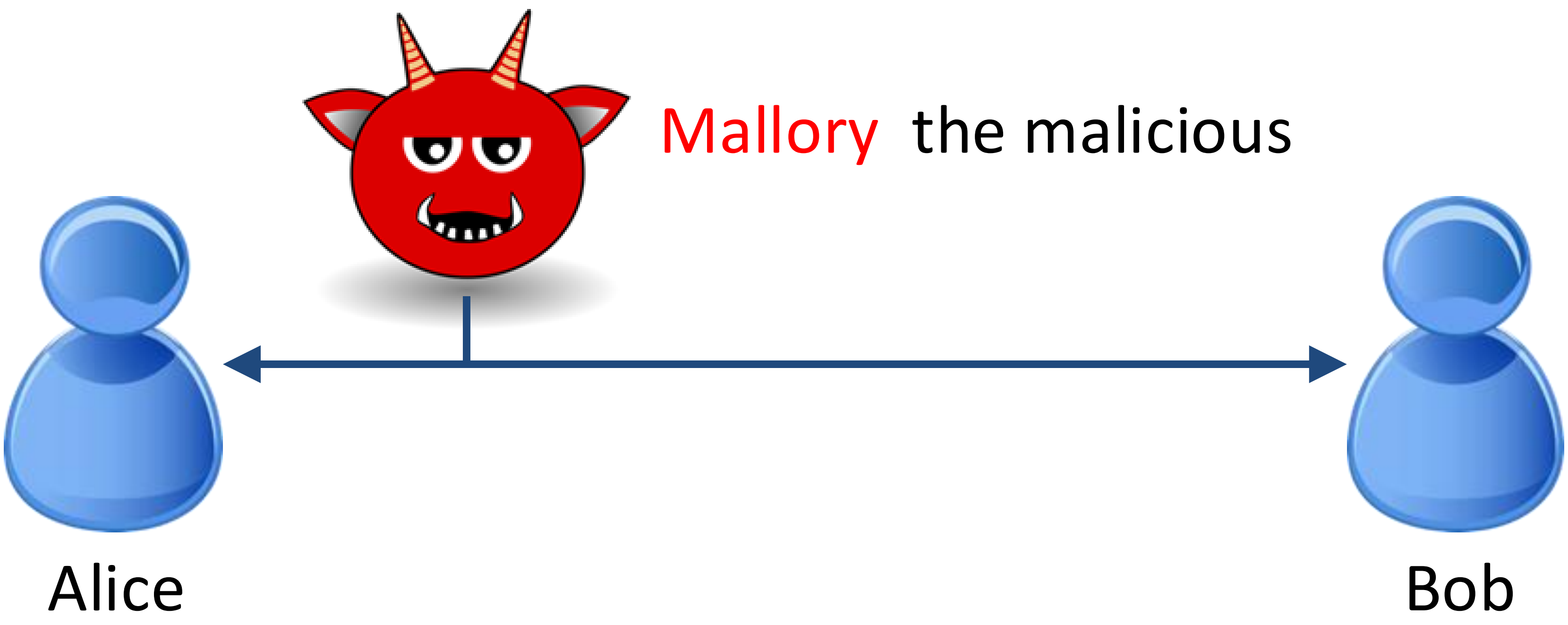
# Goals of this Lecture

- By the end of this lecture you should know the following about MAC and digital signatures:
  - Interface
  - Security definition
  - Common/recommended constructions
  - Applications

  - Relation to hashing and encryption
  - How to achieve confidentiality + integrity

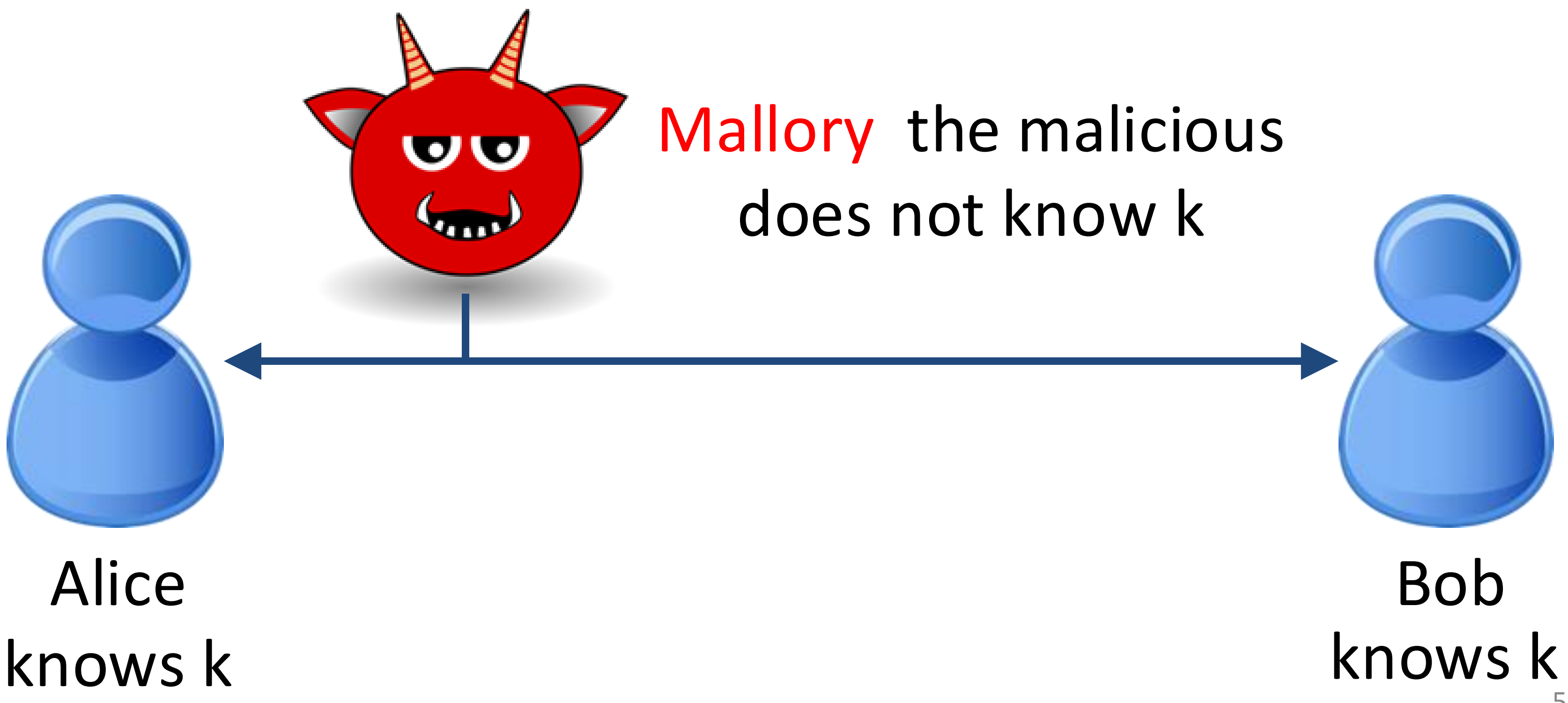
# Message Integrity

- By integrity, we meant *tamper-evident*
- Symmetric vs. asymmetric



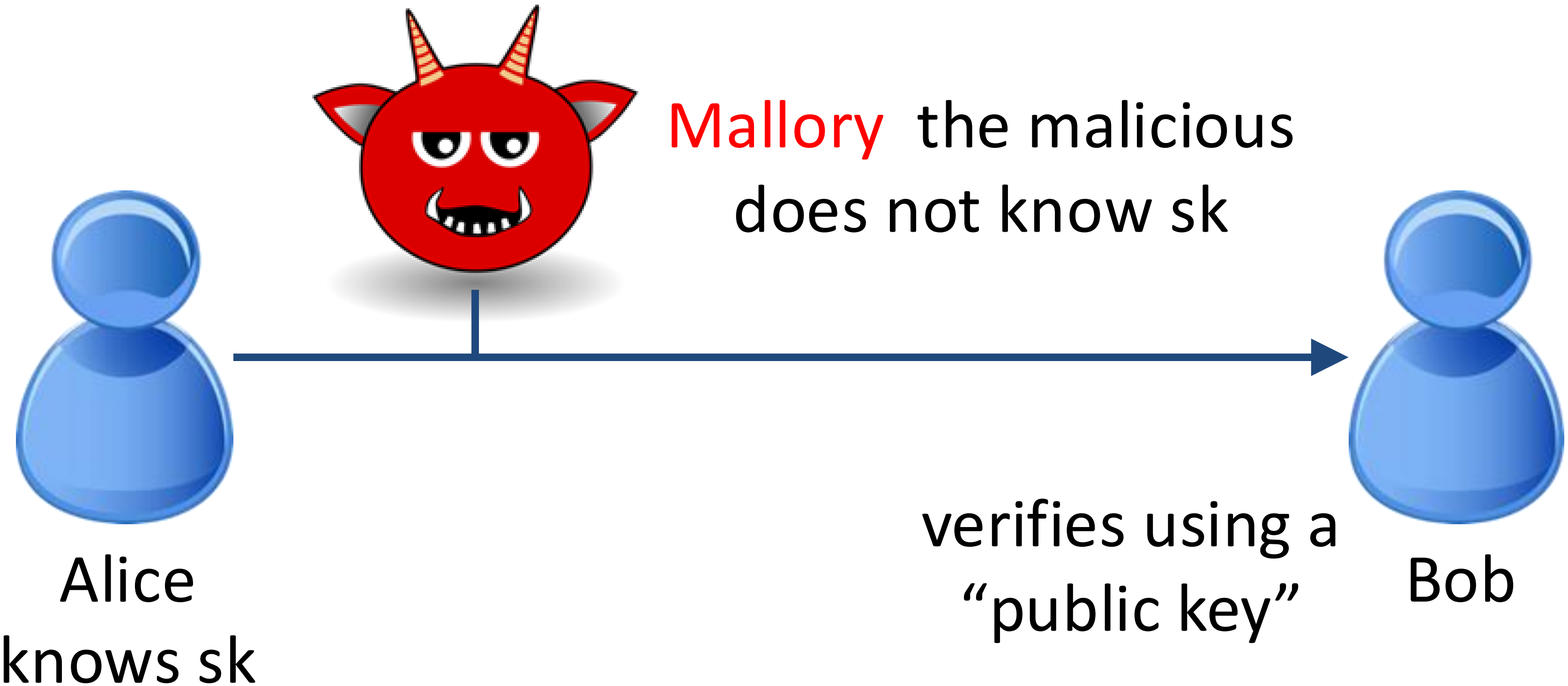
# Message Authentication Code (MAC)

- Shared secret k → MAC (symmetric)



# Digital Signature

- Shared secret k → MAC (symmetric)
- Only sender has secret sk → digital signature (asymmetric)



# Interface

- Message Authentication Code (MAC)
  - MAC(k, m) → t (called a tag)
  - Sender sends tuple (m, t)
  - Receiver checks MAC(k, m) == t?

# Interface

- Message Authentication Code (MAC)
  - MAC(k, m) → t (called a tag)
  - Sender sends tuple (m, t)
  - Receiver checks MAC(k, m) == t?
- Digital Signatures
  - KeyGen() → (vk, sk)
    - A private *signing key* and a public *verification key*
  - Sign(sk, m) → σ (called a signature)
  - Verify(vk, m, σ) → True/False

# Combine with Hash for Long Msg

- Message Authentication Code (MAC)
  - MAC(k, **H(m)**) → t (called a tag)
- Digital Signatures
  - Sign(sk, **H(m)**) → σ (called a signature)
  - Verify(vk, **H(m)**, σ) → True/False
- What property of hash is being used here?

# Message Authentication Code

# Message Authentication Code

- Interface: MAC(k, m) → t (called a tag)
- How do we define security of MAC?
  - We pick a random key k
  - The attacker Mallory wins if she can produce a **forgery,** i.e., (m, t) such that t = MAC(k, m)

# Unforgeability under Chosen Message Attack (UF-CMA)

- Interface: MAC(k, m) → t (called a tag)
- How do we define security of MAC?
  - We pick a random key k
  - Mallory can ask for MACs of any messages
  - The attacker Mallory wins if she can produce a **forgery,** i.e., (m, t) such that t = MAC(k, m)
    - m must be a message Mallory did not ask MAC for
- Compare with IND-CPA?

# UF-CMA vs. IND-CPA

- Similarities
  – Both give adversary access to many pairs of plaintext-ciphertext or message-MAC
  – Adversary can choose which message she wants to distinguish or forge

- Key difference: adversary is not allowed to forge a message-MAC pair she has seen
  – "Replay" attack is possible for MAC/signature

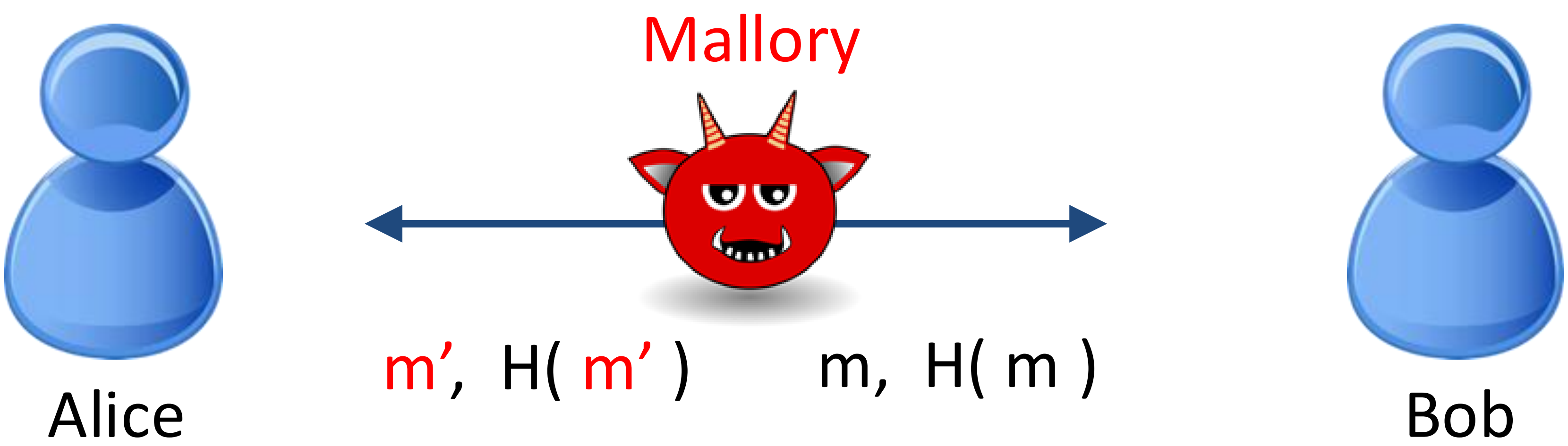
# Message Authentication Code

- Interface: MAC(k, m) → t (called a tag)
- Security definition: UF-CMA

- Common construction: hash-based MAC

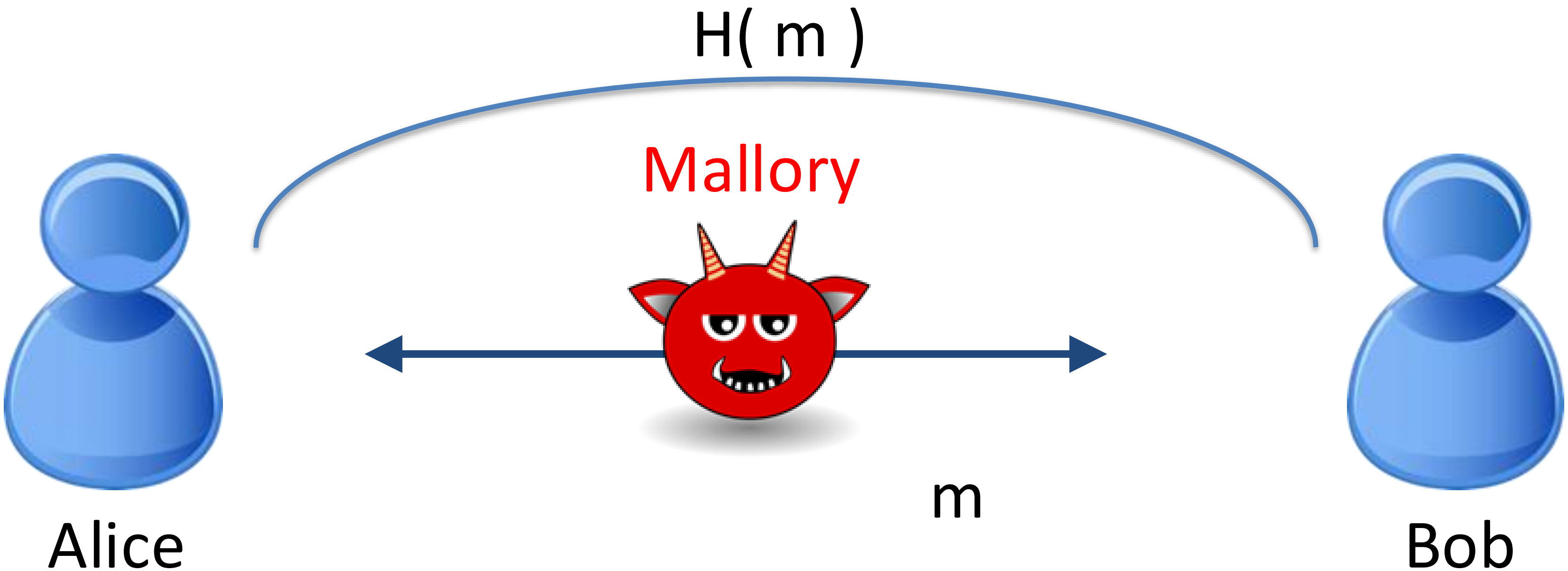
# Hash & Message Authentication

- In its simplest form, a cryptographic hash does NOT work for a message authentication



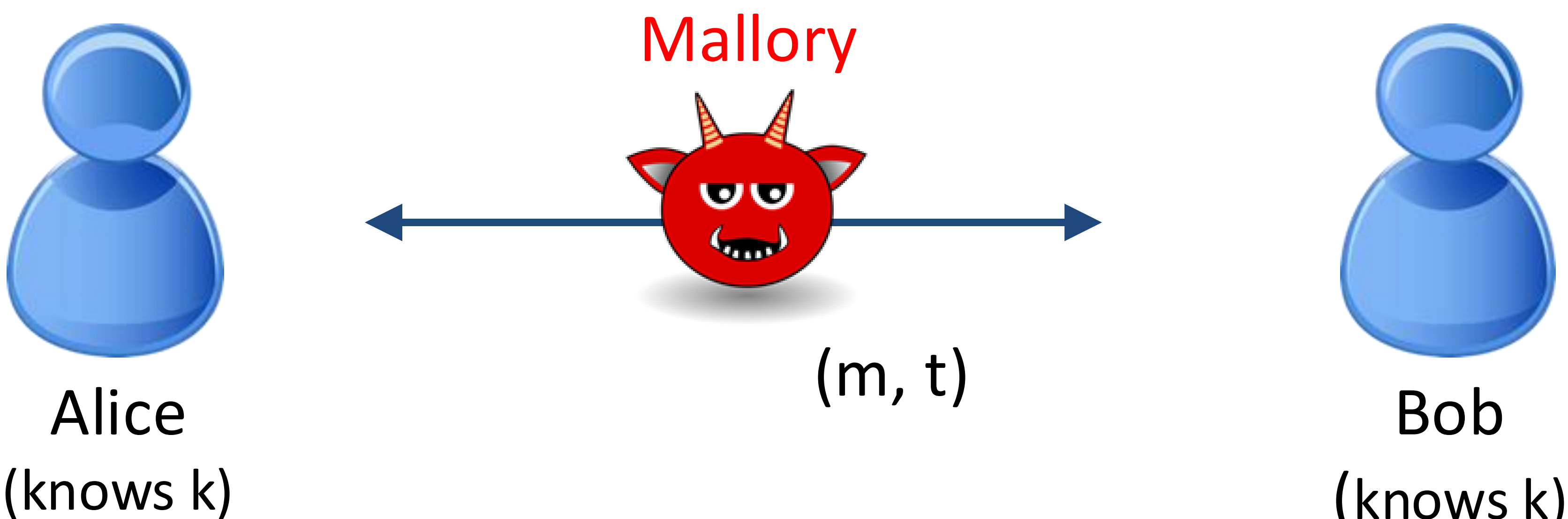
# Hash & Message Authentication

- Two settings it can work:
  - If the hash can be transmitted in another *trusted* but low-bandwidth channel



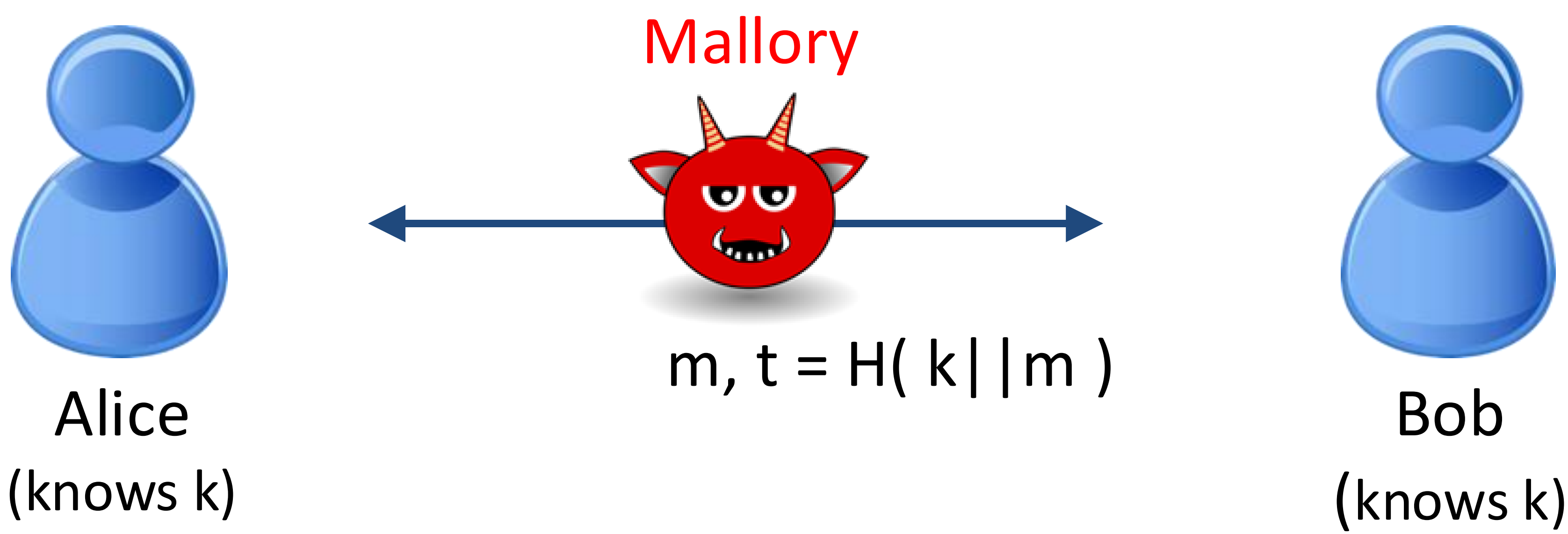
# Hash & Message Authentication

- Two settings it can work:
  - If the hash can be transmitted in another *trusted* but low-bandwidth channel, or
  - If Alice and Bob share a secret key k (can use MAC)



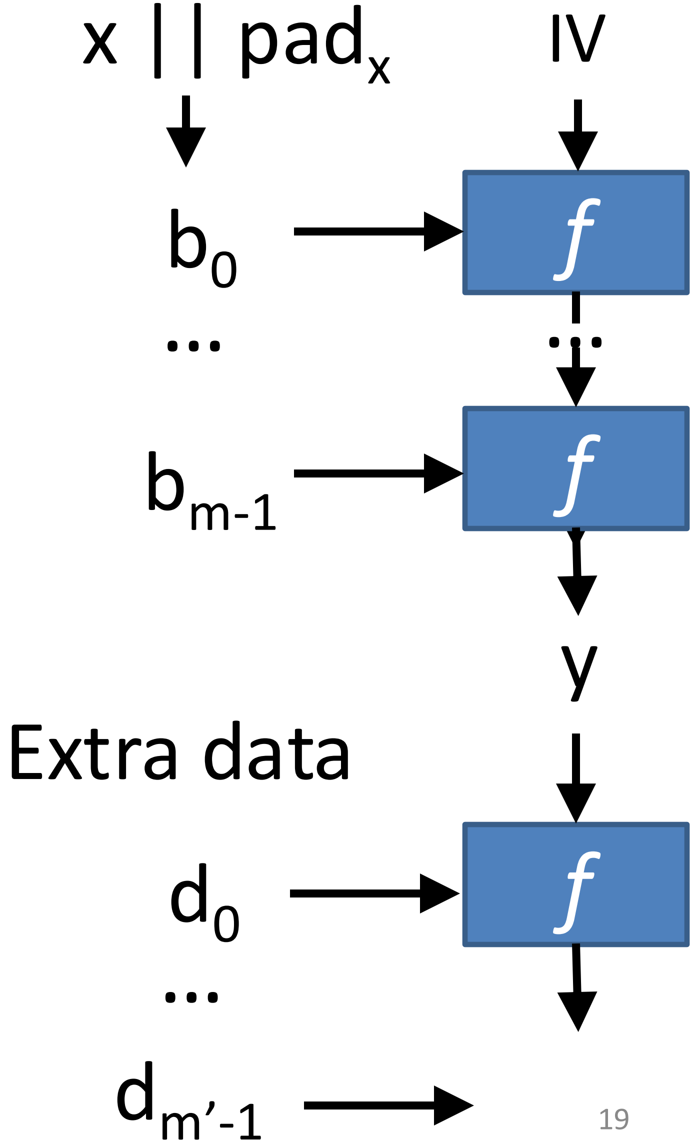
# Hash-Based MAC (HMAC)

- Natural method: t = H( k||m )
- Is this secure?
- Yes, if H is pseudorandom



# Length Extension Attacks

- Given H(x), one can compute H(x||$pad_x$||ExtraData) with more rounds of *f*
  - Apply the padding for "x||$pad_x$||ExtraData"
- A random oracle would not exhibit this behavior
  - Given { $H(x_i)=y_i$ }, for a new x', H(x') would be random

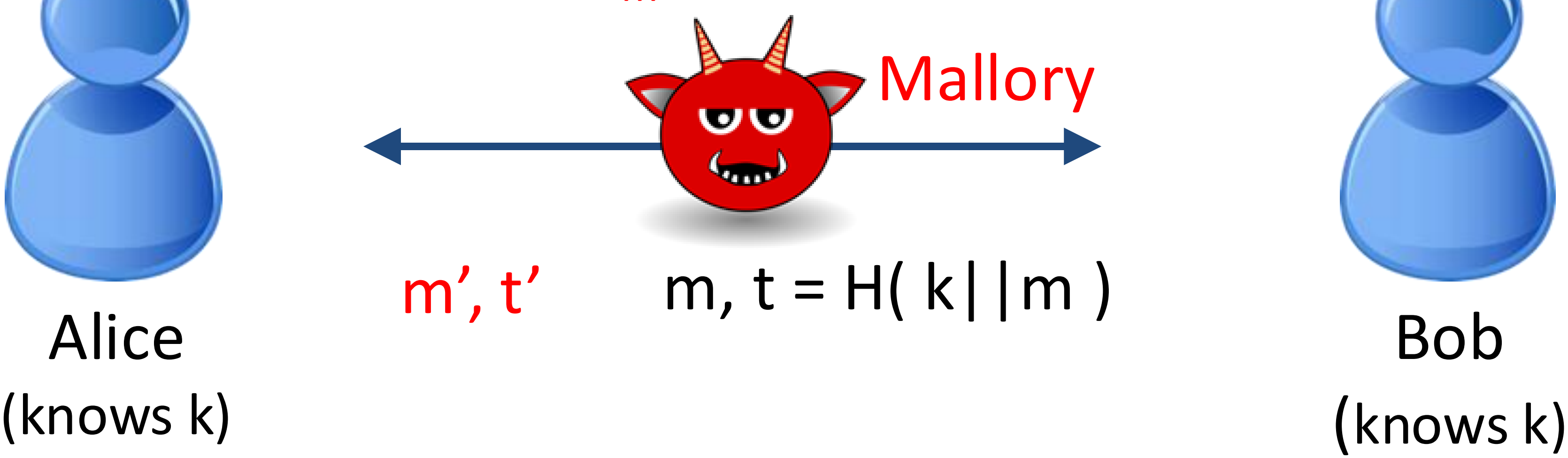


# Hash-Based MAC (HMAC)

- $t = H(\ k\ ||\ m\ )$ is secure if H is pseudorandom
- Otherwise, length extension attacks may apply
  - Given m and $t = H(\ k||m\ )$, Mallory can compute

    $t' = H(\ k||m||pad_m||ExtraData\ )$

    $m' = m||pad_m||\ ExtraData$



# Hash-Based MAC (HMAC)

- t = H( k || m ) is secure if H is pseudorandom
- Otherwise, length extension attacks may apply
  - Given m and t = H( k||m ), Mallory can compute

    $t' = H(\, k \| m \| pad_m \| ExtraData\, )$

    $m' = m \| pad_m \| ExtraData$

- What should we do?
  - Use SHA3! (No length extension in SHA3)
  - If one must use SHA2, then use H( k||m||k )

# Digital Signatures

# Digital Signatures

- Interface
  - KeyGen() → (vk, sk)
    - A private signing key and a public verification key
  - Sign(sk, m) → σ (called a signature)
  - Verify(vk, m, σ) → True/False

- How do we define security of signatures?

# Digital Signatures

- Unforgeability under Chosen Message Attacks
  - We invoke KeyGen() → (vk, sk)
  - Mallory can ask for *signatures* of any messages
  - The attacker Mallory wins if she can produce a **forgery** (m, σ) such that Verify(vk, m, σ) = True
    - m must be a message Mallory did not ask *signatures* for

# Asymmetric Encryption vs. Signature

- They are NOT inverse of each other, but may help with intuition for beginners
  - KeyGen() → (vk, sk) KeyGen() → (pk, sk)
  - Sign(sk, m) → σ Dec(sk, m) → σ
    - Treat m as ciphertext, signing key as decryption key
  - Verify(vk, m, σ) → T/F Enc(vk, σ) == m?
    - Treat σ as plaintext, verification key as encryption key

# Recommended Schemes

- Most widely used: RSA digital signature
  - (Previous analogy applies to some extent)
  - KeyGen(): $N = pq$, $ed \equiv 1 \bmod (p-1)(q-1)$
  - Sign($d$, $m$): $\sigma = H(m)^d \bmod N$
  - Verify($e$, $m$, $\sigma$): $H(m) \stackrel{?}{=} \sigma^e \bmod N$

- More recommended: Elliptic-Curve Digital Signature Algorithms (ECDSA)
  - Faster, shorter keys, no other secrets

# Confidentiality + Integrity

# Three Natural Methods for Confidentiality + Integrity

- Encrypt-and-MAC
  - (c, t) where c = Enc($k_1$, m) and t = MAC($k_2$, m)
  - m = Dec($k_1$, c), MAC($k_2$, m) ?= t
- Encrypt-then-MAC
  - (c, t) where c = Enc($k_1$, m) and t = MAC($k_2$, c)
  - MAC($k_2$, c) ?= t, if yes, m = Dec($k_1$, c)
- MAC-then-encrypt
  - c where t = MAC($k_2$, m) and c = Enc($k_1$, m||t)
  - m||t = Dec($k_1$, c), MAC($k_2$, m) ?= t

# Three Natural Methods for Confidentiality + Integrity

- ~~Encrypt-and-MAC~~
  - (c, t) where $c = Enc(k_1, m)$ and $t = MAC(k_2, m)$
  - $m = Dec(k_1, c)$, $MAC(k_2, m) ?= t$

- The MAC tag t may reveal information about m

# Three Natural Methods for Confidentiality + Integrity

- ~~Encrypt-and-MAC~~
- Encrypt-then-MAC
  - (c, t) where c = Enc($k_1$, m) and t = MAC($k_2$, c)
  - MAC($k_2$, c) ?= t, if yes, m = Dec($k_1$, c)
- MAC-then-encrypt
  - c where t = MAC($k_2$, m) and c = Enc($k_1$, m||t)
  - m||t = Dec($k_1$, c), MAC($k_2$, m) ?= t
- The other two are both secure in theory but MAC-then-encrypt is more bug-prone (MP3)

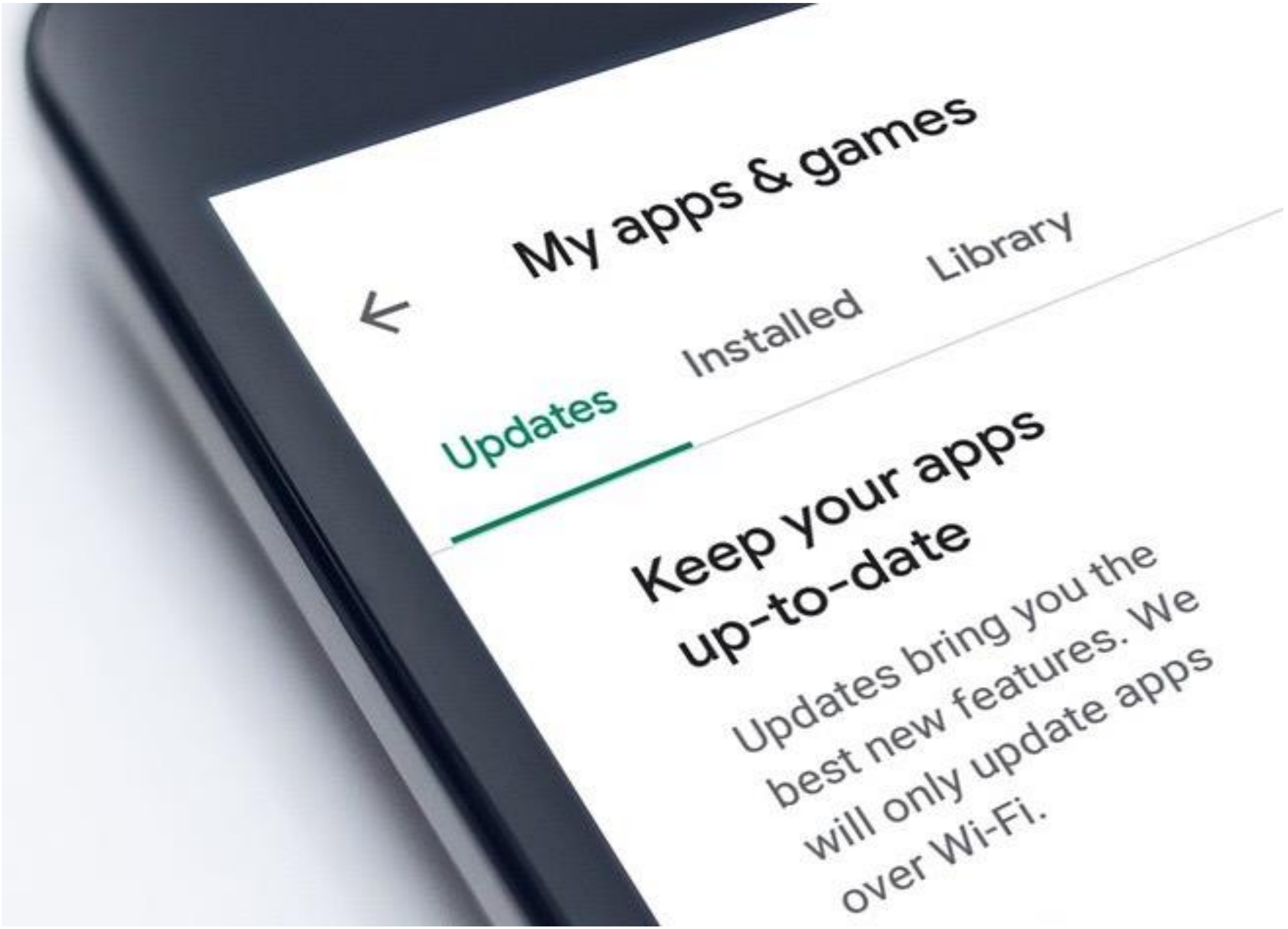
# Recommended Schemes for Confidentiality + Integrity

- Encrypt-then-MAC
  - E.g., c = AES-CTR($k_1$, m) and t = SHA3( $k_2 || c$ )
  - SHA3( $k_2 || c$ ) ?= t, if yes, m = AES-CTR($k_1$, c)

- Some block cipher modes provide confidentiality + integrity, e.g., AES-GCM
  - A legit argument for block cipher over stream cipher (which needs orthogonal mechanisms for integrity)

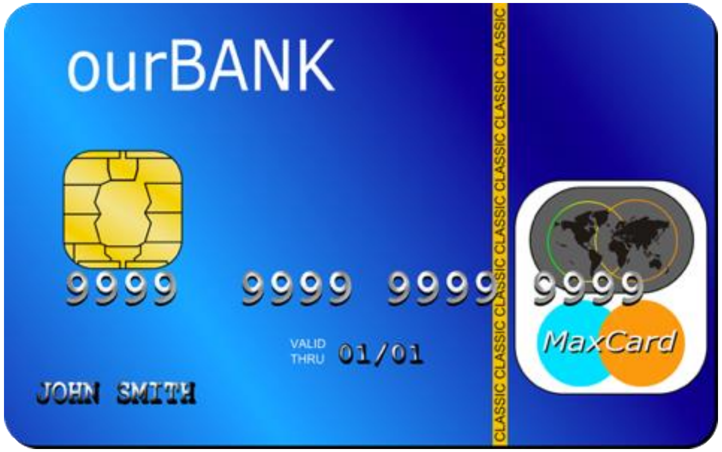
# Applications

# Software Updates



# Payment Card

What happens during …



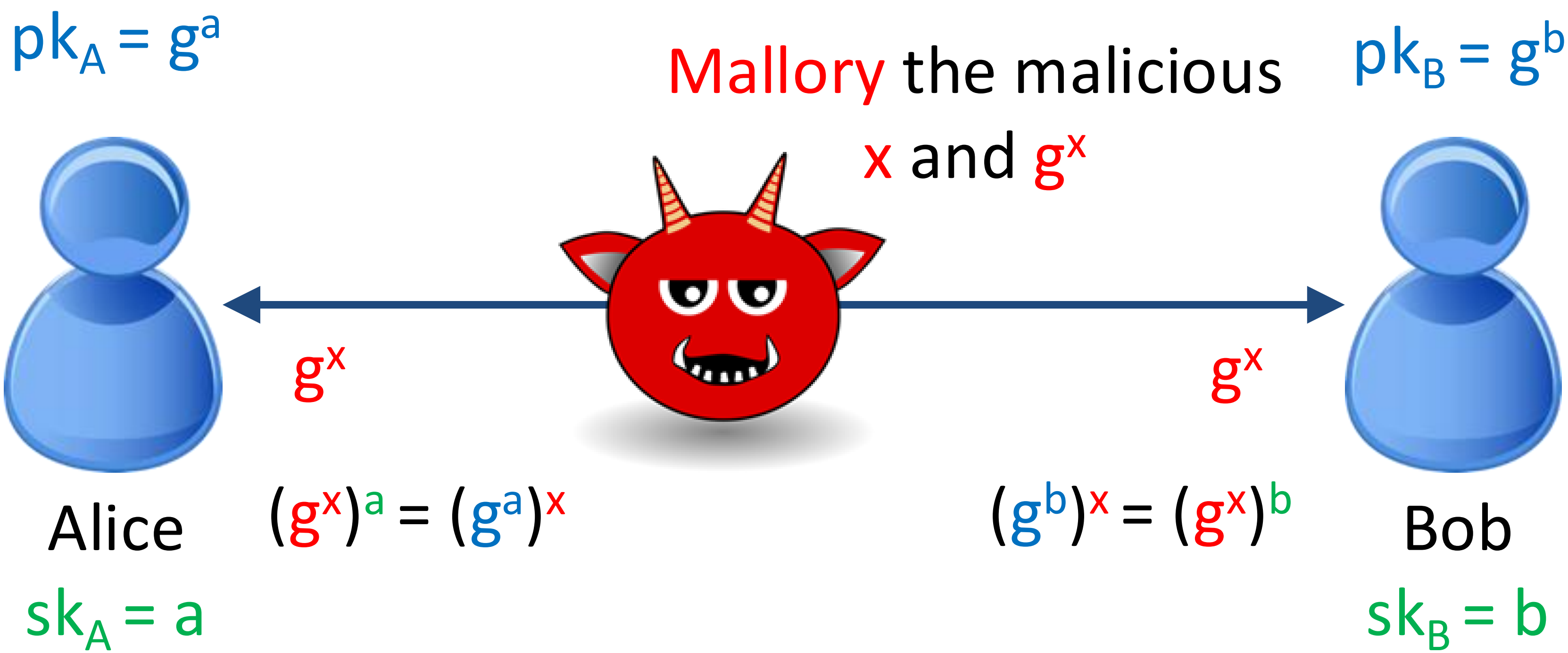


For comparison: What happens during card swipe?

MAC or signature?

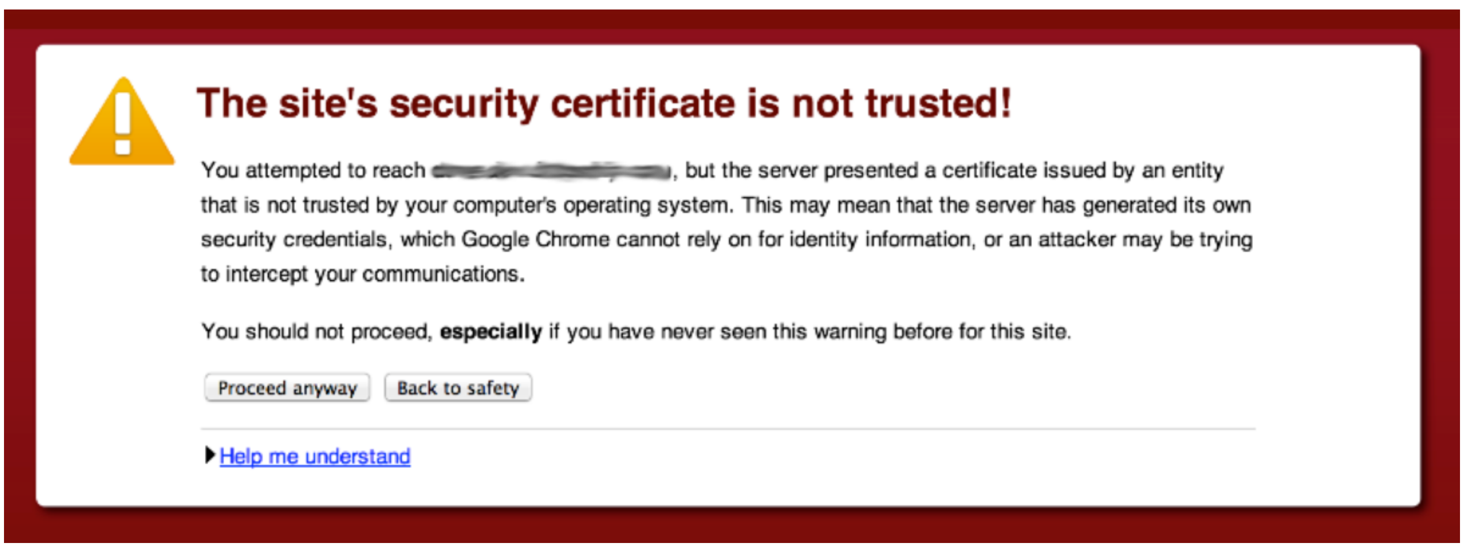
# Man-in-the-Middle (MitM) Attacks

- Asymmetric encryption and key exchange give two ways to establish a symmetric key
- However, both are susceptible to MitM
- At least one party's pk must be **certified**



# Website Certificates

- A trusted entity called Certificate Authority (CA) vouches for (signs) a website's public key
- If the signature is invalid, browser shows:



```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            0f:77:30:d4:eb:75:d6:c4:22:1e:4b:a1:f6:16:2b:83
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=US, O=DigiCert Inc, OU=www.digicert.com,
                CN=DigiCert High Assurance CA-3
        Validity
            Not Before: Sep  7 00:00:00 2012 GMT
            Not After : Nov 11 12:00:00 2015 GMT
        Subject: C=US, ST=California, L=La Jolla,
                 O=University of California, San Diego,
                 OU=ACT Data Center, CN=*.ucsd.edu
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
                Modulus (2048 bit):
                    00:cf:73:a9:a0:dd:69:de:98:c5:65:2d:fa:c0:dc:
                    47:ed:ff:f9:0b:16:3a:ee:e4:74:6a:de:26:37:7b:
                    ce:f7:de:3e:50:25:13:49:23:ec:c8:b3:19:5f:05:
                    9e:05:72:41:a9:f7:26:b3:d2:bd:88:37:51:e8:d5:
                    c3:01:d9:c2:15:bf:eb:87:a3:4b:80:3b:6c:f6:ce:
                    c5:78:4c:d2:b3:24:af:3d:8b:d8:ba:b9:c9:eb:16:
                    b4:83:68:06:b6:1e:96:0e:2e:1c:78:91:41:b4:8d:
                    3c:fe:2a:f5:93:ac:e5:bd:98:78:e5:db:4a:c2:88:
                    46:3a:1f:1e:07:fd:79:8a:96:c7:e9:b7:05:4d:40:
                    5d:4d:52:2c:e4:bc:6b:eb:2c:3e:09:e1:27:49:1b:
                    46:ab:53:cf:d9:df:8f:35:74:b4:40:1f:0b:7f:c1:
                    e4:ac:3d:5a:7b:98:e1:c4:fb:d1:e7:16:47:d9:ba:
                    51:28:1b:bf:77:f7:42:f2:dc:53:e2:38:18:b9:d2:
                    59:9a:e2:44:2a:cc:e5:99:60:a1:d1:dc:aa:2f:ba:
```

The issuing CA

Identify of the subject

Public key of the subject

```
TLS Web Server Authentication, TLS Web Client Authentication
           X509v3 CRL Distribution Points:
               URI:http://crl3.digicert.com/ca3-g14.crl
               URI:http://crl4.digicert.com/ca3-g14.crl

           X509v3 Certificate Policies:
               Policy: 2.16.840.1.114412.1.1
                 CPS: http://www.digicert.com/ssl-cps-repository.htm
                 User Notice:
                   Explicit Text:

           Authority Information Access:
               OCSP - URI:http://ocsp.digicert.com
               CA Issuers - URI:http://cacerts.digicert.com/DigiCertHighAssuranceCA-
3.crt

           X509v3 Basic Constraints: critical
               CA:FALSE
    Signature Algorithm: sha1WithRSAEncryption
        21:9f:9b:89:0d:43:02:0e:07:cd:dd:3c:2a:7b:aa:f2:4c:f2:
        5e:f4:fa:2f:74:db:38:0e:51:5c:76:fe:36:06:d7:6d:00:b3:
        aa:3a:4a:8c:c3:86:f1:61:c6:9d:35:4d:0c:17:c9:90:2c:8f:
        db:d8:f2:2b:46:37:00:ca:92:7b:25:86:17:b4:44:92:dc:a7:
        45:bc:1c:eb:2a:35:a5:03:bb:0b:57:c2:aa:22:a9:08:60:32:
        90:99:55:9b:c7:4c:99:25:6e:07:0d:ae:21:4a:b5:01:4e:dc:
        7e:eb:dc:3f:83:18:19:e8:b5:d1:22:e8:40:a6:61:17:6d:8a:
        cc:64:a9:ab:c3:31:d4:d3:90:db:18:14:1a:d4:8a:17:dd:0a:
        c7:c8:64:68:94:49:88:0a:1b:c2:9e:74:1a:23:15:96:91:10:
        50:13:ea:88:01:c9:79:12:93:19:29:27:12:78:9d:66:10:5c:
        72:bc:a4:f5:59:07:7a:0e:0c:69:09:ab:44:d8:24:39:ec:a3:
        53:8b:1b:18:25:aa:57:9e:e6:7a:64:87:0f:e8:6b:42:1f:ad:
        d1:38:0f:44:a8:a3:31:4f:bc:e8:74:cc:50:f6:69:10:4f:db:
```

RSA Signature by the CA (not encryption!)

# Summary of Message Integrity

- Security definition: UF-CMA
- MAC interface: t = MAC(k, m)
  - Recommend: HMAC (note length extension)
- Signature interface: Sign(sk, m), Verify(vk, m, σ)
  - Recommend: ECDSA
    - Inverse RSA with hash is OK (broken without hash)
- Confidentiality + Integrity: Encrypt-then-MAC or AES-GCM
- Applications: software update, payment card, website certificates, …