Message Authentication Codes

Presentation by:
V. Balasubramanian
SSN College of Engineering

Objectives

- Message Integrity
- Encryption vs. Message Authentication
- Message Authentication Codes
- Constructing Secure Message Authentication Codes

Message Authentication Codes

- The principle behind MACs
- The security properties that can be achieved with MACs
- How MACs can be realized with hash functions and with block ciphers

Goal

- Encryption can be used to prevent an eavesdropper from learning anything about the content of messages sent over an unprotected communication channel.
- To guarantee message integrity (or message authentication) in the sense that each party should be able to identify when a message it receives was sent by the party claiming to send it, and was not modified in transit.

Use Case

Consider the case of a user communicating with their bank over the Internet. When the bank receives a request to transfer $1,000 from the user's account to the account of some other user X, the bank has to consider the following:

1. Is the request authentic? That is, did the user in question really issue this request, or was the request issued by an adversary (perhaps X itself) who is impersonating the legitimate user?

2. Assuming a transfer request was issued by the legitimate user, are the details of the request as received exactly those intended by the legitimate user? Or was, e.g., the transfer amount modified?

TCP error correction

- Error-correcting codes are only intended to detect and recover from “random” errors that a□ect only a small portion of the transmission, but they do nothing to protect against a malicious adversary who can choose exactly where to introduce an arbitrary number of errors

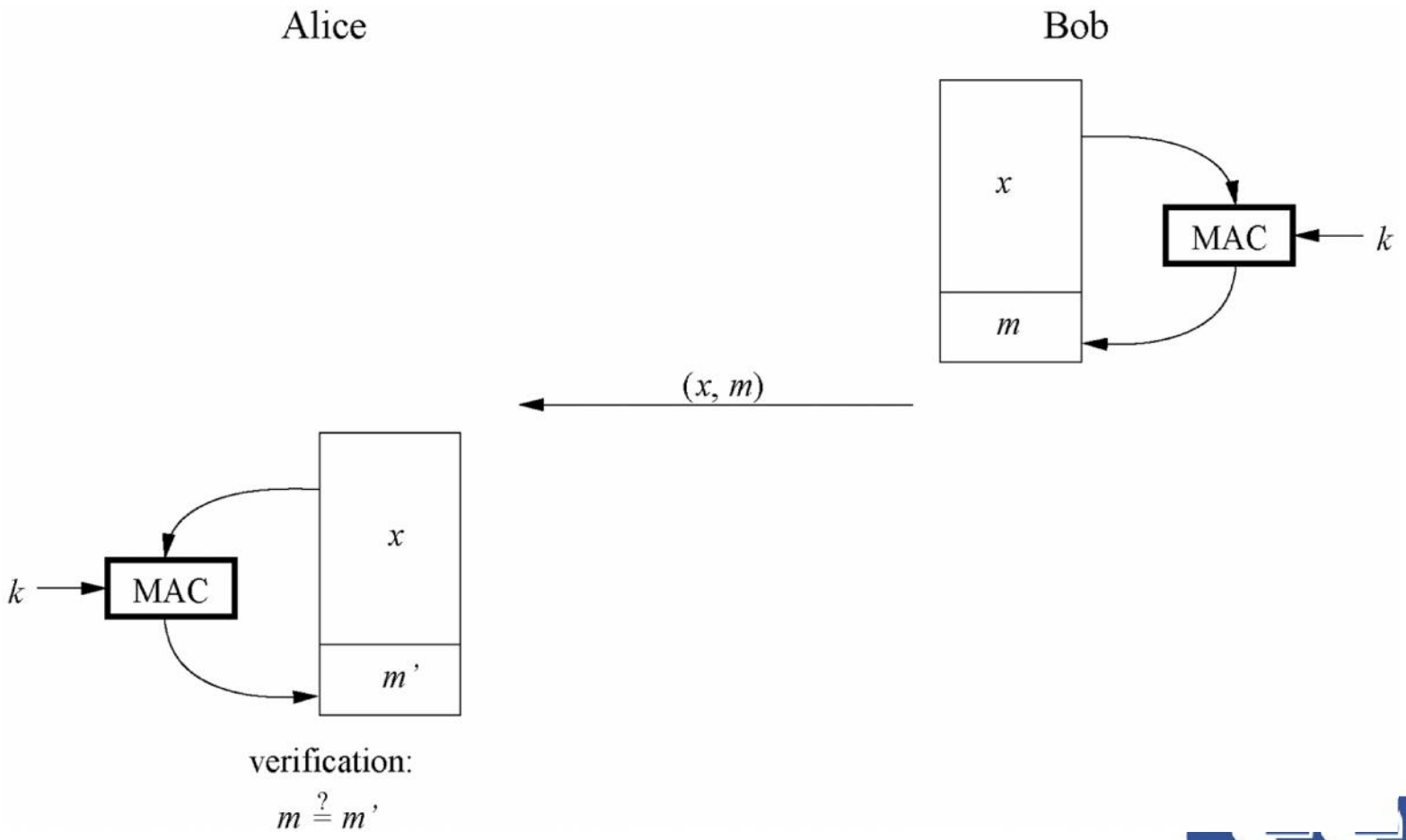
MAC

- To achieve message integrity by using cryptographic techniques to prevent the undetected tampering of messages sent over an open communication channel.

MAC

- A Message Authentication Code (MAC), also known as a cryptographic checksum or a keyed hash function
- Unlike digital signatures, MACs are symmetric-key schemes and they do not provide nonrepudiation
- Similar to digital signatures, MACs append an authentication tag to a message
- MACs use a symmetric key k for generation and verification
- Computation of a MAC: $m = MAC_k(x)$

MAC



Motivation

The motivation for using MACs is typically that Alice and Bob want to be assured that any manipulations of a message x in transit are detected. For this, Bob computes the MAC as a function of the message and the shared secret key k. He sends both the message and the authentication tag m to Alice. Upon receiving the message and m, Alice verifies both. Since this is a symmetric set-up, she simply repeats the steps that Bob conducted when sending the message: She merely recomputes the authentication tag with the received message and the symmetric key.

Message authentication codes

- Integrity as an orthogonal security concern
  - Secrecy and integrity are different
  - Encryption and message authentication are different
- Message authentication codes, and definition of security
- Basic MAC from any PRF
  - Short, fixed-length messages only

Introduction

- Unfortunately, secrecy and integrity are often confused and unnecessarily intertwined
- Encryption does not provide any integrity, and encryption should never be used with the intent of achieving message authentication unless it is specifically designed with that purpose in mind

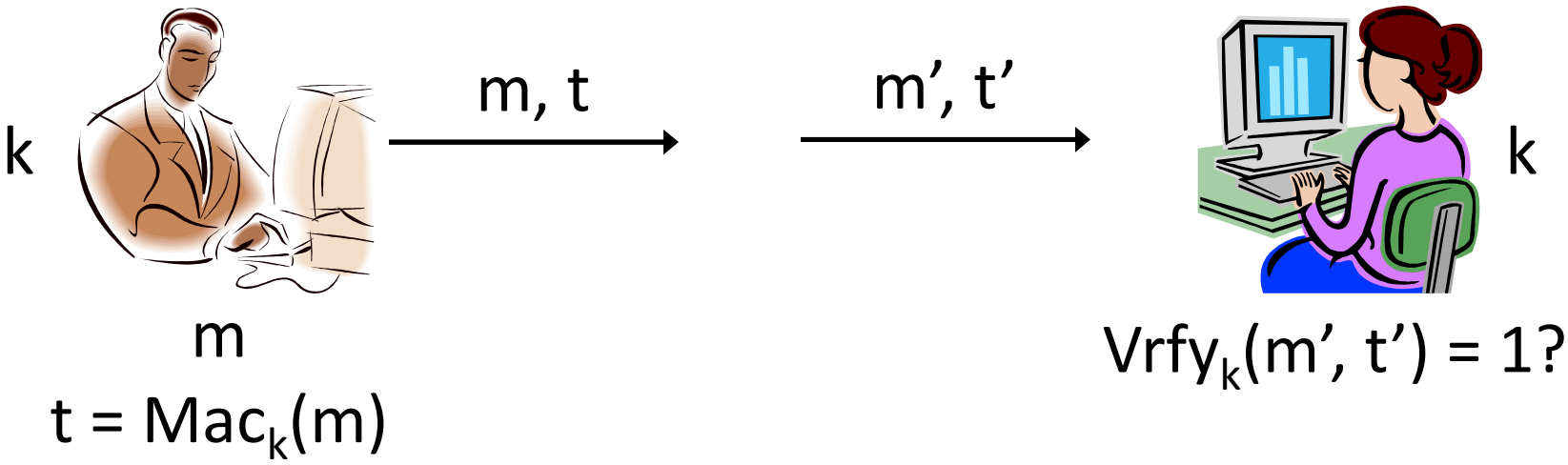
Introduction

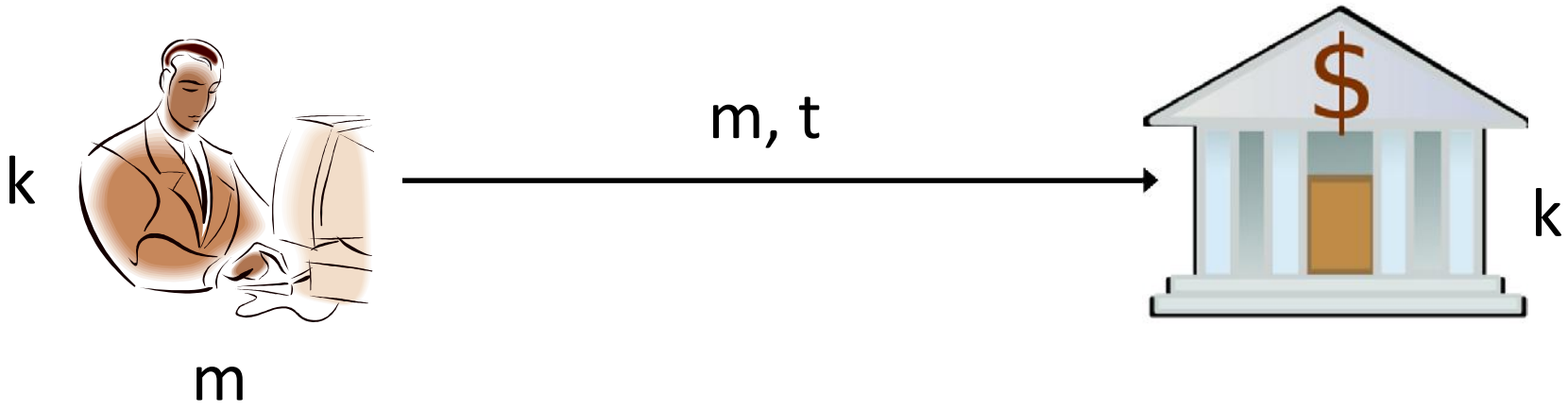
- Encryption solves the problem of message authentication.
- (In fact, this is a common error.) This is due to the fuzzy, and incorrect
- Ciphertext completely hides the contents of the message, an adversary cannot possibly modify an encrypted message in any meaningful way.
- This reasoning is completely false. The encryption schemes we have seen thus far do not provide message integrity.

Secrecy vs. integrity

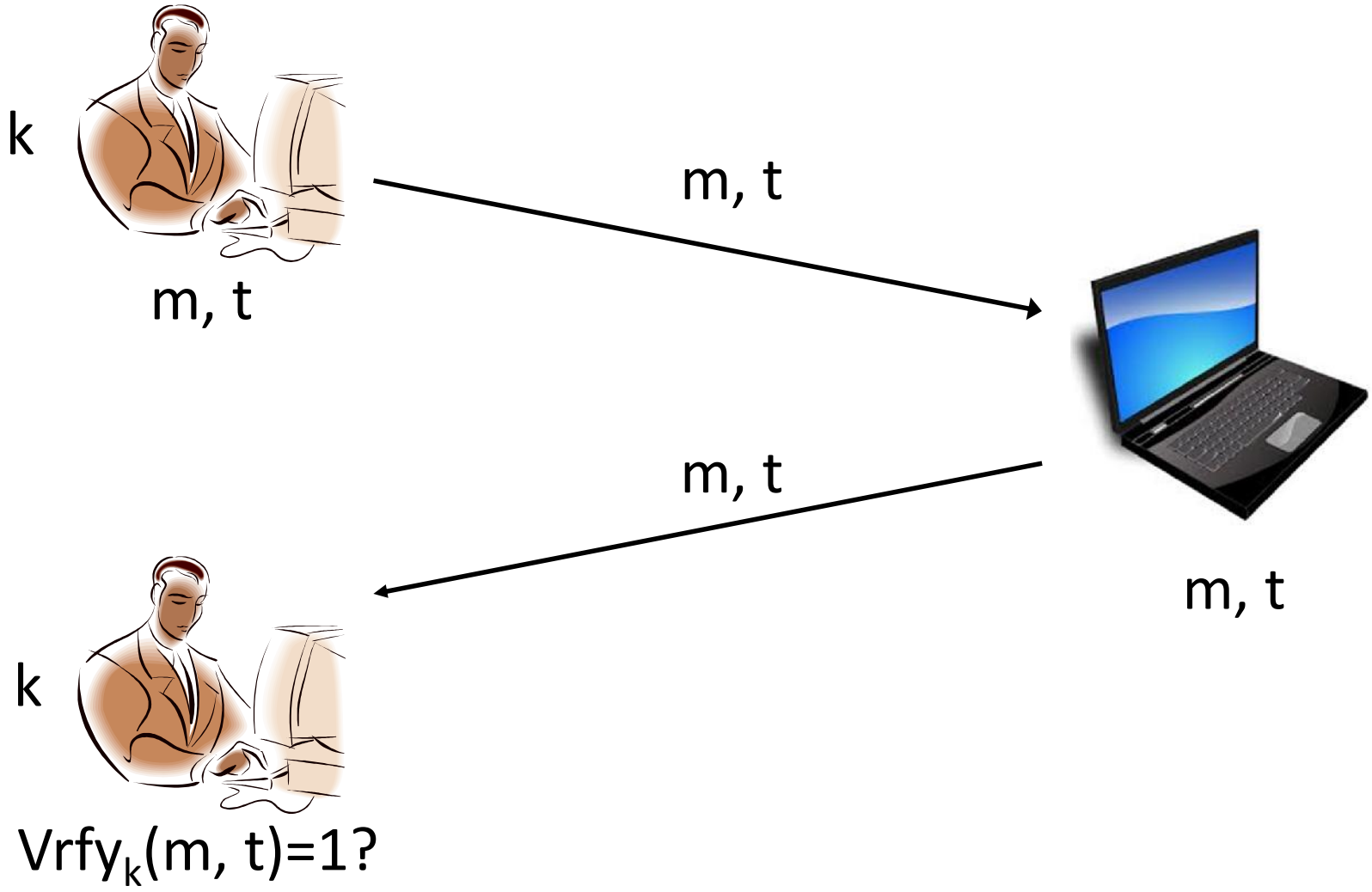
- So far we have been concerned with ensuring *secrecy* of communication

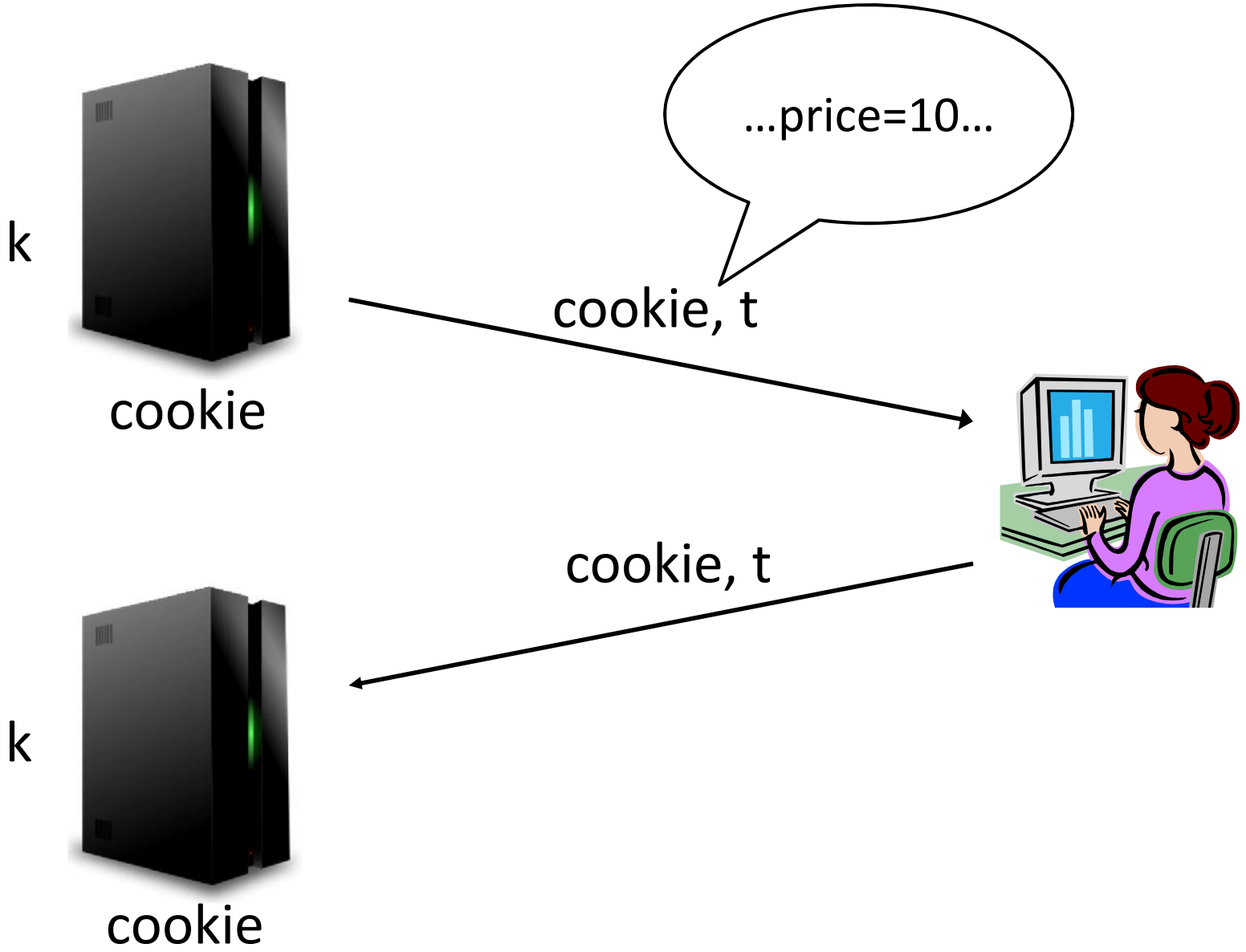
- What about *integrity*?
  - I.e., ensuring that a received message originated from the intended party, and was not modified
    - Even if an attacker controls the channel!
  - Standard error-correction techniques not enough!
    - The right tool is a *message authentication code*


k
m, t
m', t'
k
m
t = Mac_k(m)
Vrfy_k(m', t') = 1?



advance of their communication. When one party wants to send a message m to the other, she computes a MAC tag (or simply a tag) t based on the message and the shared key, and sends the message m and the tag t to the other party. The tag is computed using a *tag-generation algorithm* denoted by Mac; thus, rephrasing what we have already said, the sender of a message m computes $t \leftarrow \mathsf{Mac}_k(m)$ and transmits (m, t) to the receiver. Upon receiving (m, t), the


k
m, t
m, t
m, t
m, t
k
Vrfy$_k$(m, t)=1?


k
cookie
…price=10…
cookie, t
cookie, t
k
cookie

Secrecy vs. integrity

- Secrecy and integrity are *orthogonal* concerns
  - Possible to have either one without the other
  - Sometimes you might want one without the other
  - Most often, both are needed

- Encryption does not (in general) provide *any* integrity
  - Integrity is even stronger than non-malleability
  - None of the schemes we have seen so far provide any integrity

Message authentication code (MAC)

- A *message authentication code* is defined by three PPT algorithms (Gen, Mac, Vrfy):
  - Gen: takes as input 1^n; outputs k. (Assume $|k| \geq n$.)
  - Mac: takes as input key k and message $m \in \{0,1\}^*$; outputs tag t

    $$t := \mathrm{Mac}_k(m)$$

  - Vrfy: takes key k, message m, and tag t as input; outp

For all m and all k output by Gen,
$\mathrm{Vrfy}_k(m, \mathrm{Mac}_k(m)) = 1$

MAC Definition

DEFINITION 4.1 *A* message authentication code *(or* MAC*) consists of three probabilistic polynomial-time algorithms* (Gen, Mac, Vrfy) *such that:*

1. *The* key-generation algorithm Gen *takes as input the security parameter* 1^n *and outputs a key* k *with* $|k| \geq n$.

2. *The* tag-generation algorithm Mac *takes as input a key* k *and a message* $m \in \{0,1\}^*$*, and outputs a tag* t*. Since this algorithm may be randomized, we write this as* $t \leftarrow \mathsf{Mac}_k(m)$.

3. *The deterministic* verification algorithm Vrfy *takes as input a key* k*, a message* m*, and a tag* t*. It outputs a bit* b*, with* $b = 1$ *meaning* valid *and* $b = 0$ *meaning* invalid*. We write this as* $b := \mathsf{Vrfy}_k(m, t)$.

Security?

- Only one standard definition
- Threat model
  - "Adaptive chosen-message attack"
  - Assume the attacker can induce the sender to authenticate *messages of the attacker's choice*
- Security goal
  - "Existential unforgeability"
  - Attacker should be unable to forge a valid tag on *any* message not previously authenticated by the sender

k

$t_1 := Mac_k(m_1)$

$t_2 := Mac_k(m_2)$

…

$t_i := Mac_k(m_i)$

m_1, t_1 →

m_2, t_2 →

⋮

m_i, t_i →

m', t' →

k

$Vrfy_k(m', t')$??

Formal definition

- Fix A, Π
- Define randomized experiment $\text{Forge}_{A,\Pi}(n)$:
  1. $k \leftarrow \text{Gen}(1^n)$
  2. A interacts with an oracle $\text{Mac}_k(\cdot)$; let M be the set of messages submitted to this oracle
  3. A outputs (m, t)
  4. A *succeeds*, and the experiment evaluates to 1, if $\text{Vrfy}_k(m, t)=1$ and $m \notin M$

Formal Definition

The message authentication experiment $\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n)$**:**

1. *A key* k *is generated by running* $\mathsf{Gen}(1^n)$.
2. *The adversary* $\mathcal{A}$ *is given input* 1^n *and oracle access to* $\mathsf{Mac}_k(\cdot)$. *The adversary eventually outputs* (m, t). *Let* $\mathcal{Q}$ *denote the set of all queries that* $\mathcal{A}$ *asked its oracle.*
3. $\mathcal{A}$ succeeds *if and only if (1)* $\mathsf{Vrfy}_k(m, t) = 1$ *and (2)* $m \notin \mathcal{Q}$. *In that case the output of the experiment is defined to be* 1.

A MAC is secure if no efficient adversary can succeed in the above experiment with non-negligible probability:

Goal

A MAC satisfying the level of security specified above is said to be *existentially unforgeable under an adaptive chosen-message attack*. "Existential unforgeability" refers to the fact that the adversary must not be able to forge a valid tag on *any* message, and "adaptive chosen-message attack" refers to the fact that the adversary is able to obtain MAC tags on arbitrary messages chosen adaptively during its attack.

Unforgeable

DEFINITION 4.2 *A message authentication code* $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ *is* existentially unforgeable under an adaptive chosen-message attack, *or just* secure, *if for all probabilistic polynomial-time adversaries* $\mathcal{A}$, *there is a negligible function* negl *such that:*

$$\Pr[\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n) = 1] \leq \mathsf{negl}(n).$$

Security for MACs

- Π is *secure* if for all PPT attackers A, there is a negligible function ε such that

$$\Pr[\text{Forge}_{A,\Pi}(n) = 1] \leq \varepsilon(n)$$

Tools

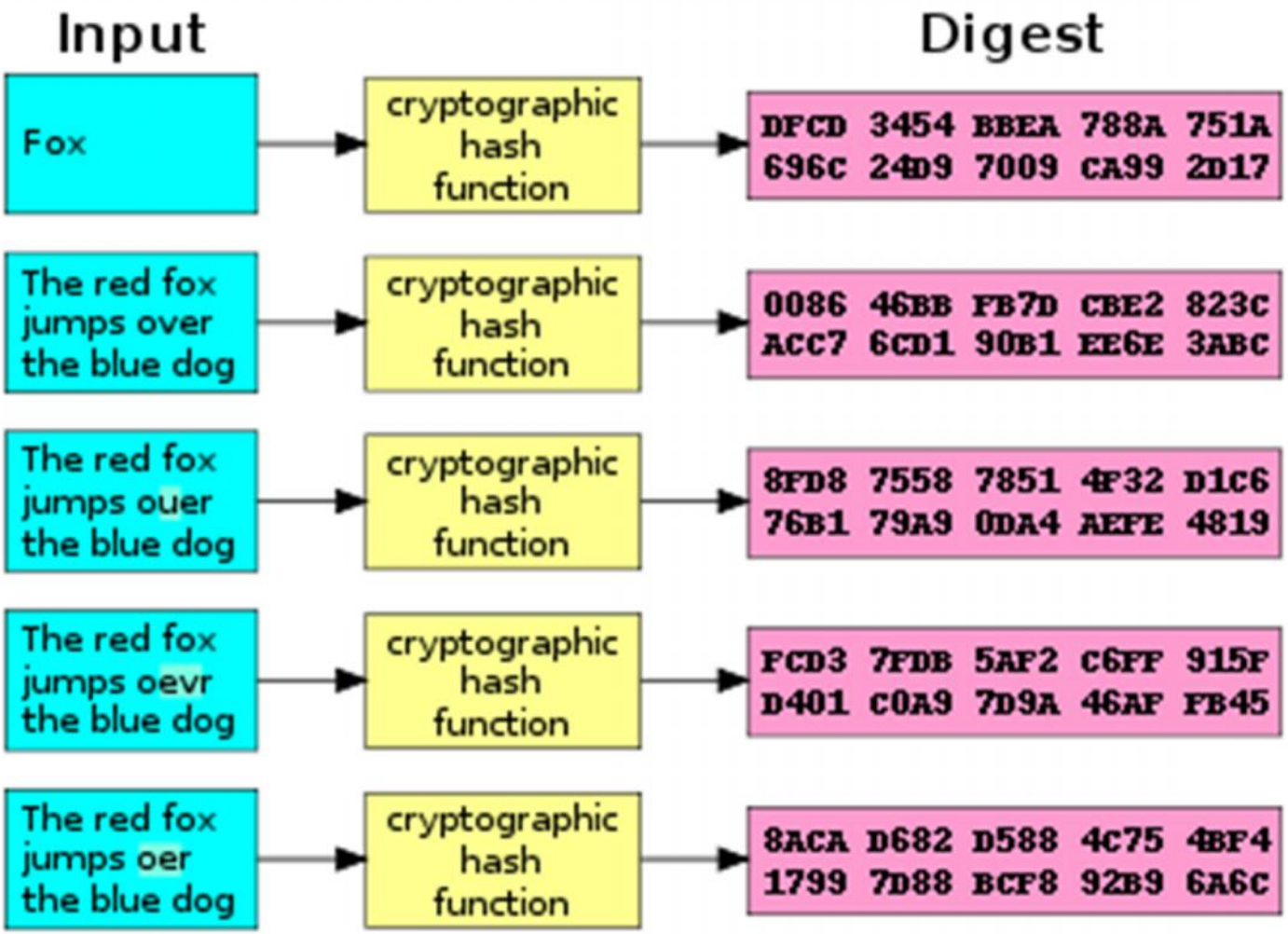
- WinHex: Computer Forensics & Data Recovery Software, Hex Editor & Disk Editor
- Openssl
- Boringssl – Google
- S2n - Amazon

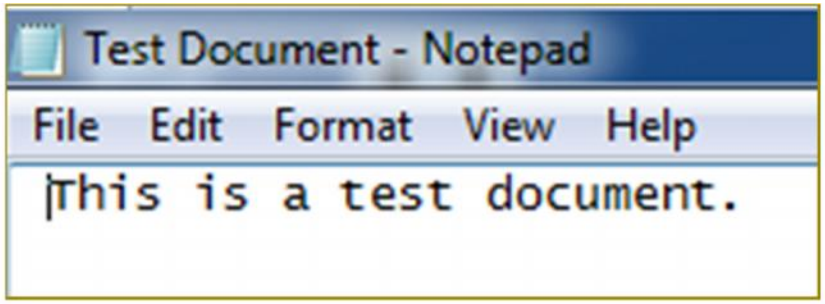
BoringSSL

amazon web services

s2n

OpenSSL
Cryptography and SSL/TLS Toolkit


Input
Digest
Fox
cryptographic hash function
DFCD 3454 BBEA 788A 751A 696C 24D9 7009 CA99 2D17
The red fox jumps over the blue dog
cryptographic hash function
0086 46BB FB7D CBE2 823C ACC7 6CD1 90B1 EE6E 3ABC
The red fox jumps ouer the blue dog
cryptographic hash function
8FD8 7558 7851 4F32 D1C6 76B1 79A9 0DA4 AEFE 4819
The red fox jumps oevr the blue dog
cryptographic hash function
FCD3 7FDB 5AF2 C6FF 915F D401 C0A9 7D9A 46AF FB45
The red fox jumps oer the blue dog
cryptographic hash function
8ACA D682 D588 4C75 4BF4 1799 7D88 BCF8 92B9 6A6C



Hash Values of above document	
MD5 Hash (128 bit)	7EB7781398042342E50BC37E93CCC854
SHA 1 **(160 bit)**	**CB80455993111C16FD13E70125852AEFC911F31E**
SHA 256 **(256 bit)**	**EC7F4FEDDD1C1349AD5A4D9C913A5C4E21A226E6719CD1B2805225DF7075D22F**

Evidence Preserving

- Examples of hash functions are MD5 and SHA-1, SHA-2.
- MD5 was developed by Professor Ronald L. Rivest of MIT.
- The MD5 algorithm takes as input a message of arbitrary length and produces as output a 128-bit fingerprint of the input.

Example

=

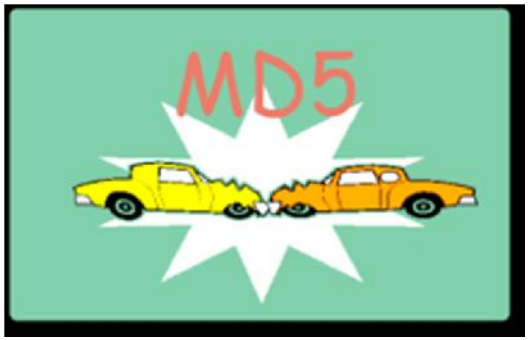
79054025
255fb1a2
6e4bc422
aef54eb4

MD5

[redacted]

[redacted]

MD5 Collision



d131dd02c5e6eec4693d9a0698aff95c2fcab58712467eab4004583eb8fb7f89
55ad340609f4b30283e4888325 71415a085125e8f7cdc99fd91dbdf280373c5b
d8823e3156348f5bae6dacd436c919c6dd53e2b487da03fd02396306d248cda0
e99f33420f577ee8ce54b67080a80d1ec69821bcb6a8839396f9652b6ff72a70

and

d131dd02c5e6eec4693d9a0698aff95c2fcab50712467eab4004583eb8fb7f89
55ad340609f4b30283e4888325f1415a085125e8f7cdc99fd91dbd7280373c5b
d8823e3156348f5bae6dacd436c919c6dd53e23487da03fd02396306d248cda0
e99f33420f577ee8ce54b6708028 0d1ec69821bcb6a8839396f965ab6ff72a70


A 30-23-12
A 30-23-05
ENTERING VEHICLE CONTROL ZONE

SHA

- SHA stands for Secure Hash Algorithm.
- The SHA hash functions are a set of cryptographic hash functions designed by the National Security Agency (NSA).
- The five algorithms are denoted SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512. SHA-1 produces a message digest that is 160 bits long; the number in the other four algorithms' names denote the bit length of the digest they produce.

Tools

- md5sum tool produces an md5 message digest (hash value).
- hashcalc application can also create hash values using different hashing methods.
- WinHex can calculate hash value of entire drive
- The hashing is done on the data itself, and not on the names of files

Security?

- Is the definition too strong?
  - We don't want to make any assumptions about what the sender might authenticate
  - We don't want to make any assumptions about what forgeries are "meaningful"
- A MAC satisfying this definition can be used anywhere integrity is needed

Replay Attacks

Replay attacks. We emphasize that the above definition, and message authentication codes on their own, offer no protection against *replay attacks* whereby a previously sent message (and its MAC tag) are replayed to an honest party. Nevertheless, replay attacks are a serious concern! Consider again the scenario where a user (say, Alice) sends a request to her bank to transfer $1,000 from her account to some other user (say, Bob). In doing so, Alice can compute a MAC tag and append it to the request so the bank knows the request is authentic. If the MAC is secure, Bob will be unable to intercept the request and change the amount to $10,000 because this would involve forging a valid tag on a previously unauthenticated message. However, nothing prevents Bob from intercepting Alice's message and replaying it *ten times* to the bank. If the bank accepts each of these messages, the net effect is that $10,000 will be transferred to Bob's account rather than the desired $1,000.

Replay attacks

- Note that *replay attacks* are not prevented
  - No stateless mechanism can prevent them
- Replay attacks are often a significant real-world concern
- Need to protect against replay attacks at a higher level
  - Decision about what to do with a replayed message is application-dependent

Counter Measures

Two common techniques for preventing replay attacks are to use *sequence numbers* (also known as *counters*) or *time-stamps*. The first approach, described (in a more general context) in Section 4.5.3, requires the communicating users to maintain (synchronized) state, and can be problematic when users communicate over a lossy channel where messages are occasionally dropped (though this problem can be mitigated). In the second approach, using time-stamps, the sender prepends the current time T (say, to the nearest millisecond) to the message before authenticating, and sends T along with the message and the resulting tag t. When the receiver obtains T, m, t, it verifies that t is a valid tag on $T \| m$ and that T is within some acceptable clock skew from the current time T' at the receiver. This method has certain drawbacks as well, including the need for the sender and receiver to maintain closely synchronized clocks, and the possibility that a replay attack can still take place if it is done quickly enough (specifically, within the acceptable time window).

Strong MACs

Formally, we consider a modified experiment Mac-sforge that is defined in exactly the same way as Mac-forge, except that now the set $\mathcal{Q}$ contains *pairs* of oracle queries and their associated responses. (That is, $(m, t) \in \mathcal{Q}$ if $\mathcal{A}$ queried $\mathsf{Mac}_k(m)$ and received in response the tag t.) The adversary $\mathcal{A}$ succeeds (and experiment Mac-sforge evaluates to 1) if and only if $\mathcal{A}$ outputs (m, t) such that $\mathsf{Vrfy}_k(m, t) = 1$ and $(m, t) \notin \mathcal{Q}$.

DEFINITION 4.3 *A message authentication code* $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ *is* strongly secure, *or a* strong MAC, *if for all probabilistic polynomial-time adversaries* $\mathcal{A}$, *there is a negligible function* negl *such that:*

$$\Pr[\mathsf{Mac\text{-}sforge}_{\mathcal{A},\Pi}(n) = 1] \leq \mathsf{negl}(n).$$

Properties of MAC

1. Cryptographic checksum
   A MAC generates a cryptographically secure authentication tag for a given message.
2. Symmetric
   MACs are based on secret symmetric keys. The signing and verifying parties must share a secret key.
3. Arbitrary message size
   MACs accept messages of arbitrary length.
4. Fixed output length
   MACs generate fixed-size authentication tags.
5. Message integrity
   MACs providemessage integrity: Any manipulations of a message during transit will be detected by the receiver.
6. Message authentication
   The receiving party is assured of the origin of the message.
7. No nonrepudiation
   Since MACs are based on symmetric principles, they do not provide nonrepudiation.

A fixed-length MAC

Intuition?

- We need a keyed function Mac such that:
  - Given $Mac_k(m_1)$, $Mac_k(m_2)$, …,
  - …it is infeasible to predict the value $Mac_k(m)$ for any $m \notin \{m_1, \ldots, \}$
- Let Mac be a pseudorandom function!

Construction

- Let F be a length-preserving pseudorandom function (aka block cipher)

- Construct the following MAC Π:
  - Gen: choose a uniform key k for F
  - $Mac_k(m)$: output $F_k(m)$
  - $Vrfy_k(m, t)$: output 1 iff $F_k(m)=t$

- Theorem: Π is a secure MAC

Message authentication codes

- Constructing a MAC on longer messages?
  - Different attacks to watch out for
- (Basic) CBC-MAC
  - Secure for fixed-length messages
- CBC-MAC
  - Secure for arbitrary-length messages
  - Used in the real world

Authenticated encryption

- Communication with secrecy and integrity
- An AE scheme is an encryption scheme that achieves both
  - CCA-security
  - Unforgeability
- Encrypt and authenticate is not a sound generic construction

Authenticated encryption

Secrecy + integrity?

- We have shown primitives for achieving *secrecy* and *integrity* in the private-key setting

- What if we want to achieve both?

Authenticated encryption

- An encryption scheme that achieves both secrecy and integrity
- Secrecy notion: CCA-security
- Integrity notion: unforgeability
  - Adversary cannot generate ciphertext that decrypts to a previously unencrypted message

Generic constructions

- Generically combine an encryption scheme and a MAC
  - Useful when these are already available in some library

- Goal: the combination should be an authenticated encryption scheme when instantiated with *any* CPA-secure encryption scheme and *any* secure MAC

Generic constructions?

- Encrypt and authenticate
- Authenticate then encrypt
- Encrypt then authenticate

Encrypt and authenticate

k1, k2

m

$c \leftarrow Enc_{k1}(m)$

$t = Mac_{k2}(m)$

c, t

k1, k2

$m = Dec_{k1}(c)$

$Vrfy_{k2}(m, t) = 1?$

Problems

- The tag t might leak information about m!
  - Nothing in the definition of security for a MAC implies that it hides information about m
  - So the combination may not even be EAV-secure
- If the MAC is deterministic (as is CBC-MAC), then the tag leaks whether the same message is encrypted twice
  - I.e., the combination will not be CPA-secure

Authenticate then encrypt

k1, k2

c

k1, k2

m

$t = Mac_{k2}(m)$

$c \leftarrow Enc_{k1}(m \mid t)$

$m \mid t = Dec_{k1}(c)$

$Vrfy_{k2}(m, t) = 1?$

Problems

- Padding-oracle attack still works (if possible to distinguish padding failure from MAC failure)
- Other counterexamples are also possible
- The combination may not be CCA-secure

Encrypt then authenticate

k1, k2

m

$c \leftarrow Enc_{k1}(m)$

$t = Mac_{k2}(c)$

c, t

k1, k2

$Vrfy_{k2}(c, t) = 1?$

$m = Dec_{k1}(c)$

Security?

- If the encryption scheme is CPA-secure and the MAC is secure (with unique tags) then this is an authenticated encryption scheme

- It achieves something even stronger:
  - Given ciphertexts corresponding to (chosen) plaintexts $m_1,\ldots, m_k$, it is infeasible for an attacker to generate *any* new, valid ciphertext!

Authenticated encryption

- Encrypt-then-authenticate (with independent keys) is the recommended generic approach for constructing authenticated encryption

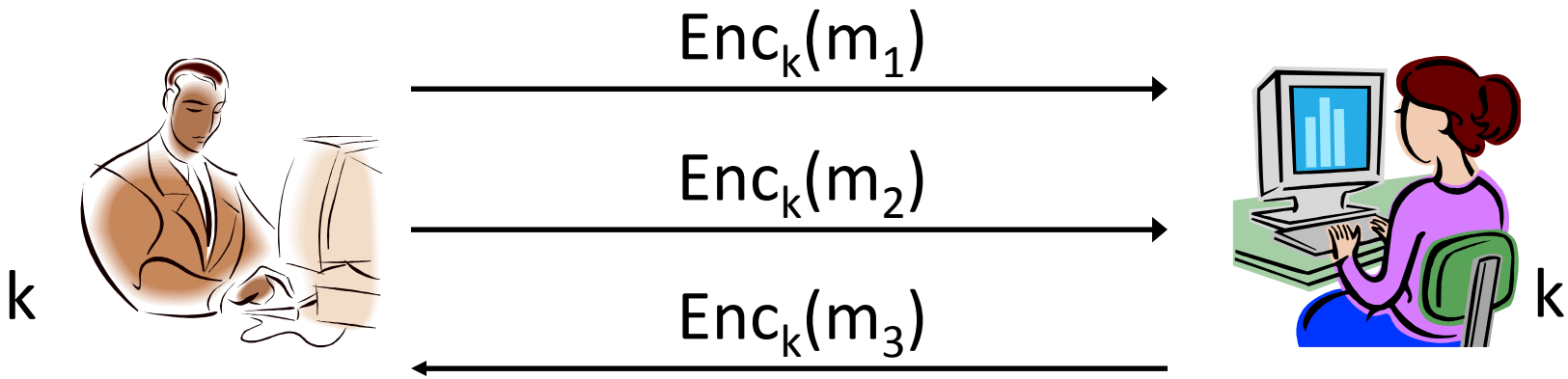
Direct constructions

- Other, more-efficient constructions have been proposed and are an active area of research and standardization
- E.g., OCB, CCM, GCM
- Others…
  - Active competition: https://competitions.cr.yp.to/caesar.html

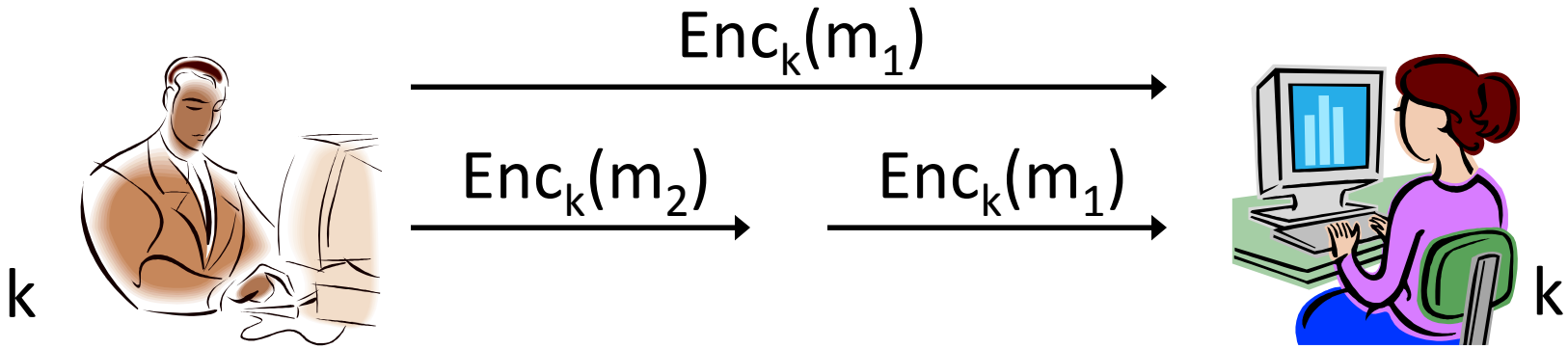
Secure sessions

Secure sessions?

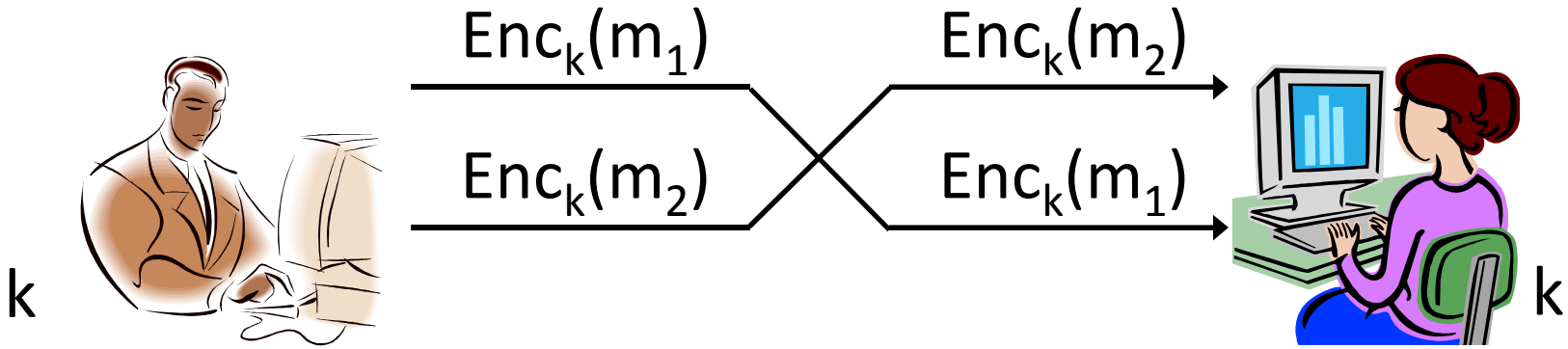
- Consider parties who wish to communicate securely over the course of a session
  - "Securely" = secrecy and integrity
  - "Session" = period of time over which the parties are willing to maintain state
- Can use authenticated encryption…


$Enc_k(m_1)$
$Enc_k(m_2)$
$Enc_k(m_3)$
k
k

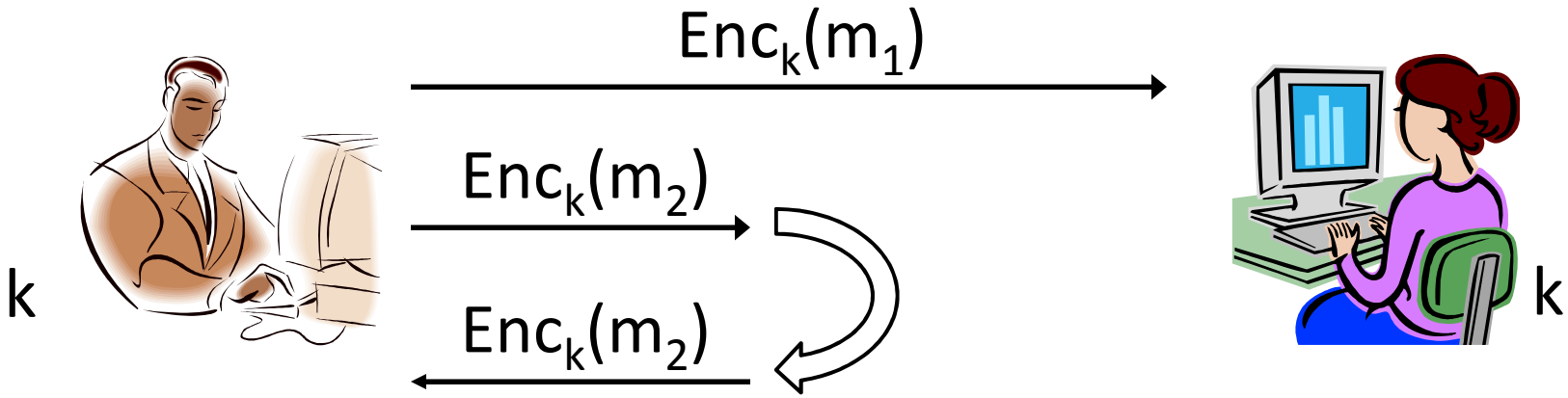
Replay attack



Re-ordering attack

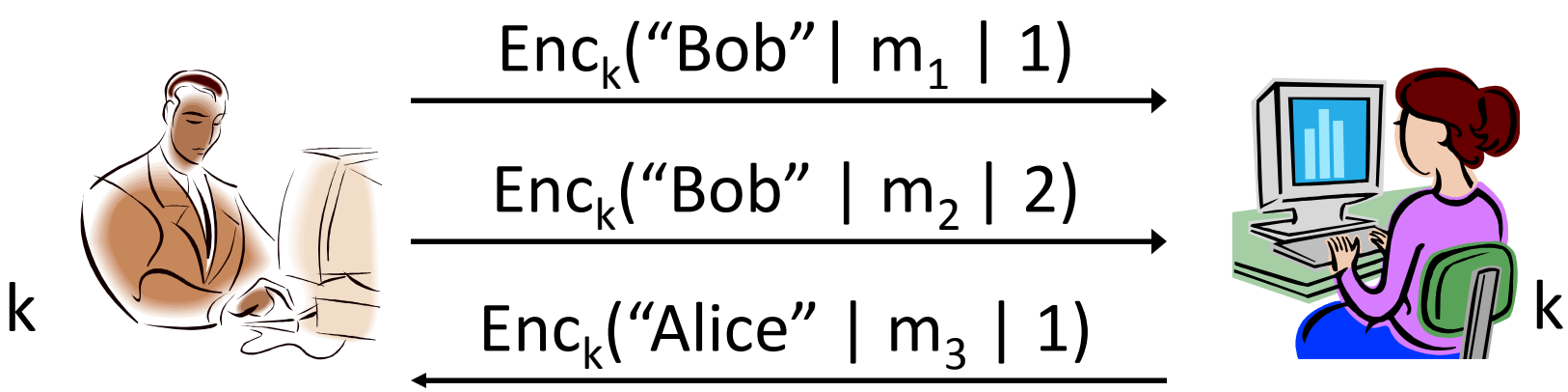


Reflection attack



Secure sessions

- These attacks (and others) can be prevented using *counters/sequence numbers* and *identifiers*


Enc_k("Bob"| m_1 | 1)
Enc_k("Bob" | m_2 | 2)
Enc_k("Alice" | m_3 | 1)
k
k

Secure sessions

- These attacks (and others) can be prevented using *counters* and *identifiers*
  - Can also use a *directionality bit* in place of identifiers