HMAC

Introduction

- A *Message Authentication Code (MAC), also known as a cryptographic checksum* or a *keyed hash function, is widely used in practice.*
- *In terms of security functionality,* MACs share some properties with digital signatures, since they also provide message integrity and message authentication.
- However, unlike digital signatures, MACs are symmetric-key schemes and they do not provide nonrepudiation.

Contd…

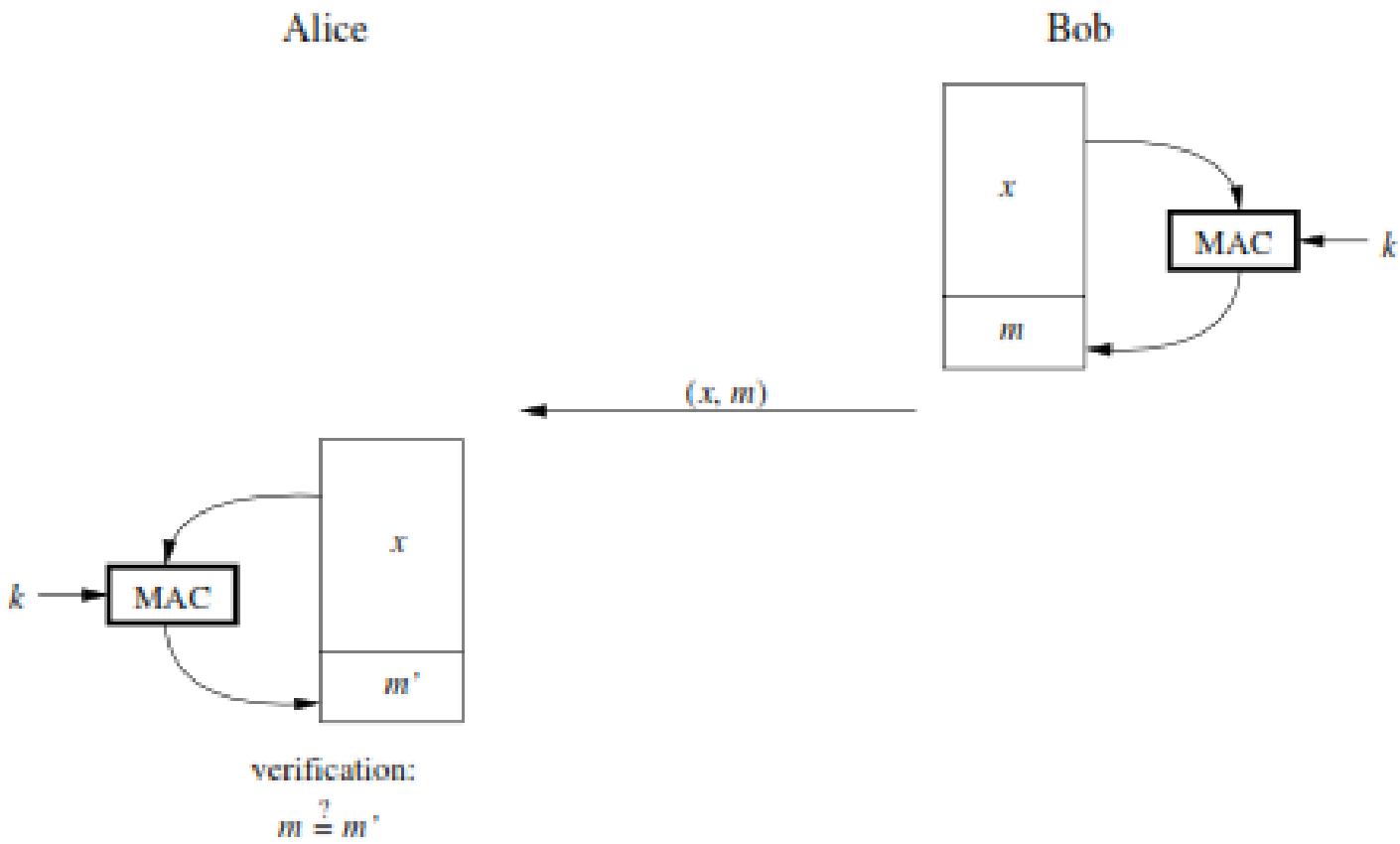
- One advantage of MACs is that they are much faster than digital signatures since they are based on either
- block ciphers or hash functions.

HMAC Principle

- Similar to digital signatures, MACs append an authentication tag to a message.
- The crucial difference between MACs and digital signatures is that MACs use a symmetric key *k for both generating the authentication tag and verifying it.*
- *A MAC is a* function of the symmetric key *k and the message x.*

$$m = \mathrm{MAC}_k(x)$$

Principle for MAC



Contd…

- 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.

Contd…

- The underlying assumption of this system is that the MAC computation will yield an incorrect result if the message x was altered in transit. Hence, message integrity isprovided as a security service. Furthermore, Alice is now assured that Bob was the originator of the message since only the two parties with the same secret key k have the possibility to compute the MAC.
- If an adversary, Oscar, changes the message during transmission, he cannot simply compute a valid MAC since he lacks the secret key. Any malicious or accidental (e.g., due to transmission errors) forgery of the message will be detected by the receiver due to a failed verification of the MAC.

Properties

Properties of Message Authentication Codes

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 provide message 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.

MAC’s from Hash functions: HMAC

- An option for realizing MACs is to use cryptographic hash functions such as SHA-1 as a building block.
- One possible construction, named HMAC, has become very popular in practice over the last decade. For instance, it is used in both the Transport Layer Security (TLS) protocol (indicated by the little lock symbol in your Web browser) as well as in the IPsec protocol suite.

- The basic idea behind all hash-based message authentication codes is that the key is hashed together with the message. Two obvious constructions are possible. The
- first one:
- $m = MAC_k(x)=h(k||x)$
- is called *secret prefix MAC, and the second one:*
- $m = MAC_k(x)=h(x||k)$
- is known as *secret suffix MAC.*

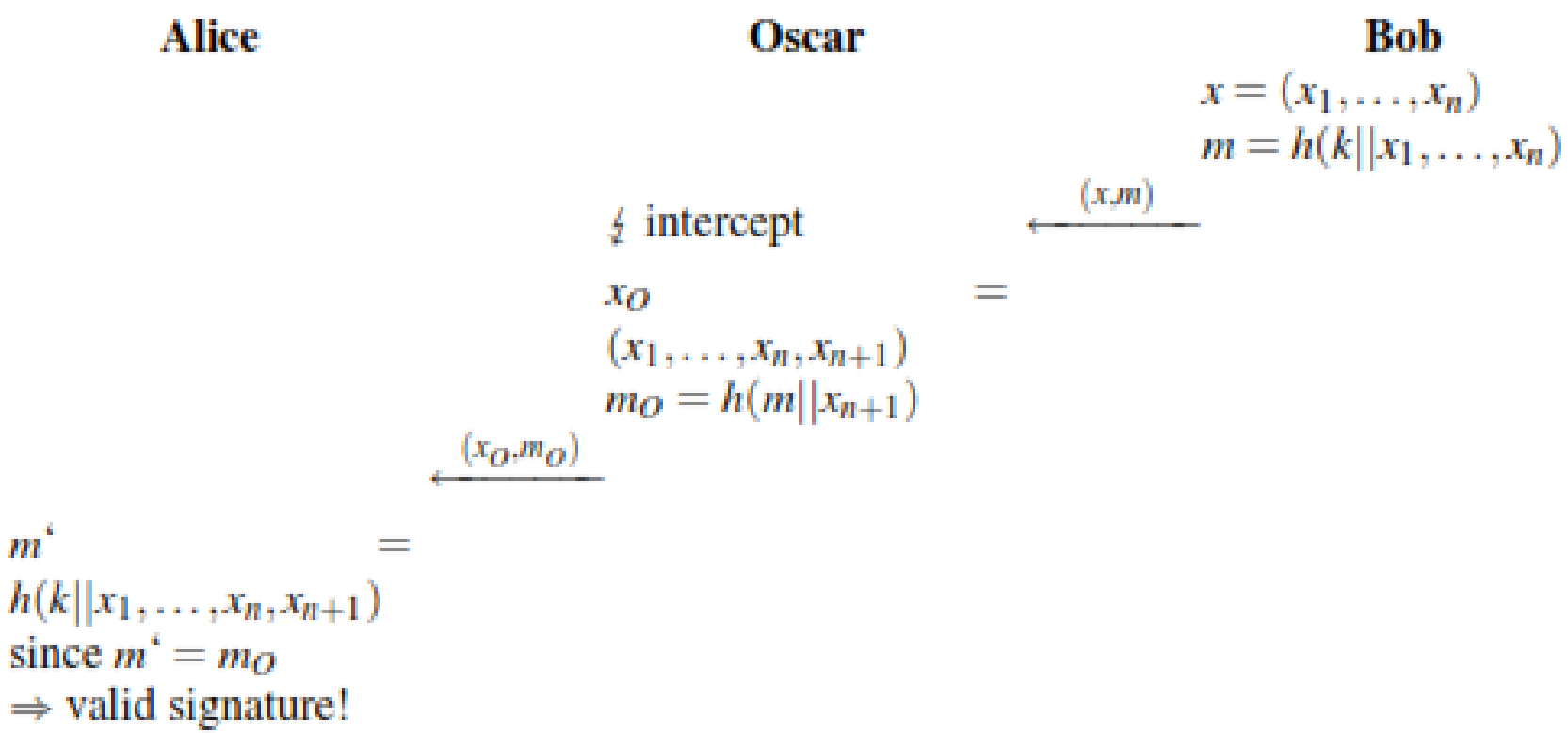
Attack

The message x that Bob wants to sign is a sequence of blocks $x = (x_1, x_2, \ldots, x_n)$, where the block length matches the input width of the hash function. Bob computes an authentication tag as:

$$m = \mathrm{MAC}_K(x) = h(k||x_1, x_2, \ldots, x_n)$$

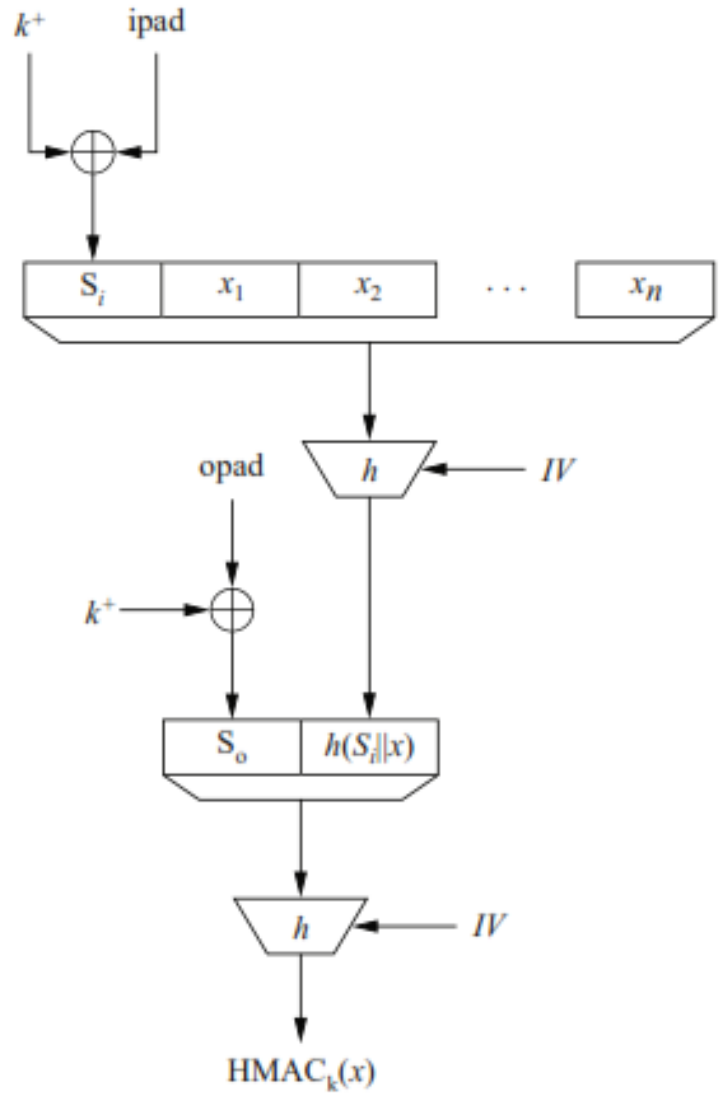
The problem is that the MAC for the message $x = (x_1, x_2, \ldots, x_n, x_{n+1})$, where x_{n+1} is an arbitrary additional block, can be constructed from m without knowing the secret key. The attack is shown in the protocol below.

Attack Against Secret Prefix MACs



HMAC

- A hash-based message authentication code which does not show the security weakness described above is the HMAC construction proposed by Mihir Bellare, Ran Canetti and Hugo Krawczyk in 1996.


k^+
ipad
S_i
x_1
x_2
. . .
x_n
opad
h
IV
k^+
S_o
$h(S_i||x)$
h
IV
$HMAC_k(x)$

- The MAC computation starts with expanding the symmetric key *k with zeros on* the left such that the result *k*+ is *b bits in length, where b is the input block width of* the hash function.
- The expanded key is XORed with the inner pad, which consists the repetition of the bit pattern:
- ipad = 00110110, 00110110,...,00110110
- so that a length of *b bit is achieved. The output of the XOR forms the first input* block to the hash function. The subsequent input blocks are the message blocks (*x1, x2, …xn)*.

The second, outer hash is computed with the padded key together with the output of the first hash. Here, the key is again expanded with zeros and then XORed with the outer pad:

$$\text{opad} = 0101\,1100, 0101\,1100, \ldots, 0101\,1100.$$

The result of the XOR operation forms the first input block for the outer hash. The other input is the output of the inner hash. After the outer hash has been computed, its output is the message authentication code of x. The HMAC construction can be expressed as:

$$\text{HMAC}_k(x) = h\left[(k^+ \oplus \text{opad}) || h\left[(k^+ \oplus \text{ipad}) || x\right]\right].$$

The hash output length l is in practice longer than the width b of an input block. For instance, SHA-1 has an $l = 160$ bit output but accepts $b = 512$ bit inputs. It does not pose a problem that the inner hash function output does not match the input size of outer hash because hash functions have preprocessing steps to match the input string to the block width. As an example, Section 11.4.1 described the preprocessing for SHA-1.

In terms of computational efficiency, it should be noted that the message x, which can be very long, is only hashed once in the inner hash function. The outer hash consists of merely two blocks, namely the padded key and the inner hash output. Thus, the computational overhead introduced through the HMAC construction is very low.