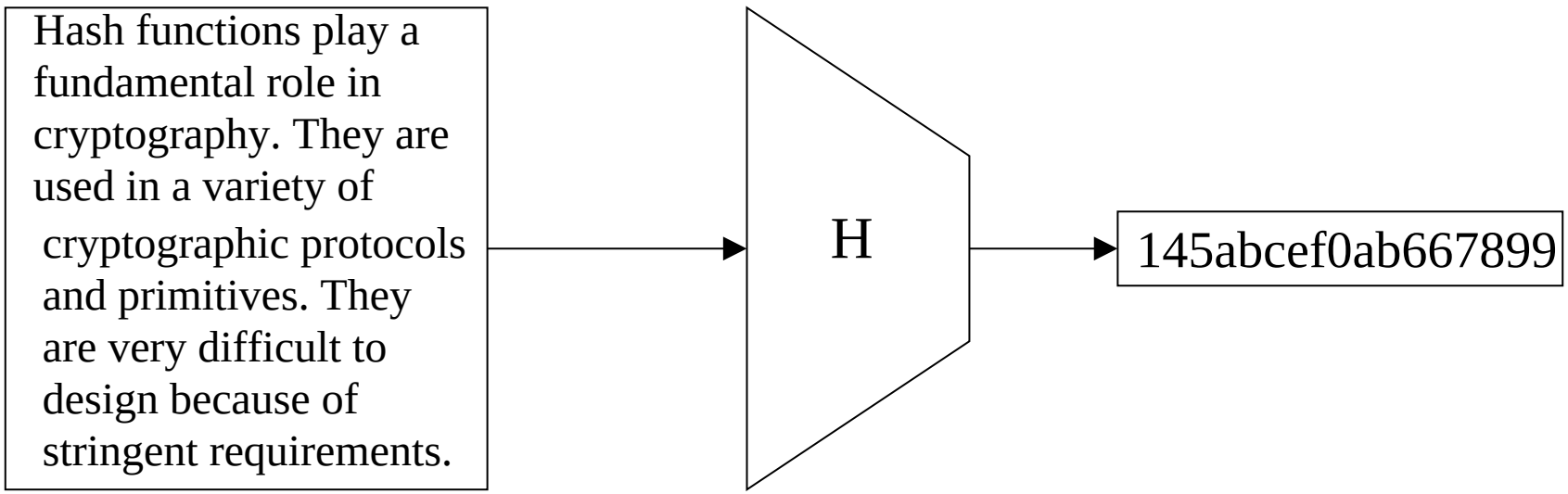
Part IV

Hash functions

Definitions and terminology

- Hash functions play a fundamental role in cryptography.
- They are used in a variety of cryptographic primitives and protocols.
- They are very difficult to design because of very stringent security and performance requirements.
- Examples: SHA-1, RIPEMD-160, SHA-224, SHA-256, SHA-384, SHA-512, BLAKE, SHA-3.

What is a hash function?



Definition of a hash function

- A hash function is a mapping H such that:
  - (i) H maps inputs of arbitrary lengths to outputs of a fixed length n: $H : \{0,1\}^* \longrightarrow \{0,1\}^n$.
    (More generally, H maps elements of a set S to a set T where $|S| > |T|$.)
  - (ii) $H(x)$ can be efficiently computed for all $x \in \{0,1\}^*$.
- H is called an n-bit hash function.
- $H(x)$ is called the hash value, hash, or message digest of x.

- Note: The description of a hash function is public. There are no secret keys.

Typical cryptographic requirements

- Preimage resistance: Given a hash value $y \in_R \{0,1\}^n$, it is computationally infeasible to find (with non-negligible probability of success) any input x such that $H(x) = y$.
  - x is called a preimage of y.
  - $y \in_R \{0,1\}^n$ means that y is chosen uniformly at random from $\{0,1\}^n$.
- 2nd preimage resistance: Given an input $x \in_R \{0,1\}^*$, it is computationally infeasible to find (with non-negligible probability of success) a second input $x' \neq x$ such that $H(x) = H(x')$.
- Collision resistance: It is computationally infeasible to find (with non-negligible probability of success) two distinct inputs x, x' such that $H(x) = H(x')$.
  - The pair (x, x') is called a collision for H.

Generic attacks

- A generic attack on hash functions $H : \{0,1\}^* \to \{0,1\}^n$ does not exploit any properties a specific hash function may have.
- In the analysis of a generic attack, we view H as a random function in the sense that for each $x \in \{0,1\}^*$, the value $y = H(x)$ was chosen by selecting y uniformly at random from $\{0,1\}^n$ (written $y \in_R \{0,1\}^n$).
- From a security point of view, a random function is an ideal hash function. However, random functions are not suitable for practical applications because they cannot be compactly stored.

Generic attack for finding preimages

- Given $y \in \{0,1\}^n$, select arbitrary $x \in \{0,1\}^*$ until $H(x) = y$.
- Expected number of steps is $\approx 2^n$.
  (Here, a step is a hash function evaluation.)
- This attack is infeasible if $n \geq 128$.

Note: It has been proven that this generic attack for finding preimages is optimal, i.e., no better *generic* attack exists.

Generic attack for finding collisions

- Select arbitrary $x \in \{0,1\}^*$ and store $(H(x), x)$ in a table sorted by first entry. Continue until a collision is found.
- Expected number of steps: $\sqrt{\pi 2^n/2} \approx \sqrt{2^n}$ (by birthday paradox). (Here, a step is a hash function evaluation.)
- It has been proven that this generic attack for finding collisions is optimal in terms of the number of hash function evaluations.
- Expected space required: $\sqrt{\pi 2^n/2} \approx \sqrt{2^n}$.
- This attack is infeasible if $n \geq 256$.
- If $n = 128$, the expected running time is about 2^{64} steps.

Some applications of hash functions

1. Password protection on a multi-user computer system:
   - Server stores (userid, H(password)) in a password file. Thus, if an attacker gets a copy of the password file, she does not learn any passwords.
   - Requires preimage-resistance.

2. Modification Detection Codes (MDCs).
   - To ensure that a message m is not modified by unauthorized means, one computes $H(m)$ and protects $H(m)$ from unauthorized modification.
   - e.g. virus protection.
   - Requires 2nd preimage resistance.

Some applications of hash functions

3. Message digests for digital signature schemes:
   - For reasons of efficiency, instead of signing a (long) message, the (much shorter) message digest is signed.
   - Requires preimage-resistance, 2nd preimage resistance and collision resistance. (More on this later)
   - To see why collision resistance is required:
     - Suppose that Alice can find two messages x_1 and x_2, with $x_1 \neq x_2$ and $H(x_1) = H(x_2)$.
     - Alice can sign x_1 and later claim to have signed x_2.

4. Message Authentication Codes (MACs).
   - Provides data integrity & data origin authentication.

Some applications of hash functions

5. Pseudorandom bit generation:
   - Distilling random bits s from several "random" sources $x_1, x_2, \ldots, x_t$.
   - Output $s = H(x_1, x_2, \ldots, x_t)$.

6. Key derivation function (KDF): Deriving a cryptographic key from a shared secret.

Notes:

- Collision resistance is not always necessary.
- Depending on the application, other properties may be needed, for example 'near-collision resistance', 'partial preimage resistance', ...

Examples

The following hash functions are widely used:

- MD5 (RSA Laboratories; 1991) [128 bits]
- SHA-1 (NIST/NSA; 1995) [160 bits]
- SHA-2 (NIST/NSA; 2001) [various bit lengths as indicated]
  - SHA-224
  - SHA-256
  - SHA-384
  - SHA-512
- SHA-3 (Bertoni, Daemen, Peeters, Van Assche; 2012) [various bit lengths]
  - SHA3-224
  - SHA3-256
  - SHA3-384
  - SHA3-512

Less widely used hash functions: MD2, MD4, RIPEMD160, Tiger

The following are NOT hash functions: "Hash tables", CRC8, CRC16, CRC32, ...

MDx hash function family

MD4 and MD5 were designed by Ron Rivest of RSA Laboratories, and share common design principles.

- "MD" = "Message Digest"
- MD5 was a "patch" on MD4 after MD4's security was questioned.

SHA-1 was designed by the NSA, and is clearly inspired by MDx.

- "SHA" = "Secure Hash Algorithm"
- Also known as SHS (Secure Hash Standard)
- SHA-1 was a "patch" on SHA (now called SHA-0).

SHA-2 was designed by the NSA, and has much in common with SHA-1/MDx, but is more complicated.

Non-generic attacks

Non-generic attacks are attacks which exploit a specific function.

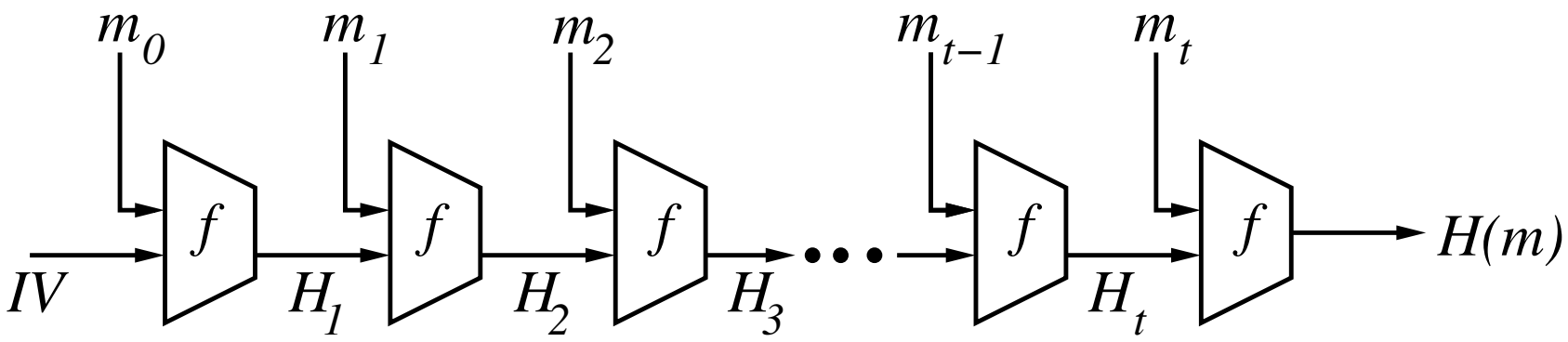
- MD4 (RSA Laboratories):
  - Collisions found in 2^{15} steps (Dobbertin, 1996)
  - ... found in 2^{3} steps (Wang et al., 2005)
  - ... found in 2^{0} steps (Sasaki et al., 2009)
- MD5 (RSA Laboratories):
  - Collisions found in 2^{39} steps (Wang and Yu, 2004)
  - Preimages (theoretical) in $2^{123.4}$ steps (Sasaki and Aoki, 2009)
- SHA (NIST/NSA, based on MD4):
  - Collisions (theoretical) in 2^{61} steps (Chabaud and Joux, 1998)
  - ... found in 2^{51} steps (Joux et al., 2004)
  - ... found in 2^{40} steps (Wang et al., 2004)
- SHA-1 (NIST/NSA, based on SHA):
  - Collisions (theoretical) in 2^{63} steps (Wang et al., 2005)
  - ... (theoretical) in 2^{61} steps (Stevens, 2010)
  - ... found in 2^{63} steps (`https://shattered.io`, 2017)

Iterated Hash Functions (Merkle's Meta-Method)

MDx and SHA-x use the following iterated design:

- Components:
  - Fixed initializing value $IV \in \{0,1\}^n$.
  - Compression function $f : \{0,1\}^{n+r} \longrightarrow \{0,1\}^n$.
- To compute $H(x)$ where x has bitlength $b < 2^r$ do:
  - Break up x into r-bit blocks: $\overline{x} = x_1, x_2, \ldots, x_t$, padding out the last block with 0 bits if necessary.
  - Define x_{t+1}, the length-block, to hold the right-justified binary representation of b.
  - Define $H_0 = IV$.
  - Compute $H_i = f(H_{i-1}, x_i)$ for $i = 1, 2, \ldots, t+1$.
  - H_i's are called chaining variables.
  - Define $H(x) = H_{t+1}$.

Collision Resistance of Iterated Hash Functions



- Theorem (Merkle): If the compression function f is collision resistant, then the function H is also collision resistant.
- Merkle's theorem reduces the problem of finding collision-resistant hash functions to that of finding collision-resistant compression functions.
- However, if a collision is found in f, the consequences are devastating — anything can be appended to a collision to yield another collision.

General structure of MDx hash functions

Hash	Number of rounds in f
MD4	48
MD5	64
SHA-1	80
SHA-256	64
SHA-512	80

Note: Do not confuse the number of rounds with t, which is simply the number of 512-bit message blocks.

- A single 512-bit input block is divided into 32-bit words (1024-bit block and 64-bit words for SHA-512).
- The sixteen 32-bit words are used in the first 16 rounds of f.
- Subsequent rounds of f use either some combination of the original 16 words (MD4, MD5), or words derived from the original sixteen (SHA-1, SHA-2) via a *message schedule*.
- CAUTION: Endian-ness of words and padding differs between MDx and SHA-x.

MD5

- Designed by R. Rivest (1991)
- Based on MD4 with improvements
- Very widely deployed (UNIX `md5sum`)
- Collisions found in 2^{39} steps (Wang and Yu, 2005)
- Still OK for preimage resistance ...

MD5 collision example (Wang et al., 2005)

M_0 :

```
d131dd02 c5e6eec4 693d9a06 98aff95c 2fcab587 12467eab 4004583e b8fb7f89
55ad3406 09f4b302 83e48883 2571415a 085125e8 f7cdc99f d91dbdf2 80373c5b
d8823e31 56348f5b ae6dacd4 36c919c6 dd53e2b4 87da03fd 02396306 d248cda0
e99f3342 0f577ee8 ce54b670 80a80d1e c69821bc b6a88393 96f9652b 6ff72a70
```

M_1 :

```
d131dd02 c5e6eec4 693d9a06 98aff95c 2fcab507 12467eab 4004583e b8fb7f89
55ad3406 09f4b302 83e48883 25f1415a 085125e8 f7cdc99f d91dbd72 80373c5b
d8823e31 56348f5b ae6dacd4 36c919c6 dd53e234 87da03fd 02396306 d248cda0
e99f3342 0f577ee8 ce54b670 8028 0d1e c69821bc b6a88393 96f965ab 6ff72a70
```

SHA-1

- Designed by NSA (1994)
- Based on SHA-0 (1993) which was based on MD4/MD5
- Theoretical collisions in 2^{63} steps (Wang et al., 2005)
- Collisions found in 2^{63} steps (`https://shattered.io`, 2017)
- Still OK for preimage resistance ...

SHA-1 collision (https://shattered.io)

CV_0	4e	a9	62	69	7c	87	6e	26	74	d1	07	f0	fe	c6	79	84	14	f5	bf	45
$M_1^{(1)}$			7f	46	dc	93	a6	b6	7e	01	3b	02	9a	aa	1d	b2	56	0b		
			45	ca	67	d6	88	c7	f8	4b	8c	4c	79	1f	e0	2b	3d	f6		
			14	f8	6d	b1	69	09	01	c5	6b	45	c1	53	0a	fe	df	b7		
			60	38	e9	72	72	2f	e7	ad	72	8f	0e	49	04	e0	46	c2		
$CV_1^{(1)}$	8d	64	d6	17	ff	ed	53	52	eb	c8	59	15	5e	c7	eb	34	f3	8a	5a	7b
$M_2^{(1)}$			30	57	0f	e9	d4	13	98	ab	e1	2e	f5	bc	94	2b	e3	35		
			42	a4	80	2d	98	b5	d7	0f	2a	33	2e	c3	7f	ac	35	14		
			e7	4d	dc	0f	2c	c1	a8	74	cd	0c	78	30	5a	21	56	64		
			61	30	97	89	60	6b	d0	bf	3f	98	cd	a8	04	46	29	a1		
CV_2	1e	ac	b2	5e	d5	97	0d	10	f1	73	69	63	57	71	bc	3a	17	b4	8a	c5

CV_0	4e	a9	62	69	7c	87	6e	26	74	d1	07	f0	fe	c6	79	84	14	f5	bf	45
$M_1^{(2)}$			73	46	dc	91	66	b6	7e	11	8f	02	9a	b6	21	b2	56	0f		
			f9	ca	67	cc	a8	c7	f8	5b	a8	4c	79	03	0c	2b	3d	e2		
			18	f8	6d	b3	a9	09	01	d5	df	45	c1	4f	26	fe	df	b3		
			dc	38	e9	6a	c2	2f	e7	bd	72	8f	0e	45	bc	e0	46	d2		
$CV_1^{(2)}$	8d	64	c8	21	ff	ed	52	e2	eb	c8	59	15	5e	c7	eb	36	73	8a	5a	7b
$M_2^{(2)}$			3c	57	0f	eb	14	13	98	bb	55	2e	f5	a0	a8	2b	e3	31		
			fe	a4	80	37	b8	b5	d7	1f	0e	33	2e	df	93	ac	35	00		
			eb	4d	dc	0d	ec	c1	a8	64	79	0c	78	2c	76	21	56	60		
			dd	30	97	91	d0	6b	d0	af	3f	98	cd	a4	bc	46	29	b1		
CV_2	1e	ac	b2	5e	d5	97	0d	10	f1	73	69	63	57	71	bc	3a	17	b4	8a	c5

SHA-2

- Designed by NSA (2001)
- Longer outputs, slower and more complicated processing
- Choice of output lengths: 224 bits (SHA-224), 256 bits (SHA-256), 384 bits (SHA-384) and 512 bits (SHA-512)
- Standardized in FIPS 180-2
- No weaknesses known

SHA-3

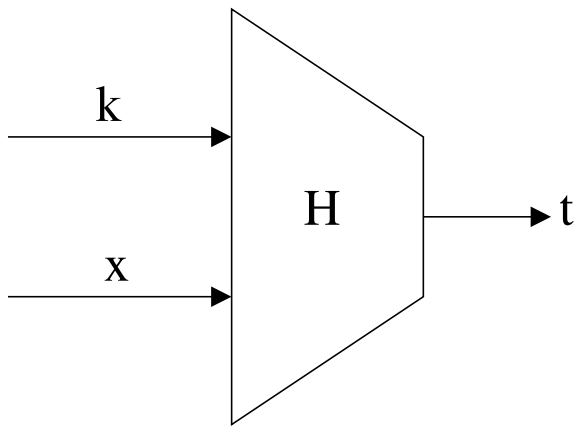
- SHA-2 is similar to SHA-1, and thus there are concerns that the SHA-1 weaknesses will extend to SHA-2.
- SHA-3: NIST hash function competition.
  - 64 candidates submitted by Oct 31 2008 deadline.
  - 51 were accepted for the first round.
  - (July 2009) 14 were selected for the second round.
  - Encourage the public to study the hash functions.
  - Third quarter 2010: Select a list of finalists.
    (BLAKE, Grøstl, JH, Keccak, Skein)
  - Encourage the public to study the finalists.
  - October 2, 2012: Keccak selected as SHA-3

Part V

Message authentication codes

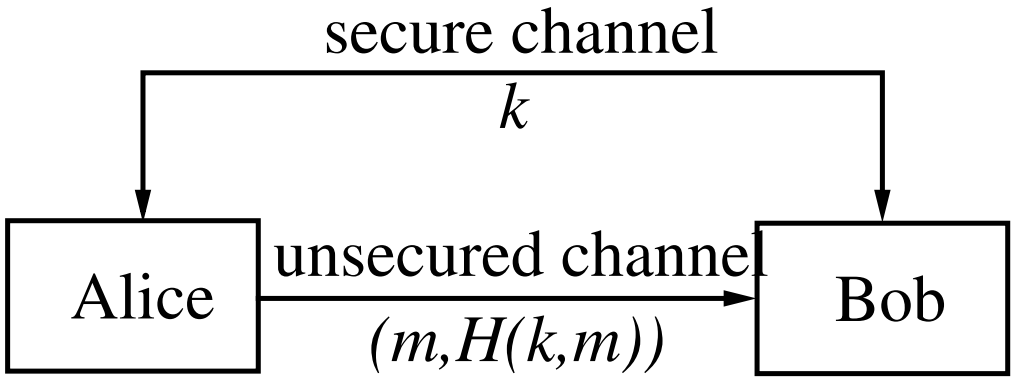
Definition

- A message authentication code (MAC) scheme is a family of functions $H_k : \{0,1\}^* \longrightarrow \{0,1\}^n$ parameterized by an ℓ-bit key k, where each function H_k can be efficiently computed.
- $H_k(x)$ is called the MAC or tag of x.



- MAC schemes are used for providing (symmetric-key) data integrity and data origin authentication.

Applications of MAC Schemes



- To provide data integrity and data origin authentication:
  1. Alice and Bob establish a secret key $k \in \{0,1\}^{\ell}$.
  2. Alice computes $t = H_k(x)$ and sends (x, t) to Bob.
  3. Bob verifies that $t = H_k(x)$.
- Note: No confidentiality or non-repudiation.
- To avoid replay, add a timestamp, or sequence number.
- Widely used in banking applications.

Security Definition

- Let k be the secret key shared by Alice and Bob.
- The adversary does not know k, but is allowed to obtain (from Alice or Bob) tags for messages of her choosing. The adversary's goal is to obtain the tag of any message whose tag she did not already obtain from Alice or Bob.
- Definition: A MAC scheme is secure if given some MAC tags $H_k(x_i)$ for x_i's of one's own choosing, it is computationally infeasible to compute (with non-negligible probability of success) a pair $(x, H_k(x))$ for any new message x.
  - That is, the MAC scheme must be existentially unforgeable against chosen-message attack.
- Note: A secure MAC scheme can be used to provide data integrity and data origin authentication.

Generic Attacks

Guessing the MAC of a message x:

- Select $y \in \{0,1\}^n$ and guess that $H_k(x) = y$.
- Assuming that H_k is a random function, the probability of success is $1/2^n$.
- Note: Guesses cannot be directly checked.
- Depending on the application where the MAC algorithm is employed, one could choose n as small as 32 (say). In general, $n \geq 80$ is preferred.

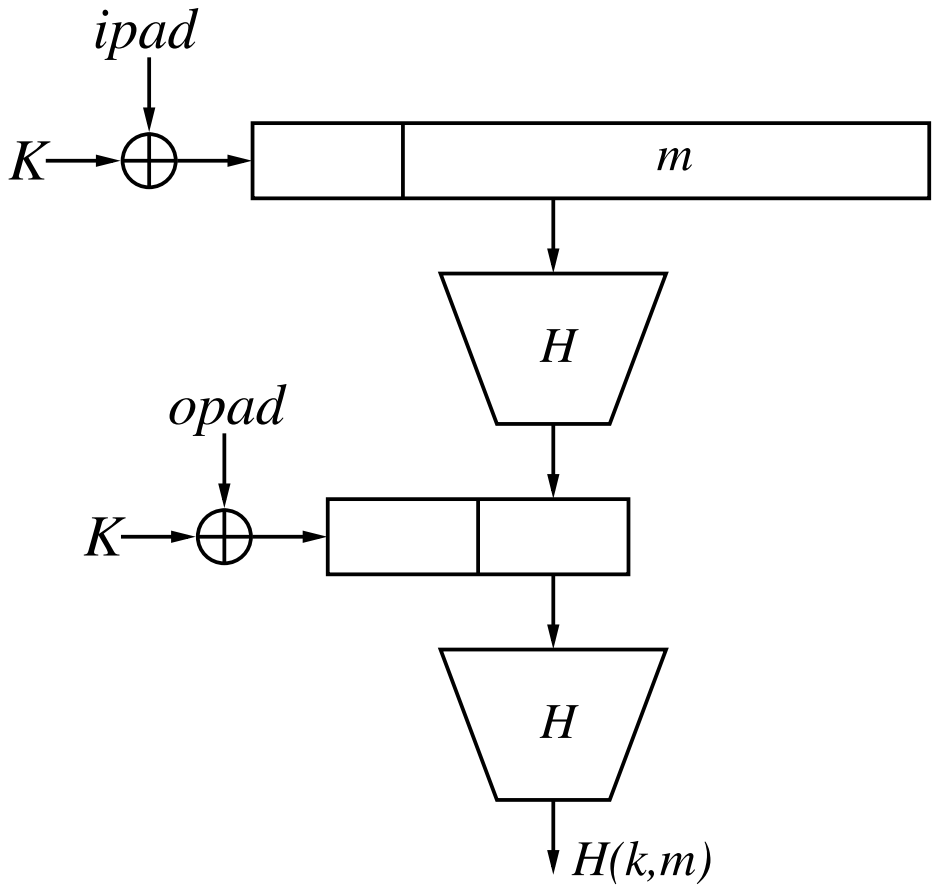
Generic Attacks

Exhaustive search on the key space:

- Given r known message-MAC pairs: $(x_1, t_1), \ldots, (x_r, t_r)$, one can check whether a guess k of the key is correct by verifying that $H_k(x_i) = t_i$, for $i = 1, 2, \ldots, r$.
- Assuming that the H_k's are random functions, the expected number of keys for which the tags verify is $\mathrm{FK} = (2^\ell - 1)/2^{nr}$.
  - Example: If $\ell = 56$, $n = 64$, $r = 2$, then $\mathrm{FK} \approx 1/2^{72}$.
- Expected number of s is $\approx 2^\ell$.
- Exhaustive search is infeasible if $\ell \geq 80$.

HMAC

- "Hash-based" MAC; Bellare, Canetti & Krawczyk (1996).
- Define 2 r-bit strings (in hexadecimal notation): ipad = 0x36, opad = 0x5C; each repeated $r/8$ times.



- MAC definition: $H_K(x) = H(K \oplus \text{opad}, H(K \oplus \text{ipad}, x))$.

HMAC

- Proved secure:
  Theorem: Suppose that the compression function used in H is a secure MAC with fixed length messages and a secret IV as the key. Then HMAC is a secure MAC algorithm.
- HMAC is specified in IETF RFC 2104 and FIPS 198.
- SHA-3 is designed to be safe to use directly without HMAC: $H_K(x) = H(K, x)$. Other hash functions are **unsafe** to use directly.

Combining encryption and authentication

- In many situations, one wishes to achieve both data confidentiality and data authentication.
- Given a shared secret key:
  - A symmetric-key encryption scheme achieves confidentiality.
  - A message authentication code achieves data authentication.
- Combining the two is trivial. Right? Well, not really...

"people had been doing rather poorly when they tried to glue together a traditional (privacy-only) encryption scheme and a message authentication code (MAC)"

—*M. Bellare, P. Rogaway, and D. Wagner*

"it is very easy to accidentally combine secure encryption schemes with secure MACs and still get insecure authenticated encryption schemes"

—*T. Kohno, J. Viega, and D. Whiting*

Possible strategies

Let m be a message. There are (at least) three obvious ways to combine encryption and authentication:

- MAC-then-encrypt (MtE):

  $$\text{compute } t = \text{MAC}(m) \text{ and } c = \text{Enc}(m||t), \text{ transmit } c$$

- encrypt-then-MAC (EtM):

  $$\text{compute } c = \text{Enc}(m) \text{ and } t = \text{MAC}(c), \text{ transmit } c||t$$

- encrypt-and-MAC (E&M):

  $$\text{compute } c = \text{Enc}(m) \text{ and } t = \text{MAC}(m), \text{ transmit } c||t$$

These constructions are called *composed primitives* because they combine two primitives into one.

Security goals

Integrity of plaintext (INT-PTXT):

- Produce a valid ciphertext (i.e. one that passes all authentication checks) containing an encryption of a plaintext message that was never sent by the legitimate sender.

Integrity of ciphertext (INT-CTXT):

- Produce a valid ciphertext (i.e. one that passes all authentication checks) that was never sent by the legitimate sender.

Examples

- MAC-then-encrypt: SSL/TLS
- encrypt-then-MAC: IPsec
- encrypt-and-MAC: SSH

Encrypt and MAC (E&M)

Consider encrypt-and-MAC:

$$\text{compute } c = \text{Enc}(m) \text{ and } t = \text{MAC}(m), \text{ transmit } c||t$$

When decrypting, the recipient checks that the MAC is correct.

- Problem: MACs are not required to ensure confidentiality.
  - For example, the MAC might leak one plaintext bit, and still be "secure" as a MAC. Violates semantic security!
- Even if the SKES is secure and the MAC is secure, the encrypt-and-MAC combination might be insecure.

MAC then encrypt (MtE)

Consider MAC-then-encrypt:

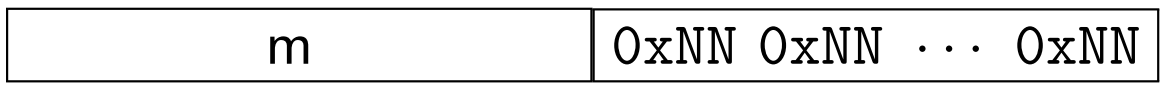
$$\text{compute } t = \text{MAC}(m) \text{ and } c = \text{Enc}(m||t), \text{ transmit } c$$

When decrypting, the recipient checks that the MAC is correct.

- Problem: SKESs are not required to ensure integrity.
  - For example, changing the ciphertext might not change the plaintext for certain values of plaintext. Violates INT-CTXT!
  - One can often then also learn information about the plaintext.
- Even if the SKES is secure and the MAC is secure, the MAC-then-encrypt combination might be insecure.

An extra complication: padding

- Often, when using block ciphers, a message needs to be padded before encrypting, to align it with block boundaries.
- Example — PKCS #7:



  where `NN` is the number of bytes of padding (in hexadecimal).
- Should the padding be included in the MAC input?

Padding example

Suppose we are designing SSL/TLS using MAC-then-encrypt.

Option 1. Apply MAC, then pad, then encrypt:

compute $t = \text{MAC}(m)$ and $c = \text{Encrypt}(m||p||t)$, transmit c

Option 2. Apply padding, then MAC, then encrypt:

compute $t = \text{MAC}(m||p)$ and $c = \text{Encrypt}(m||p||t)$, transmit c

SSL/TLS uses Option 1. Unfortunately, this is the wrong choice.

Padding oracle attacks

- SSL/TLS:
  - BEAST (2011)
  - POODLE (2014)
- SSH:
  - Plaintext recovery attack (2008)

Encrypt then MAC (EtM)

The good news: encrypt-then-MAC is safe!

Definition (Ciphertext unforgeability)

An encryption scheme is ciphertext unforgeable under chosen plaintext attack (CUF-CPA) if an attacker using the following interactions cannot achieve the following goal:

- Interaction: The attacker may choose any number of messages m_i and obtain their corresponding valid (authenticated) encryptions c_i under the key k.
- Computation: The attacker is computationally bounded.
- Goal: The attacker must produce a valid (authenticated) encryption $c \neq c_i$ (i.e. break INT-CTXT).

Theorem (Canetti & Krawczyk, 2001)

Suppose the SKES is semantically secure under chosen plaintext attack, and the MAC is EUF-CMA. Then encrypt-then-MAC is semantically secure and CUF-CPA.

Ferguson & Schneier, 1999:

> *When both encryption and authentication are provided, IPsec performs the encryption first, followed by the authentication. In our opinion, this is the wrong order. Going by the "Horton principle", the protocol should authenticate what was meant, not what was said. Authentication should thus be applied to the plaintext, and not to the ciphertext.*

WRONG!

The Cryptographic Doom Principle (Moxie Marlinspike, 2011):

> *If you have to perform any cryptographic operation before verifying the MAC on a message you've received, it will* **somehow** *inevitably lead to doom.*