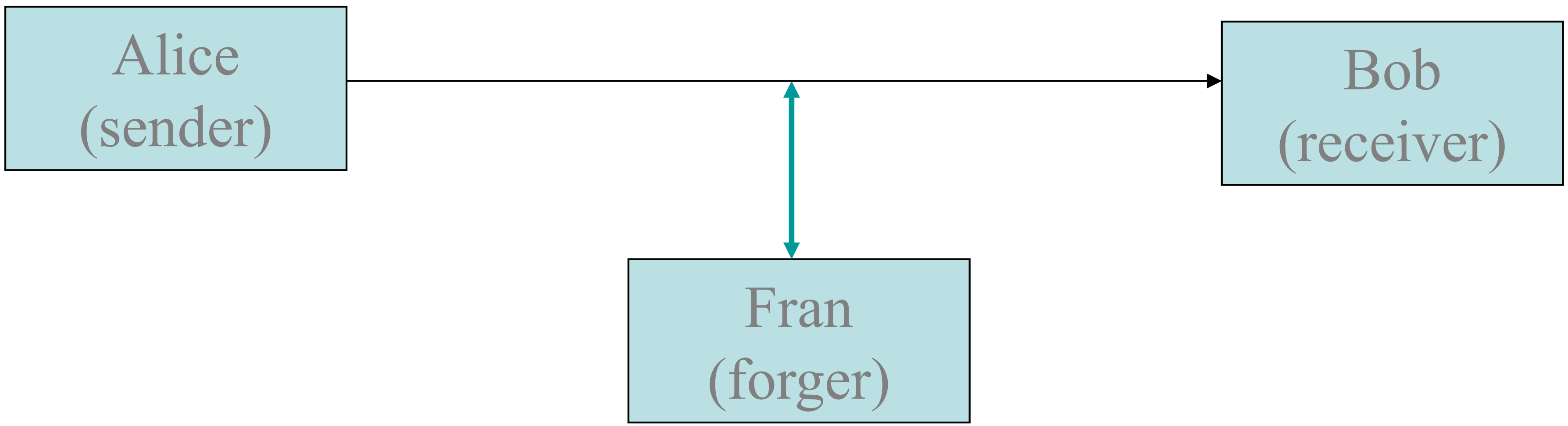
Data Integrity & Authentication

Message Authentication Codes (MACs)

Goal

Ensure integrity of messages, even in presence of an active adversary who sends own messages.



Remark: Authentication is orthogonal to secrecy, yet systems often required to provide both.

Definitions

- Authentication algorithm - A
- Verification algorithm - V ("accept"/"reject")
- Authentication key – k
- Message space (usually binary strings)
- Every message between Alice and Bob is a pair (m, $A_k(m)$)
- $A_k(m)$ is called the authentication tag of m

Definition (cont.)

- Requirement – $V_k(m, A_k(m))$ = "accept"
  - The authentication algorithm is called *MAC* (Message Authentication Code)
  - $A_k(m)$ is frequently denoted $MAC_k(m)$
  - Verification is by executing authentication on m and comparing with $MAC_k(m)$

Properties of MAC Functions

- Security requirement – adversary can't construct a *new* legal pair $(m, MAC_k(m))$ even after seeing $(m_i, MAC_k(m_i))$ $(i=1, 2, \ldots, n)$
- Output should be as *short* as possible
- The MAC function is *not* 1-to-1

Adversarial Model

- Available Data:
  - The MAC algorithm
  - Known plaintext (pairs (m, $MAC_K(m)$))
  - Chosen plaintext (choose m, get $MAC_K(m)$)
- Note: chosen MAC is unrealistic
- Goal: Given n legal pairs

  $(m_1, MAC_k(m_1)), \ldots, (m_n, MAC_k(m_n))$

  find a new legal pair $(m, MAC_k(m))$

Adversarial Model

- adversary succeeds even if message Fran forged is “meaningless“
- reasons: hard to predict
  - what has meaning and no meaning in an unknown context
  - how will the receiver react to such successful forgery

Efficiency

- Adversary goal: given n legal pairs $(m_1, MAC_k(m_1)), \ldots, (m_n, MAC_k(m_n))$ find a new legal pair $(m, MAC_k(m))$ efficiently and with non negligible probability.

- If n is large enough then n pairs $(m_i, MAC_k(m_i))$ determine the key k uniquely (with high prob.). Thus a non-deterministic machine can guess k and verify it. But doing this deterministically should be computationally hard.

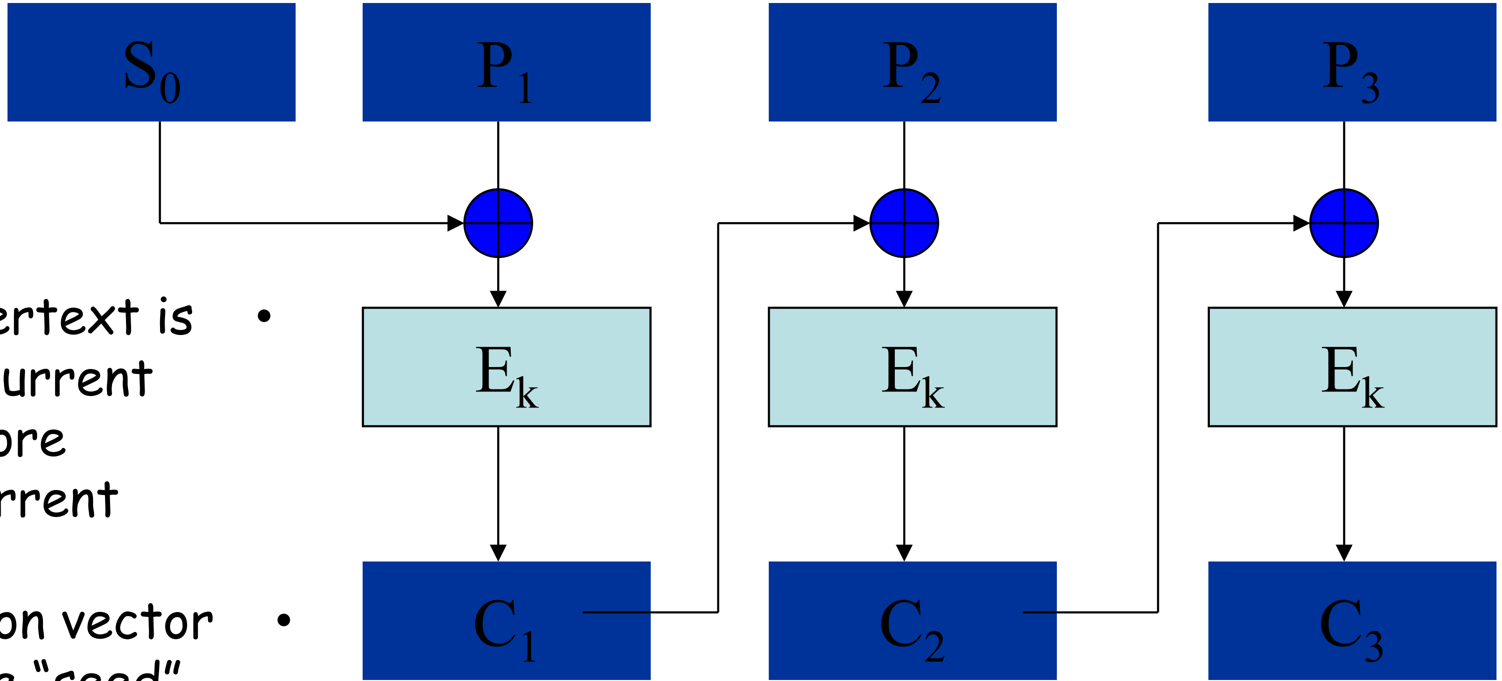
MACs Used in Practice

We describe

- MAC based on CBC Mode Encryption
  - uses a block cipher
  - slow
- MAC based on cryptographic hash functions
  - fast
  - no restriction on export

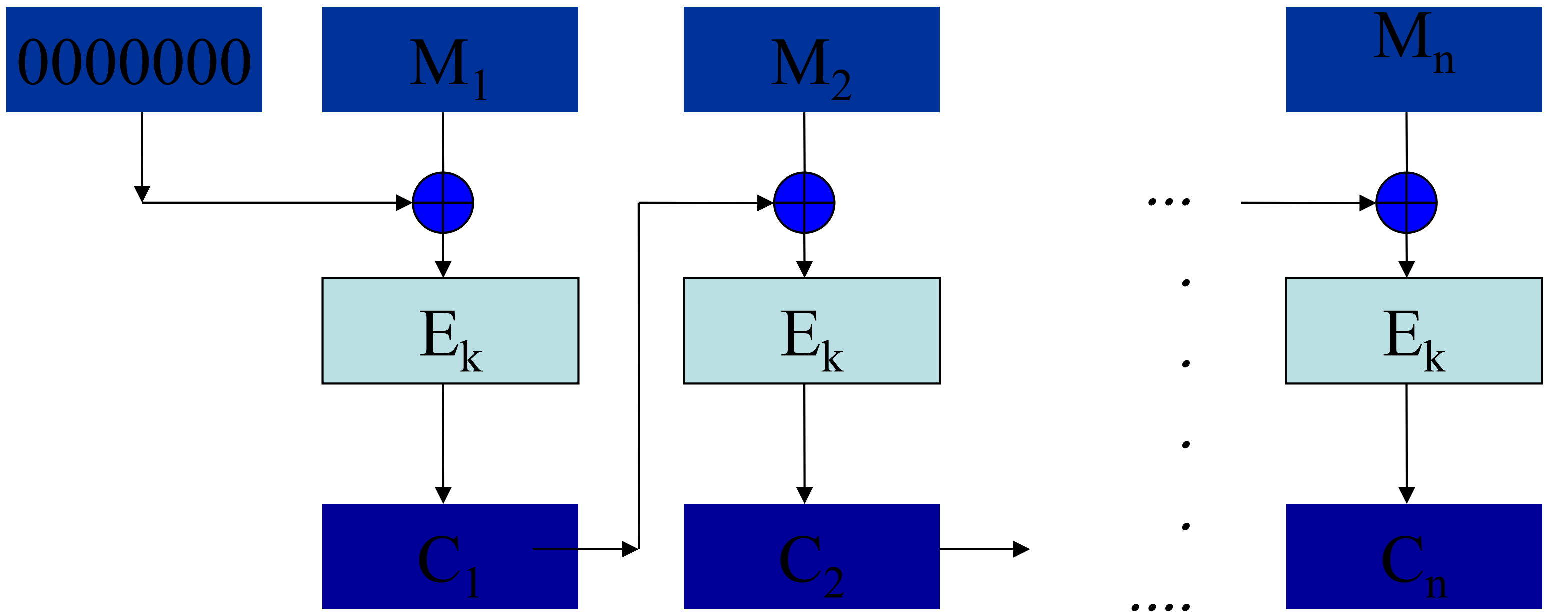
Reminder: CBC Mode Encryption
(Cipher Block Chaining)

- Previous ciphertext is XORed with current plaintext before encrypting current block
- An initialization vector S_0 is used as a "seed" for the process
- Seed can be "openly" transmitted



CBC Mode MACs

- Start with the all zero seed.
- Given a message consisting of n blocks $M_1, M_2, \ldots, M_n$, apply CBC (using the secret key k).



- Produce n "cipertext" blocks $C_1, C_2, \ldots, C_n$, discard first n-1.
- Send $M_1, M_2, \ldots, M_n$ & the authentication tag $MAC_k(M) = C_n$.

Security of CBC MAC [BKR00]

- Pseudo random function: a function that looks random (to any polynomial time alg.)
- Recall: a good encoding scheme transforms the message in an apparently random string

Claim: If E_k is a pseudo random function, then the fixed length CBC MAC is resilient to forgery.

Proof outline: Assume CBC MAC can be forged efficiently. Transform the forging algorithm into an algorithm distinguishing E_k from random function efficiently.

see file BKR2000.pdf in course website

CBC-MAC is insecure for variable-length messages

- CBC-MAC is secure for fixed-length messages, bur insecure for variable-length messages
- if attacker knows correct message-tag pairs (m, t) and (m', t') can generate a third (longer) message m'' whose CBC-MAC will also be t'
  - XOR first block of m' with t and then concatenate m with this modified m'
  - hence, $m'' = m \,||\, (m_1' \oplus t) \,||\, m_2' \,||\, \ldots \,||\, m_x'$

Combined Secrecy & MAC

- Given a message consisting of n blocks $M_1, M_2, \ldots, M_n$, apply CBC (using the secret key k1) to produce $MAC_{k1}(M)$.

- Produce n ciphertext blocks $C_1, C_2, \ldots, C_n$ under a different key, k2.
- Send $C_1, C_2, \ldots, C_n$ & the authentication tag $MAC_{k1}(M)$.

Hash Functions

- Map large domains to smaller ranges
- Example *h:* $\{0,1,\ldots,p^2\} \rightarrow \{0,1,\ldots,p-1\}$ defined by $h(x) = ax+b \bmod p$
- Used extensively for searching (hash tables)
- Collisions are resolved by several possible means – chaining, double hashing, etc.

Hash function and MAC

- Goal: compute MAC of a message using
  - hash function h
  - message m
  - Secret key k
- MAC must be a function of the key and of the message
- Examples $MAC_k(m)=h(k||m)$ or $h(m||k)$ or $h(k||m||k)$
- Also first bits of $h(k||m)$ or $h(m||k)$

Collision Resistance

- A hash function *h: D* $\rightarrow$ *R* is called *weakly collision resistant* for $x \in D$ if it is hard to find $x' \neq x$ such that $h(x')=h(x)$
- A function *h: D*$\rightarrow$*R* is called *strongly collision resistant* if it is hard to find x, x' such that $x' \neq x$ but $h(x)=h(x')$

Note: if you find collision then you might be able to find two messages with the same MAC

strong $\Rightarrow$ weak

- proof: we show ($\neg$weak $\Rightarrow$ $\neg$strong)
- given h, suppose there is poly alg. A_h: $A_h(x) = x'$ s.t. $h(x) = h(x')$
- we construct poly alg. B_h s.t. $B_h() = (x, x')$ s.t. $h(x) = h(x')$:
  - randomly choose x
  - return $(x, A_h(x))$

The Birthday Paradox

- If 23 people are chosen at random the probability that two of them have the same birth-day is greater than 0.5
- More generally, let h: $D \rightarrow R$ be any mapping. If we choose $1.1774|R|^{1/2}$ elements of D at random, the probability that two of them are mapped to the same image is 0.5.
  - the expected number of choices before finding the first collision is approximated by $(\frac{1}{2} \pi |R|)^{\frac{1}{2}}$

Birthday attack

- Given a function f, find two different inputs x_1, x_2 such that $f(x_1) = f(x_2)$.
- E.g., if a 64-bit hash is used, there are approximately 1.8×10^{19} different outputs. If these are all equally probable, then it would take approximately 5.38×10^9 (expected) attempts to generate a collision using brute force (5.1×10^9 is the number that makes probability = 0.5). This value is called **birthday bound**

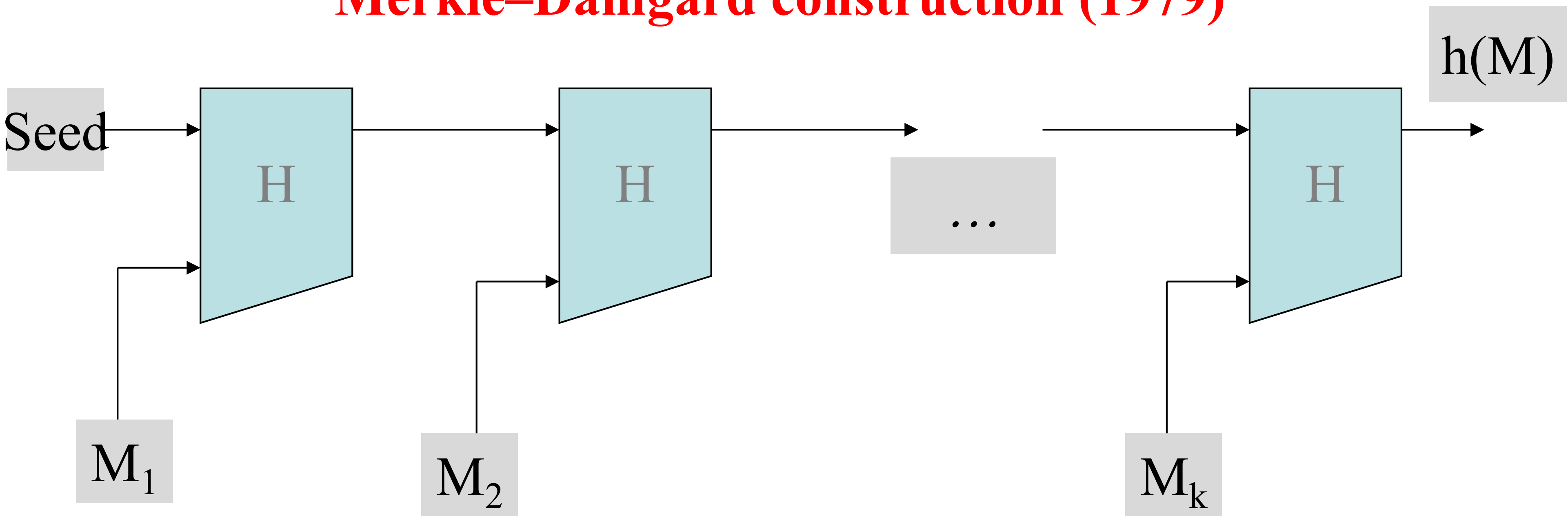
Cryptographic Hash Functions

Cryptographic hash functions are hash functions that are strongly collision resistant.

- Notice: No secret key.
- Should be very fast to compute, yet hard to find colliding pairs (impossible if P=NP).
- Usually defined by:
  - Compression function mapping n bits (e.g. 512) to m bits (e.g 160), $m < n$.

Extending to Longer Strings

Merkle–Damgård construction (1979)



H : D --> R (fixed sets, typically $\{0,1\}^n$ and $\{0,1\}^m$)

Extending the Domain (cont.)

- The seed is usually constant
- Typically, padding (including text length of original message) is used to ensure a multiple of n.
- Claim: if the basic function H is collision resistant, then so is its extension.

Lengths

- Input message length should be arbitrary. In practice it is usually up to 2^{64}, which is good enough for all practical purposes.
- Block length is usually 512 bits.
- **Output length should be at least 160 bits to prevent *birthday attacks*.**

Basing MACs on Hash Functions

- combine message and secret key, hash and produce MAC (*keyed hashing*)
  - $MAC_k(m) = h(k||m)$ KO adv. can add extra bits to message and compute correct MAC (knowledge of k not necessary, it suffices to add extra h blocks)
  - $MAC_k(m) = h(m||k)$ small problem: the adv. can exploit the birthday paradox; in fact assume k is the last block; then if adv. finds two colliding messages then she knows two messages with the same MAC - for all keys
  - $MAC_k(m)= h(k||m||k)$: OK (similar to HMAC)
  - $MAC_k(m)$ = first bits (e.g. first half) of $h(k||m)$ or $h(m||k)$: OK (adversary is not able to check correctness)

Cryptographic Hash Functions

- MD family ("message digest"), MD-4, MD-5: broken
- SHA-0 [1993] SHA-1 [1995] (secure hash standard, 160 bits) (www.itl.nist.gov/fipspubs/fip180-1.htm)
- RIPE-MD, SHA-2 256, 384 and 512 [2001] (proposed standards, longer digests, use same ideas of SHA-1)

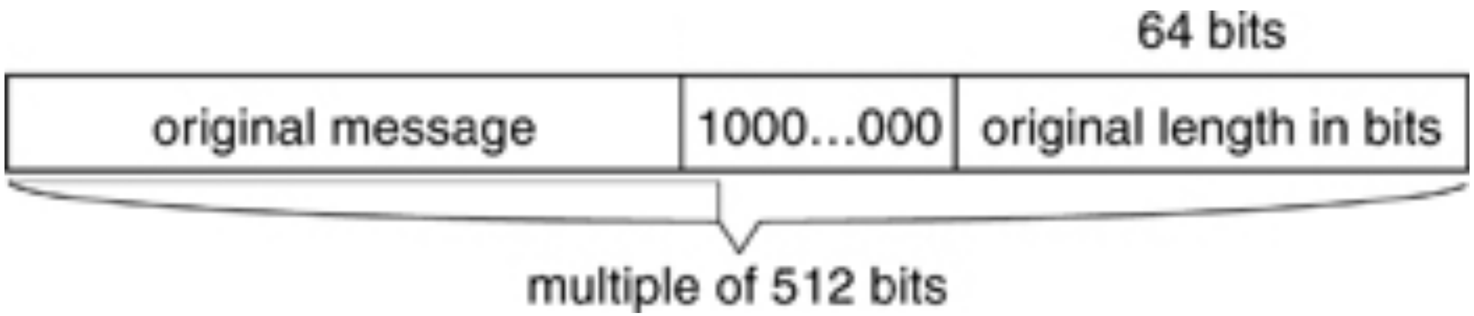
Idea: divide the message in block

- perform a number of rounds (say 80) on each block
- each round mixes changes and shuffles bit of the block
- at the end what you get looks like a random string

- SHA-3 224, 256, 384, 512 [2012]

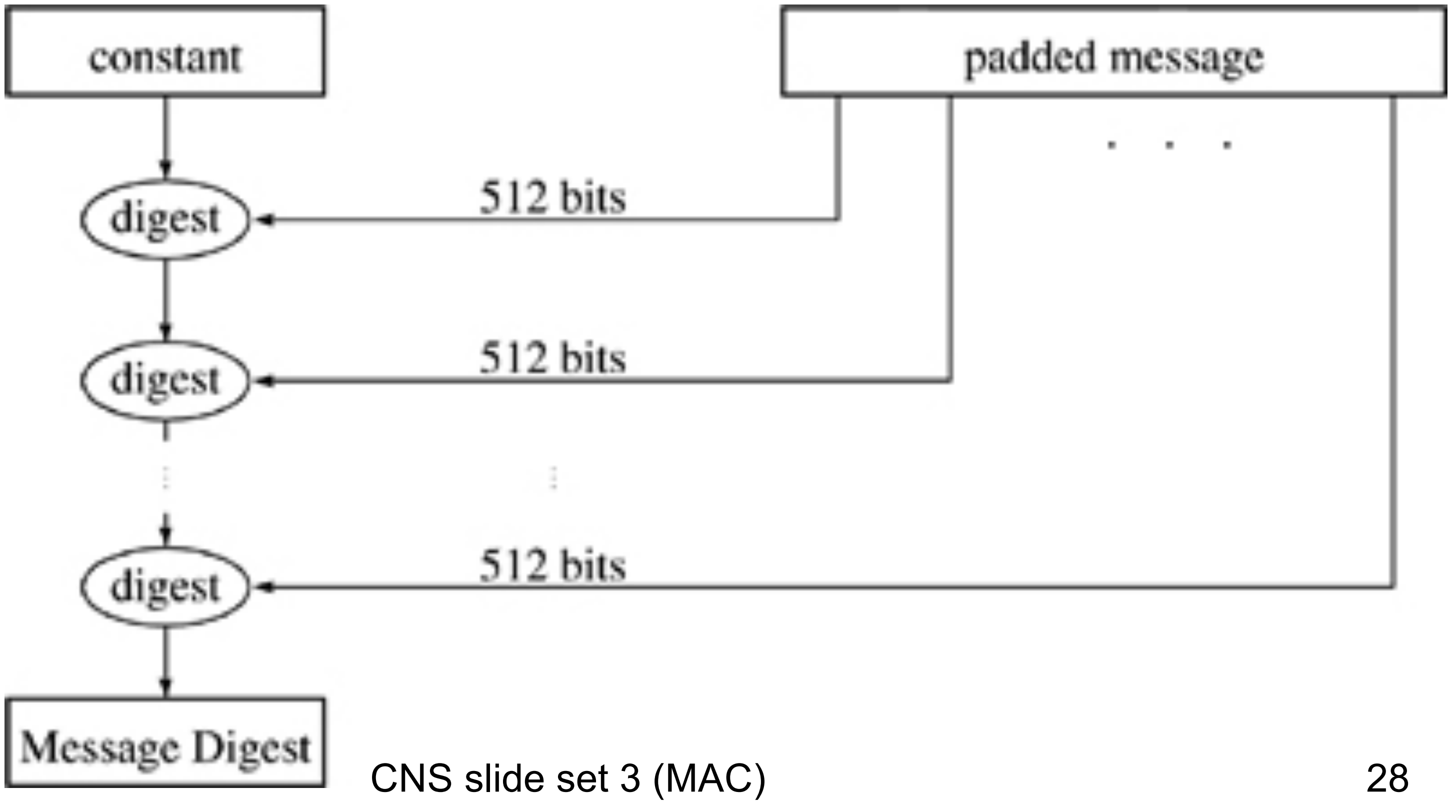
SHA-1 basics

- similar to MD4 & MD5
- |message| <= 2^{64}, |digest| = 160, |block| = 512
- original message is padded



SHA-1 overview

- The 160-bit message digest consists of five 32-bit words: A, B, C, D, and E.
- Before first stage: A = 67452301_{16}, B = $efcdab89_{16}$, C = $98badcfe_{16}$, D = 10325476_{16}, E = $c3d2e1f0_{16}$.
- After last stage A|B|C|D|E is message digest



SHA-1: processing one block

Block (512 bit, 16 words)

- 80 rounds: each round modifies the buffer (A,B,C,D,E)

Round:

$(A,B,C,D,E) \leftarrow$

$(E + f(t, B, C, D) + (A<<5) + W_t + K_t), A, (B<<30), C, D)$

- t number of round, << denotes left shift
- f(t,B,C,D) is a complicate nonlinear function
- W_t is a 32 bit word obtained by expanding original 16 words into 80 words (using shift and ex-or)
- K_t constants

$K_t = \lfloor 2^{30}\sqrt{2} \rfloor = 5a827999_{16}$ $(0 <= t <= 19)$

$K_t = \lfloor 2^{30}\sqrt{3} \rfloor = 6ed9eba1_{16}$ $(20 <= t <= 39)$

$K_t = \lfloor 2^{30}\sqrt{5} \rfloor = 8f1bbcdc_{16}$ $(40 <= t <= 59)$

$K_t = \lfloor 2^{30}\sqrt{10} \rfloor = ca62c1d6_{16}$ $(60 <= t <= 79)$

function f

$f(t, B, C, D) =$

- $(B \wedge C) \vee (\sim B \wedge D)$ $(0 \leq t \leq 19)$
- $B \oplus C \oplus D$ $(20 \leq t \leq 39)$
- $(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$ $(40 \leq t \leq 59)$
- $B \oplus C \oplus D$ $(60 \leq t \leq 79)$

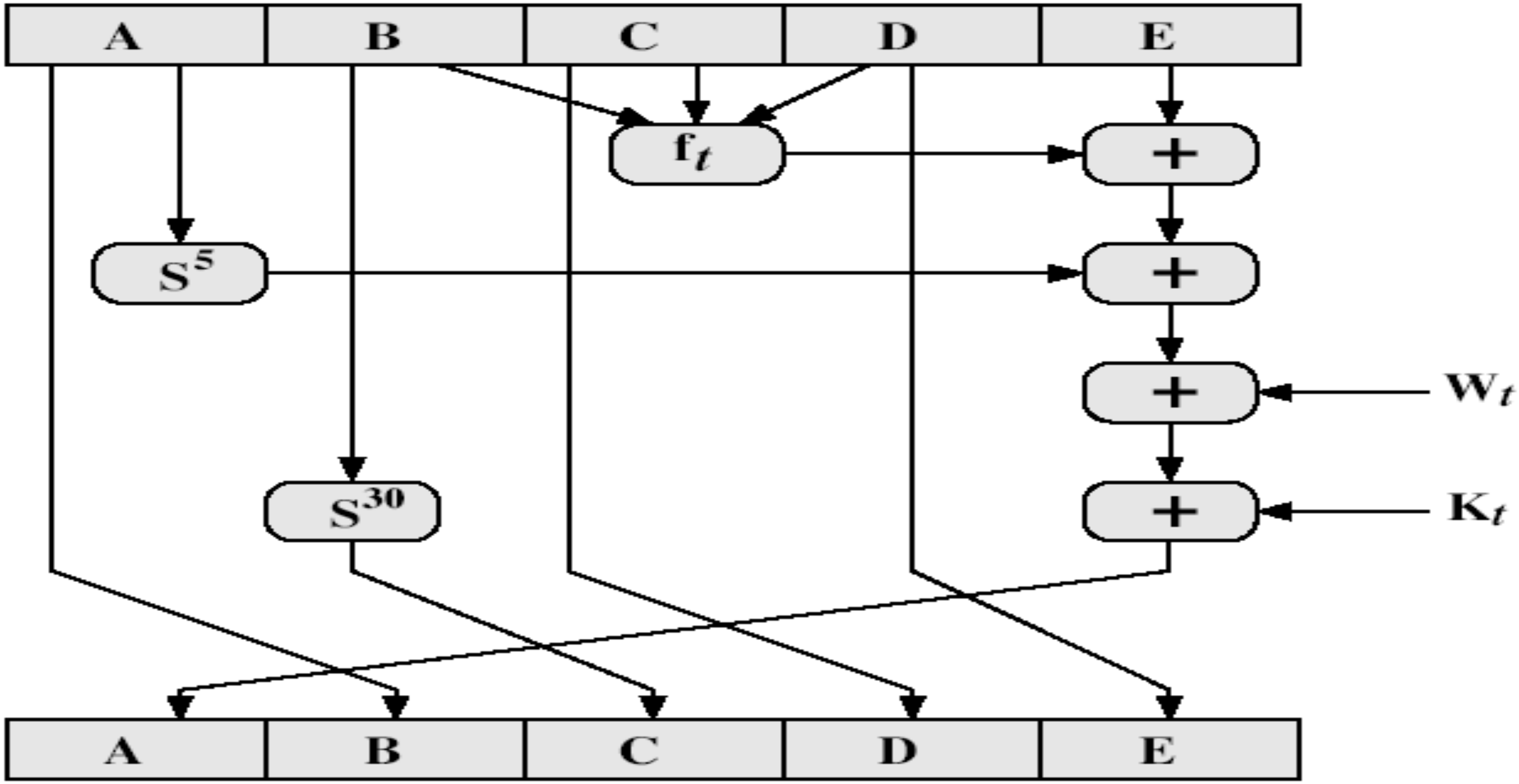
word w_t

$w_0 \dots w_{15}$ are the original 512 bits

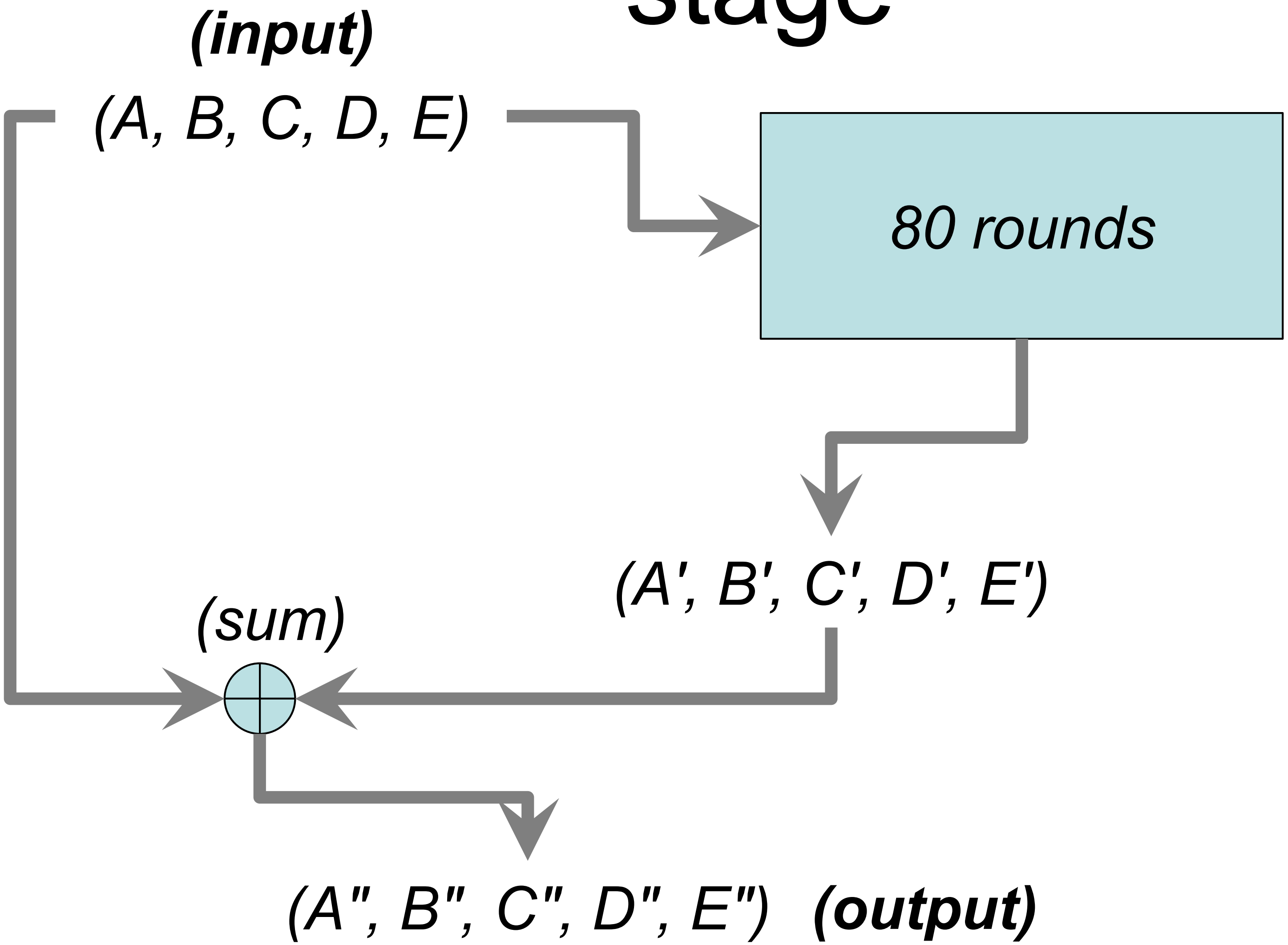
for $16 \le t \le 79$

- $w_t = (w_{t-3} + w_{t-8} + w_{t-14} + w_{t-16}) <<= 1$
- "<<=" denotes left bit rotation

SHA-1: round t



stage



SHA-1 summary

1. Pad initial message: final length must be ≡ 448 mod 512 bits
2. Last 64 bit are used to denote the message length
3. Initialize buffer of 5 words (160-bit) (A,B,C,D,E) (67452301, efcdab89, 98badcfe, 10325476, c3d2e1f0)
4. Process first block of 16 words (512 bits):
   4.1 expand the input block to obtain 80 words block W0,W1,W2,…W79 (ex-or and shift on the given 512 bits)
   4.2 initialize buffer (A,B,C,D,E)
   4.3 update the buffer (A,B,C,D,E): execute 80 rounds
      each round transforms the buffer
   4.4 the final value of buffer (H1 H2 H3 H4 H5) is the result
5. Repeat for following blocks using initial buffer (A+H1, B+H2,…)

Gradually sunsetting SHA-1

Posted: Friday, September 5, 2014

Cross-posted on the Chromium Blog

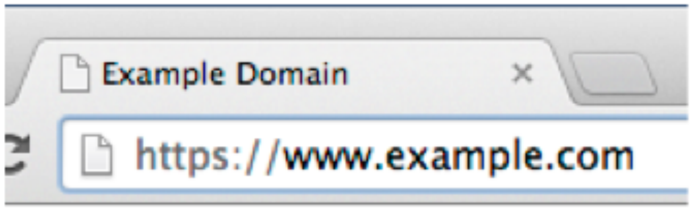
The SHA-1 cryptographic hash algorithm has been known to be considerably weaker than it was designed to be since at least 2005 — 9 years ago. Collision attacks against SHA-1 are too affordable for us to consider it safe for the public web PKI. We can only expect that attacks will get cheaper.

That's why Chrome will start the process of sunsetting SHA-1 (as used in certificate signatures for HTTPS) with Chrome 39 in November. HTTPS sites whose certificate chains use SHA-1 and are valid past 1 January 2017 will no longer appear to be fully trustworthy in Chrome's user interface.

SHA-1's use on the Internet has been deprecated since 2011, when the CA/Browser Forum, an industry group of leading web browsers and certificate authorities (CAs) working together to establish basic security requirements for SSL certificates, published their Baseline Requirements for SSL. These Requirements recommended that all CAs transition away from SHA-1 as soon as possible, and followed similar events in other industries and sectors, such as NIST deprecating SHA-1 for government use in 2010.

dismissing SHA-1 in Chrome

- Chrome 39 (Branch point 26 September 2014)
  - secure, but with minor errors
- Chrome 40 (Branch point 7 November 2014; Stable after holiday season)
  - neutral, lacking security
- Chrome 41 (Branch point in Q1 2015)
  - affirmatively insecure



HMAC

- Proposed in 1996 [Bellare Canetti Krawczyk]
  - Internet engineering task force RFC 2104
  - FIPS standard (uses a good hash function)
- Receives as input a message m, a key k and a hash function h
- Outputs a MAC by:
  - $HMAC_k(m, h) = h(k \oplus opad \,||\, h(k \oplus ipad \,||\, m))$
  - opad (outer padding: 0x5c5c5c…5c5c, one-block-long)
  - ipad (inner padding: 0x363636…3636, one-block-long)
- Theorem [BCK96]: HMAC can be forged if and only if the underlying hash function is broken (collisions found). [Bellare, Canetti, Krawczyk, 1996: http://cseweb.ucsd.edu/users/mihir/papers/hmac-cb.pdf]

HMAC - Birthday paradox

- Adversary wants to find two messages m, m' s.t. $HMAC_k(m,h) = HMAC_k(m',h)$; Adversary knows IV and h adv. does not know k
- Birthday paradox holds (you expect to find collisions with $2^{n/2+1}$ test) but the adv. is not able to check success:
  - Adv. does not know k hence he cannot generate authentic messages;
  - He must listen $2^{n/2+1}$ messages obtained with the same key (ex. n= 128 at least 2^{64+1}) to have prob. > 0.5 of a collision
- Note birthday paradox **does not** help the adv. also if we use hash(k||m||k)

HMAC in Practice

- FIPS standard
- SSL / TLS
- WTLS (part of WAP stack)
- IPSec:
  - AH
  - ESP

Exercises

Assume E_k is a good encryption function - k is the key. Show that the following are bad ways of computing MAC

- $E_k(M_1$ ex-or M_2 ex-or ... ex-or $M_n)$ or
- $E_k(M_1)$ ex-or M_2 ex-or ... ex-or M_n

- Show how an adversary can send authenticated messages using CBC if the same key is used for encryption and authentication