Authentication of messages and data integrity

What is integrity?

- Integrity = trust in the unaltered state of information
- Goal: prevent undetected or unauthorized changes
- Part of the CIA triad
  - Confidentiality
  - **Integrity**
  - Availability

we do integrity without confidentiality for now

Simple Analogy

Sealed envelope

- You receive a letter with an intact seal
- You assume the content hasn’t been altered
- If the seal is broken → loss of integrity

Why integrity matters (1)

Medical example

- Patient record
  - 10 mg → altered to → 100 mg
- Consequences: severe or fatal errors

Why integrity matters (2)

Finance & Contracts

- Transaction amount altered
- Bank account changed
- Digital contract edited after signing

Why integrity matters (3)

Software & information

- Software update injected with malware
- Download from web sites
- News article subtly modified to spread disinformation

What threatens integrity?

- **Human error**: typos, wrong input
- **Hardware failure**: bit rot, power loss
- **Malicious tampering**: insider edits
- **Transmission errors**: corrupted network packets

Two facets of integrity

1. **Data Integrity**

   Ensures the content stays unchanged

2. **Origin Integrity (authenticity)**

   Ensures the sender is legitimate

What do we want?

- A way to detect unauthorized changes
- Confidence that the information is authentic
- Methods exist to verify integrity
  - seen later
- Integrity cannot be guaranteed proactively; it can only be verified upon inspection
  - on verification failure, typical response is to reject the data, as recovery is generally not feasible

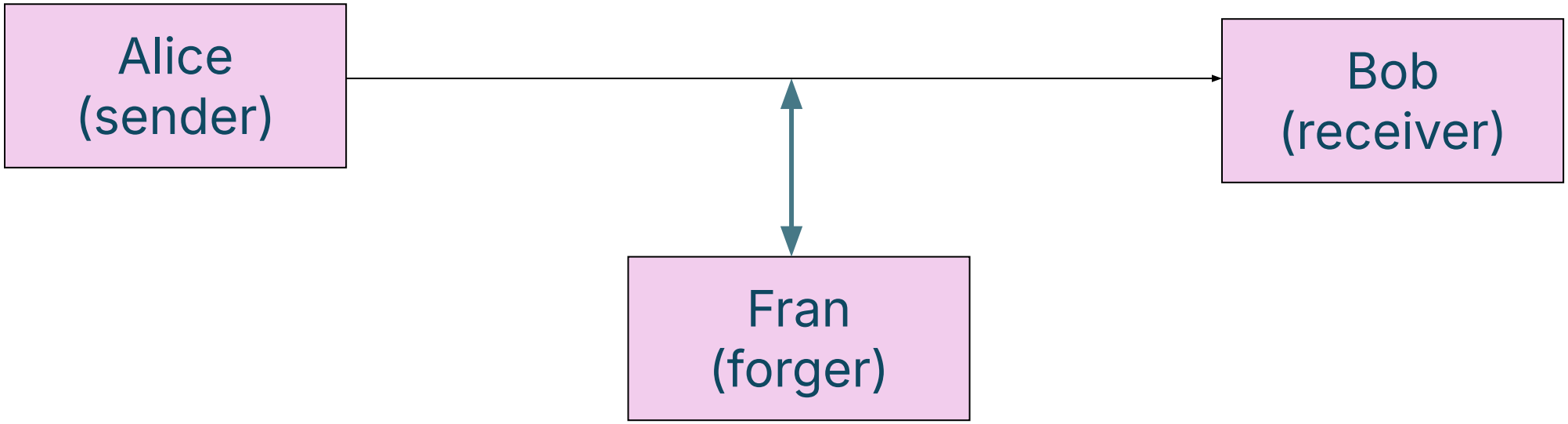
Summary

- Integrity = unaltered, trustworthy data and source
- Critical in: healthcare, finance, law, journalism etc.
- Foundation for trust in digital systems
- Before any algorithm, the need for integrity is universal
- Integrity is perhaphs the unique information security requirement that is always desired

An integrity mechanism: MAC

Recall final goal

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



Remark: Authenticity 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

Definitions (2)

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 $MAC_k(m)$

About 1:1

- More than a mere design choice, a MAC cannot assign a unique tag to each message (i.e., be one-to-one) because this would require the set of possible messages and the set of MAC values (tags) to have the same cardinality, regardless of the tag length
    - message space is vastly larger—potentially infinite—whereas tag space is finite by design. Pigeonhole principle: you cannot assign unique tags from a finite set to an unbounded set of messages
    - there are additional reasons to prefer short, fixed-length tags, including efficiency, ease of implementation and constant-time verification

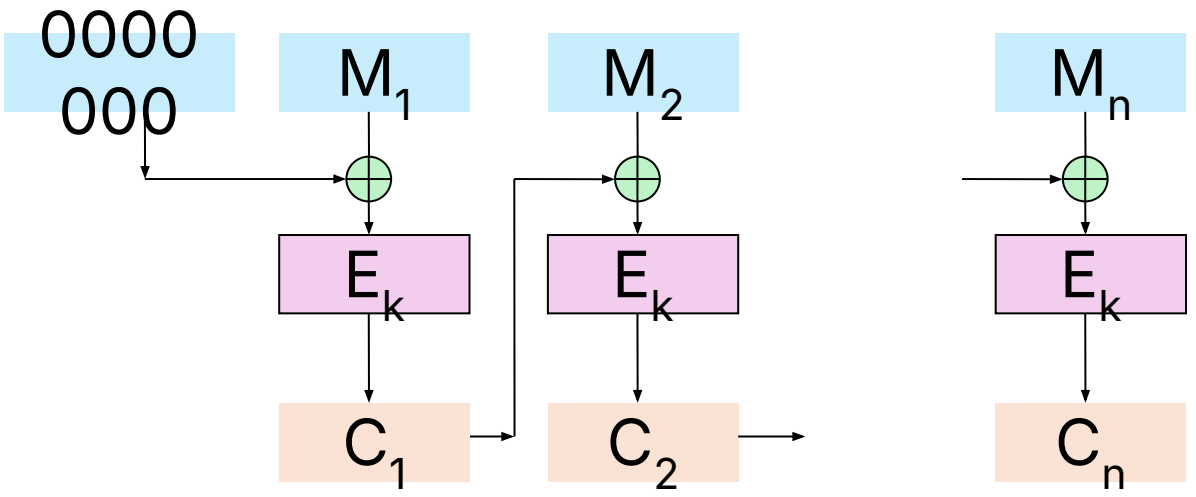
Adversary's goal

To produce a message–tag pair (m, $MAC_k(m)$) such that the verification function returns "accept", i.e., V_k (m, $MAC_k(m)$) = "accept"

- An adversary capable of controlling the communication channel (e.g., a man-in-the-middle) can easily compromise data integrity—i.e., alter the message during transmission—but cannot ensure origin integrity (authenticity) without knowledge of the secret key
- In the standard threat model, the adversary is assumed to know everything except the secret key k

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)



CBC-MAC is insecure for variable-length messages

- CBC-MAC is secure (but slow) for fixed-length messages, but 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, $m'' = m \,||\, (m_1' \oplus t) \,||\, m_2' \,||\, \ldots \,||\, m_x'$

final remarks on CBC-MAC

- secure only for messages of fixed, known length
- if confidentiality is also needed you can use CBC, but with a different shared key
- still present in some legacy or constrained-use standards
- understanding its vulnerability on variable-length messages is crucial for secure design

Attacks to integrity

Recap – Adversary's goal

- An adversary tries to forge a message–tag pair (m, τ) such that the verifier accepts it as valid:

  $$V_k(m, \tau) = \text{"accept"}$$

  where $\tau = MAC_k(m)$
- Such an adversary is called a forger, and the accepted pair (m, τ) is called a forgery

Forgery

- There are three types of forgery, with different power
    - Existential (E)
    - Selective (S)
    - Universal (U)

Existential forgery

- Adversary creates any message/signature pair (m,σ), where σ was not produced by the legitimate sender/author
- adversary need not have any control over m; m need not have any meaning
- existential forgery is essentially the weakest adversarial goal; therefore the strongest schemes are those which are "existentially unforgeable"

Selective forgery

- adversary creates a pair (m,σ) where m has been chosen by the adversary prior to the attack
- m may be chosen to have interesting mathematical properties with respect to the MAC algorithm; however, in selective forgery, m must be fixed before the start of the attack
- the ability to successfully conduct a selective forgery attack implies the ability to successfully conduct an existential forgery attack
  - **S⇒E**

Universal forgery

- adversary creates a valid tag σ for any given message m

- it is the strongest ability in forging, and it implies the other types of forgery

- **U ⇒ S ⇒ E**

Use the contrapositive

- According to the basic rules of propositional logic

$$(U \Rightarrow S \Rightarrow E) \Leftrightarrow (\neg E \Rightarrow \neg S \Rightarrow \neg U)$$

- **This means that if we can prevent E, then we are also able to prevent S and U**

About the existential forgery

An existential forgery is considered a success even if the forged message is meaningless

- Upon successful verification, the receiver might
  - wonder: “Is it encrypted?”, “Why don’t I recognize the cipher?”
  - or simply treat the message as a valid sequence of numbers

  (since any bitstring can be interpreted as a sequence of integers)

Success is defined by acceptance, not by semantic value

Hashing for integrity

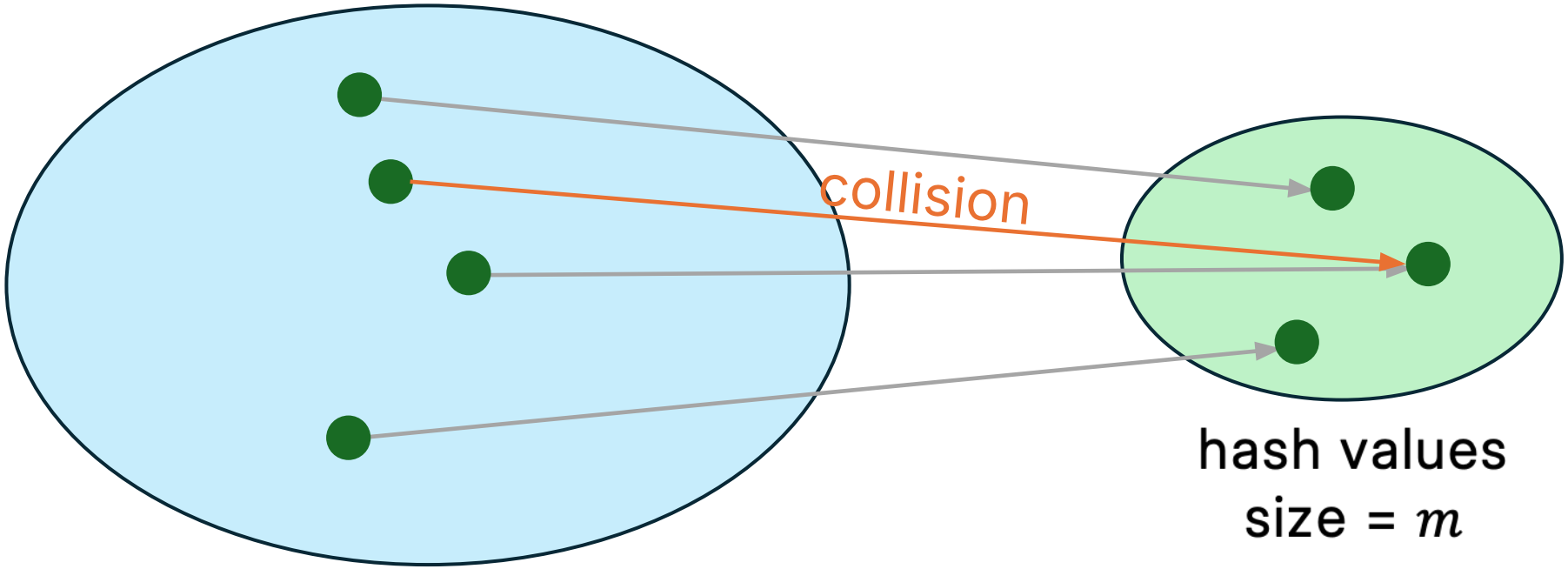
Hash functions

- 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) in computer science
- Collisions are resolved by several possible means – chaining, double hashing, etc.

Hash functions

- The idea is to implement MAC by hashing
- We need to proceed step-by-step
- We'll consider two types of hashing functions
    - Unkeyed
    - Keyed

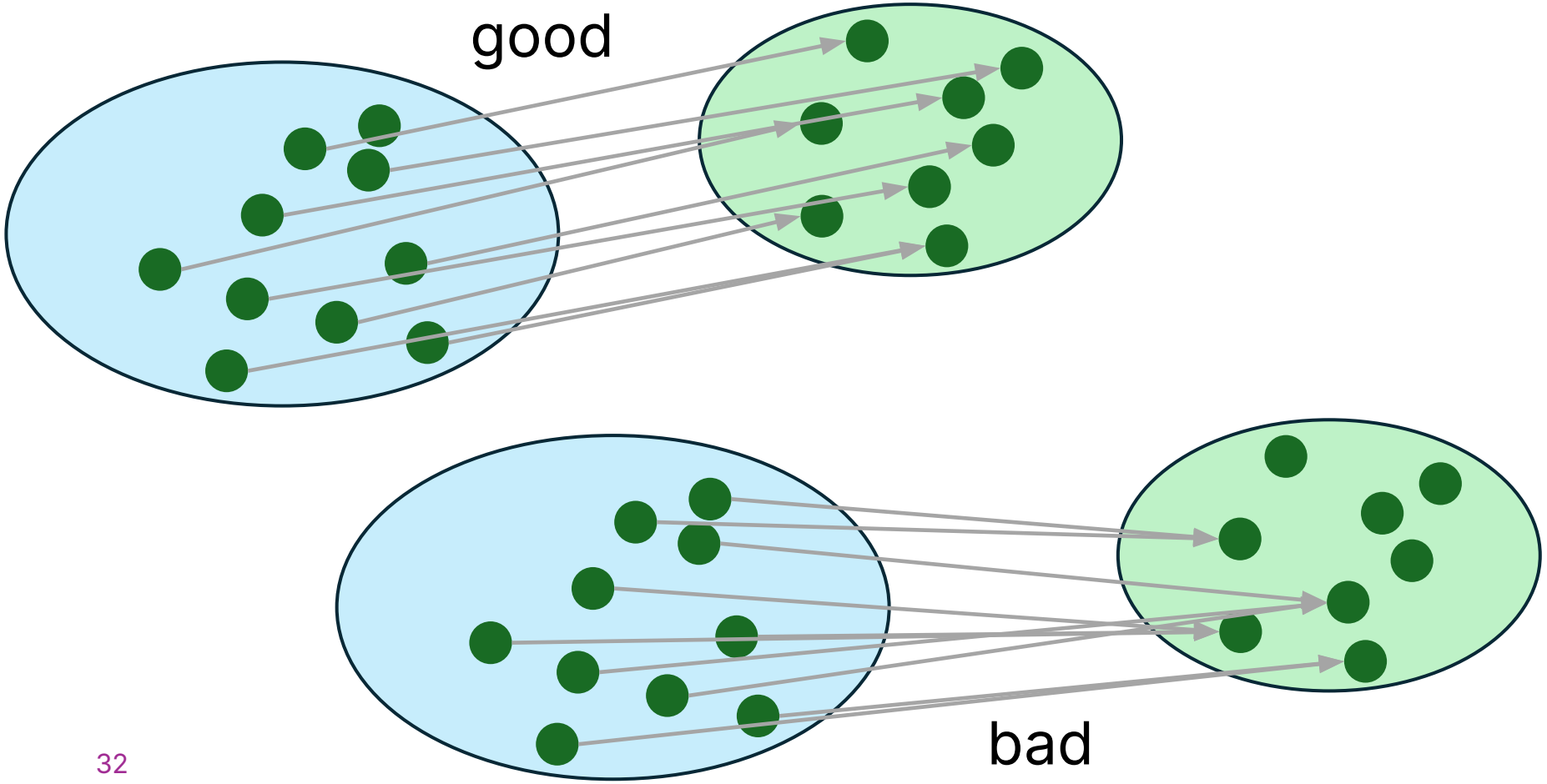
Visual



average number of collisions
$= N/m$

$$\lim_{N\to\infty} \frac{N}{m} = \infty$$

Bad and good hash



Properties of hash functions

- Avalanche effect
  - if $H(M_1)$ is the hash value of M_1, then, even if M_2 is different from M_1 in only one bit, $H(M_2)$ is strongly different from $H(M_1)$ – small changes cause big differences
- Fast to compute (except special cases that are not related to integrity)
- All hash values have the same (short) length
  - For standards, efficient operations with values and verification
- A hash value should be a random sequence of bits

About collisions

- Fact 1: collisions are (always) many
- Fact 2: the more uniformly distributed the better
- Fact 3: in traditional hashing function, although collisions are not welcome, these are handled with special algorithms
- Fact 4: in cryptography collisions are much more harmful

In cryptography hash values have two names (synonymous)

1. **digest**
2. **fingerprint**

Sending

- Alice wants to send message M to Bob, with data integrity
1. Alice computes $t = H(M)$, where H is an agreed hashing function
2. Alice sends (M, t) to Bob

Verification

Bob receives a pair (M, t)

1. Bob computes h, the fingerprint of M
2. Bob compares t and h
    1. if $t = h$ accept
    2. if $t \neq h$ reject

Practical case

- Alice is a web site
  - it contains a file F, the name of H and the fingerprint f
- Bob wants to download F
  - he reads the name of H, downloads F and f
  - he gets (F', f')
  - Bob computes $H(F')$ and verifies that this is equal to f'
  - if equal, he accepts F' as F, else no

But Fran may have compromised the web site and replaced F by F'' and f by $H(F'')$ (so, verification still works)

Other attacks to integrity

- The attacker may intercept M_1 and substitute it with a colliding message M_2

- The attacker may intercept $(M_1, H(M_1))$ and substitute it with $(M_2, H(M_2))$
  - Partial mitigation: Alice sends M_1 and $H(M_1)$ using two different channels of communication
  - Still insecure because Fran can intercept both. Just more difficult

Variant

- Bob may know already the fingerprint, so Alice doesn't need to send a pair: sending only message M is sufficient
- Still attacker can intercept transmission and replace M by a colliding M'
- Collisions are a vehicle of attack!

Summary

- Hashing functions are interesting candidate for implementing MAC functions
- Promising properties
- Still imperfect
- No origin authentication

- Some improvements are needed

Birthday paradox and attack

The Birthday Paradox

Idea first

- The birthday paradox is the surprising probability result that in a group of just 23 people, there is a greater than 50% chance that at least two people share the same birthday
- This seems counterintuitive because 23 is much less than 365, but the key lies in how many pairs of people can be formed

The Birthday Paradox

Statement

In a set of randomly chosen people, the probability that at least two of them share the same birthday exceeds 50% when the group has 23 or more individuals, assuming

- 365 equally likely birthdays
- no leap years
- birthdays are independent

The Birthday Paradox (proof)

- Build a group of people having different birthdays
- Can choose the first person in 365 ways, the second in 364 ways, third in 363 etc.
- So, probability that first has a unique birthday is 1, for second is 364/365, this 363/365 etc.
- Probability n people have different birthdays is $P(n) = \prod_{k=0}^{n-1} \left(\frac{365-k}{365}\right)$
- The probability at least two have same birthday is $1 - P(n)$
  - $1 - P(23)$ is 0.5073

The Birthday Paradox in numbers

number	probability (%)
10	11.7
20	41.1
30	70.6
50	97.0

- Hashing can be seen as mapping people to birthdays, where the range size is 365
- After inserting the 23rd person into the group the probability of collision exceeds 50%

The Birthday Paradox for length m of items in range

- Assume the range is defined by an m-bit output space (2^m digests)
- Then the probability that **at least two messages collide** exceeds 50% when

$$k \approx 1.177 \cdot 2^{m/2}$$

- is the number of randomly chosen hashed messages being mapped into the range
- The bound only depends on range size, but
- **domain should have at least k elements**

Birthday attack

- Let a cryptographic hash function output m-bit values
- A **birthday attack** finds two distinct colliding inputs $x \neq y$ with high probability after hashing roughly $k \approx 1.177 \cdot 2^{m/2}$ inputs
- Note that the brute-force bound is 2^m
- In practice, 50% probability is exceeded after roughly $\sqrt{2^m}$ attempts

Birthday attack: procedure

```
# Assume H known
map = ø
while True:
    m = generate_random_message()
    h = H(m)
    if h in hash_map:
     return (map[h], m)
    hash_map[h] = m
```

Remarks

- Expected number of iterations is $\approx 2^{m/2}$
- Attacker can't control colliding messages
  - existential forgery, the most dangerous
- The procedure can't be used for finding an element colliding with a given message
- The procedure reduces iterations from 2^m (brute-force) to $2^{m/2}$
- Digest lengths of around 160 bits are deprecated, as they yield an expected effort of 2^{80} iterations — a threshold now considered feasible and progressively rising

Check numbers

note that

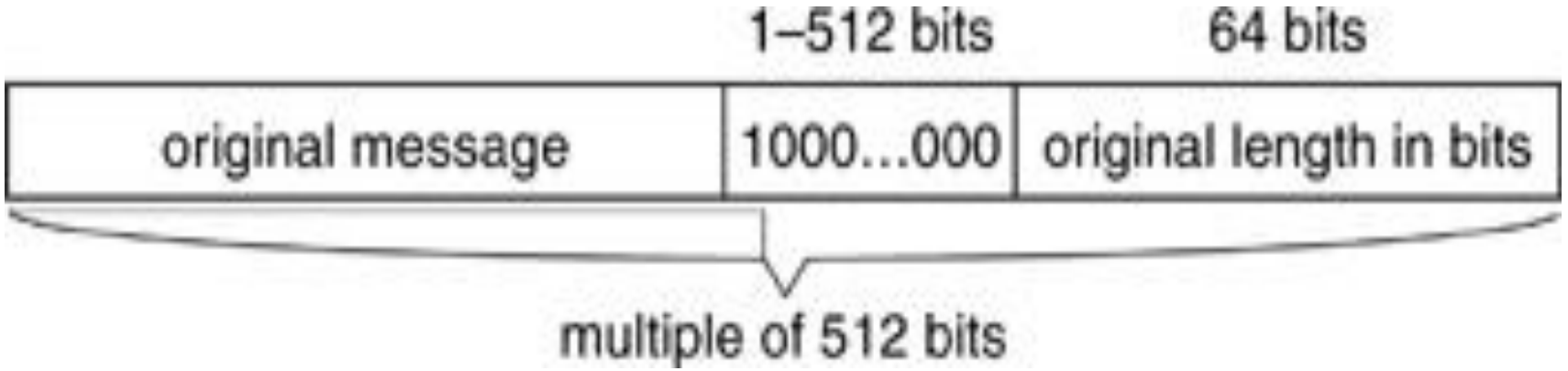
2^{80} = 1,208,925,819,614,629,174,706,176

2^{160} = 1,461,501,637,330,902,918,203,684,832,716,283,019,655,932,542,976

A hash example: SHA-1

SHA-1 basics

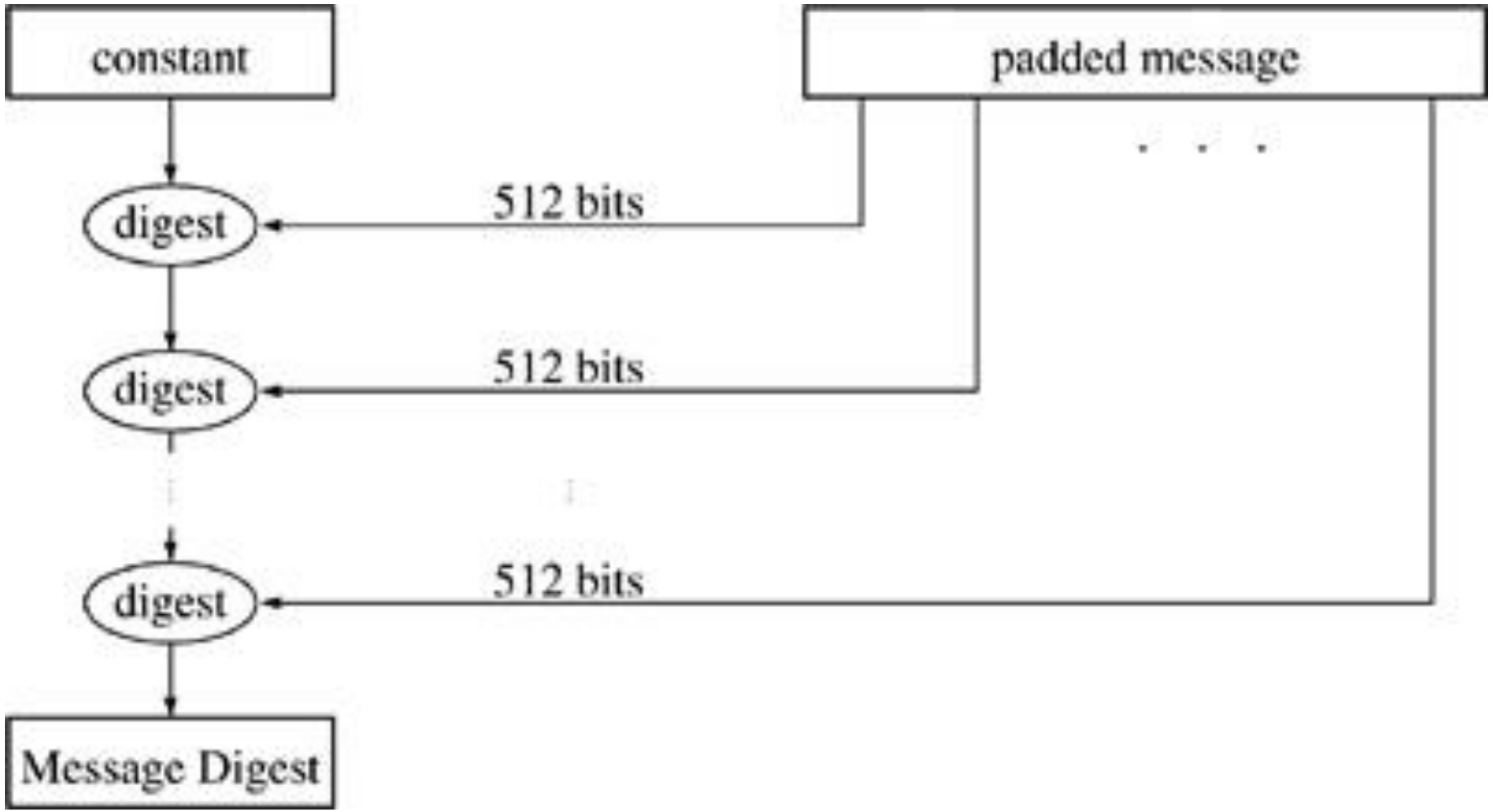
- Developed by: NIST & NSA, published in 1995 (FIPS 180-1)
- like MD4 & MD5
- $|\text{message}| < 2^{64}$
- $|\text{digest}| = 160$
- original message is padded



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

High-level scheme



SHA-1: processing one block

Block (512 bit, 5 32-bits words)

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

Round:

(A,B,C,D,E) ←

(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 words into 80 words (using shift and ex-or)
- K_t constants

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

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

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

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

Function f

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

$(B \wedge C) \vee (\sim B \wedge D) \quad (0 \leq t \leq 19)$

$B \oplus C \oplus D \quad (20 \leq t \leq 39)$

$(B \wedge C) \vee (B \wedge D) \vee (C \wedge D) \quad (40 \leq t \leq 59)$

$B \oplus C \oplus D \quad (60 \leq t \leq 79)$

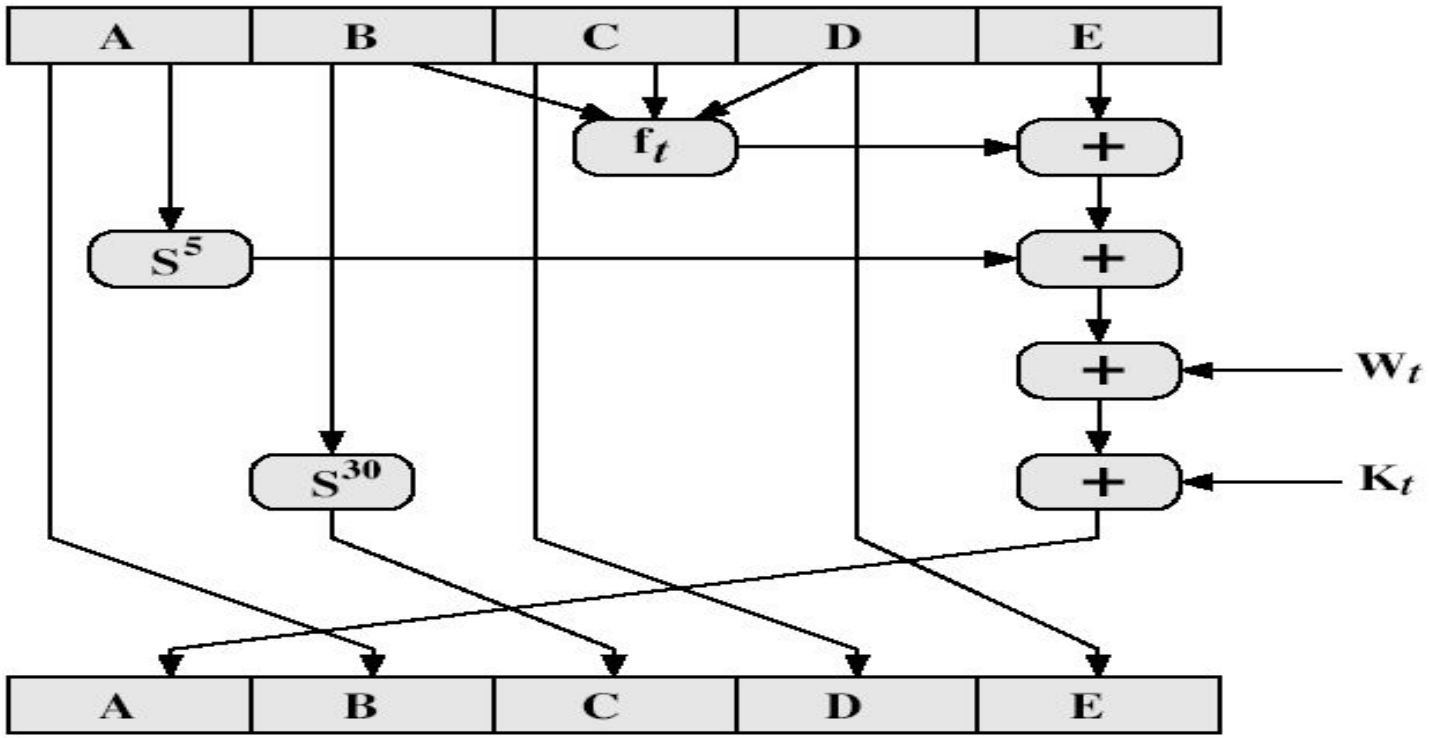
Words w_t

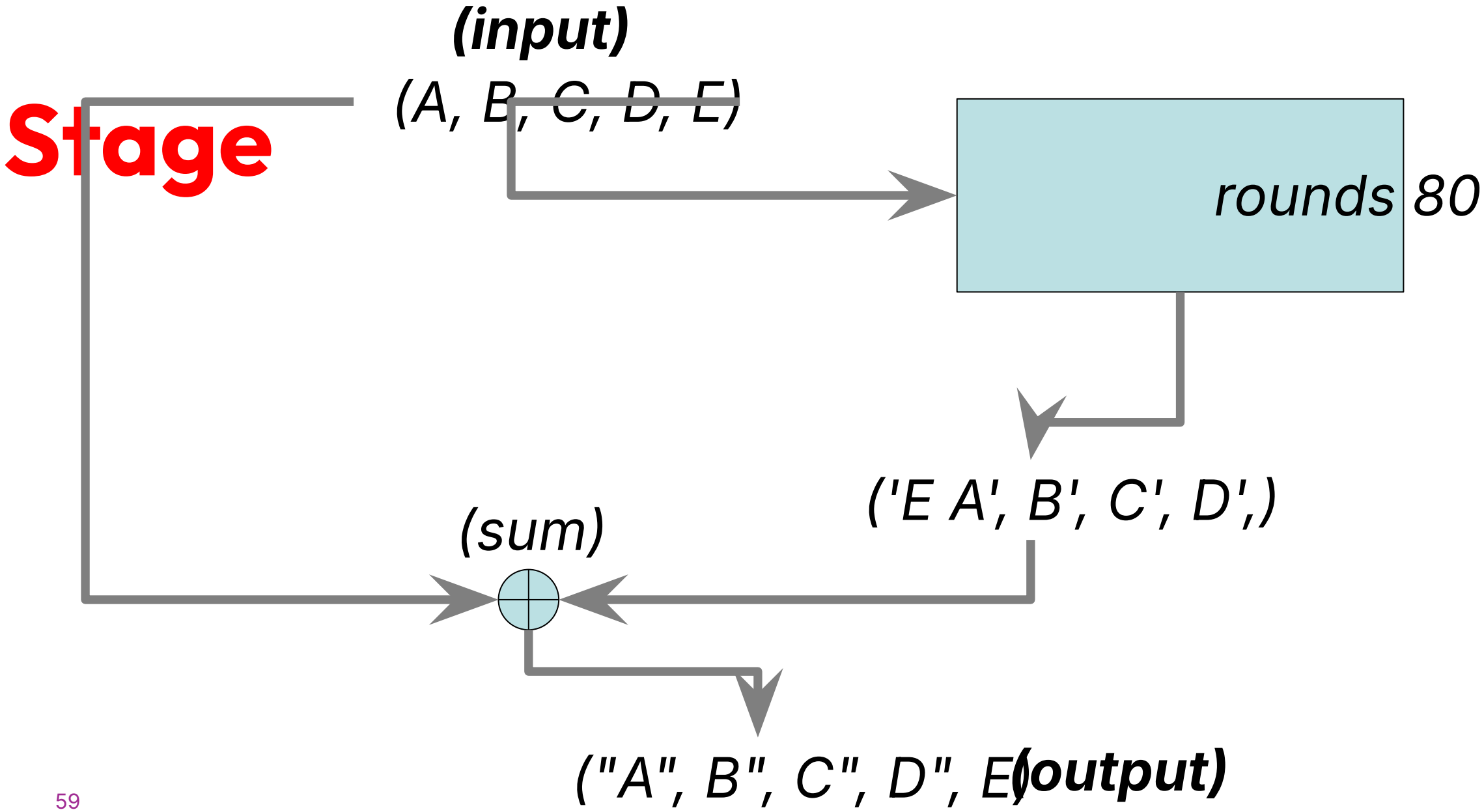
$w_0 \ldots w_{15}$ are the original 512 bits

for $16 \leq t \leq 79$

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

SHA-1: round t




(input)
(A, B, C, D, E)
Stage
rounds 80
('E A', B', C', D',)
(sum)
("A", B", C", D", E")
(output)

Summary on SHA-1

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):
    1. expand the input block to obtain 80 words block $W_0, W_1, W_2, \ldots, W_{79}$ (ex-or and shift on the given 512 bits)
    2. initialize buffer (A,B,C,D,E)
    3. update the buffer (A,B,C,D,E): execute 80 rounds
        1. each round transforms the buffer
    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,...)

Google | Online Security Blog

The latest news and insights from Google on security and safety on the Internet

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.