



Public Key Cryptography

Asymmetric Key Cryptography

- Public-Key(asymmetric) cryptography was invented in the 1970s by **Whitfield Diffie, Martin Hellman** and **Ralph Merkle**.
- Public-key cryptography needs two keys. **Public key** used to **encrypt (or code)** a message and anyone can use it. The other key allows you to **decode (or decrypt)** the message and is kept **secret (or private)** so only the person who knows the key can decrypt the message.
- First General Public-Key Algorithm used is **Knapsack Algorithm**.
- **Advantage** : No need of secure key exchange between Alice and Bob
- **An example of asymmetric cryptography** :
 - A client (for example browser) sends its public key to the server and requests for some data.
 - The server encrypts the data using client's public key and sends the encrypted data.
 - Client receives this data and decrypts it.

Trapdoor One Way Function



Locked
(difficult to unlock)



Easily Unlocked

- Publickey cryptography is an analogy to trapdoor.
- Easy to compute in one direction. Once done, it is difficult to inverse
- Trapdoor is a special function that if possessed can be used to easily invert the one way
- A one-way function is a function that satisfies the following:
 - f is easy to compute. Given x , $y=f(x)$ can be computed easily.
 - f^{-1} is difficult to compute ie $x= f^{-1}(y)$ is computationally infeasible to calculate.
 - Given y and a trapdoor, x can be computed easily.

Mathematical Trapdoor One way functions

Factorization of two primes

- Given P, Q are two primes and $N = P * Q$
- It is easy to compute N
- However given N it is difficult to factorize into P and Q
- Used in cryptosystems like RSA

Discrete Log Problem

- Consider b and g are elements in a finite group and $b^k = g$, for some k
- Given b and k it is easy to compute g
- Given b and g it is difficult to determine k
- Used in cryptosystems like Diffie-Hellman, ECC based crypto-systems

Applications

- Use of **public-key cryptosystems** can be classified into three categories
 - **Encryption /decryption:** The sender encrypts a message with the recipient's public key.
 - **Digital signature:** The sender "signs" a message with its private key. Signing is achieved by a cryptographic algorithm applied to the message or to a small block of data that is a function of the message.
 - **Key exchange:** Two sides cooperate to exchange a session key. Several different approaches are possible, involving the private key(s) of one or both parties.
- Public Key Cryptography is used in a number of applications and systems software.
 - Digitally signed document
 - E-mail encryption software such as PGP and MIME
 - Digital signatures in the Operating System software such as Ubuntu, Red Hat Linux packages distribution
 - SSL protocol
 - SSH protocol

Problems

- Slow operation
- Large cipher text size(whole of plain text is encrypted)



Knapsack Problem

- Given a set of n items with different weights W_0, W_1, \dots, W_{n-1} and a sum S , find $a_i \in \{0,1\}$ so that
- $S = a_0W_0 + a_1W_1 + \dots + a_{n-1}W_{n-1}$
- (technically, this is the *subset sum* problem)
 - Weights (62,93,26,52,166,48,91,141)
 - Problem: Find a subset that sums to $S = 302$
 - Answer: $62 + 26 + 166 + 48 = 302$ $a_i = \{10101100\}$
- The (general) knapsack is NP-complete



Knapsack Problem

- **General knapsack (GK)** is **hard** to solve. But **superincreasing knapsack (SIK)** is **easy**
- **SIK** — A superincreasing sequence is one in which the next term of the sequence is greater than the sum of all preceding terms.
- **Example:** {1, 2, 4, 9, 20, 38} is superincreasing because $1+2+4 < 9$; but {1, 2, 3, 9, 10, 24} is not because $10 < 1+2+3+9$.

How to solve?

- Weights (2,3,7,14,30,57,120,251)
- Problem: Find subset that sums to $S = 186$
- Work from largest to smallest weight
- Answer: $120 + 57 + 7 + 2 = 186$

Super Increasing Algorithm

- $S = a_1 x_1 + a_2 x_2 + \dots + a_n x_n$

$$x_n = \begin{cases} 1 \rightarrow \text{if } S \geq a_n \\ 0 \rightarrow \text{if } S < a_n \end{cases}$$

$$x_j = \begin{cases} 1 \rightarrow \text{if } S - \sum_{k=j+1}^n x_k a_k \geq a_j \\ 0 \rightarrow \text{if } S - \sum_{k=j+1}^n x_k a_k < a_j \end{cases}$$

Knapsack Cryptosystem

Plain text	10011	11010	01011	00000
Knapsack	1 6 8 15 24	1 6 8 15 24	1 6 8 15 24	1 6 8 15 24
Cipher text	1 + 15 + 24 = 40	1 + 6 + 15 = 22	6 + 15 + 24 = 45	0 = 0

- **Example**

- Weights (1, 6, 8, 15, 24)
- **Problem:** Find a subset that sums to S
- **Answer:** $1 + 15 + 24 = 40$ $a_i = \{10011\}$

- What total weights is it possible to make?
- If code is 38 then plain text is 01101.

Knapsack Cryptosystem

- Knapsack Cryptosystem was invented by **Ralph Merkle** and **Martin Hellman** in **1978**.
- One algorithm that uses a **superincreasing knapsack** for the **private (easy) key** and a **non-superincreasing knapsack** for the **public key** was created by **Merkle** and **Hellman**. They did this by taking a superincreasing knapsack problem and converting it into a non-superincreasing one that could be made public, **using modulus arithmetic**.
- **General Idea.**
- Given superincreasing knapsack (SIK). Convert SIK to “general” knapsack (GK) by **using modulus arithmetic**.
- **Public Key:** GK.
- **Private Key:** SIK and conversion factor.

Knapsack Cryptosystem

1. Take a super-increasing sequence.
2. Multiply all the values by a number, m , modulo n . The n should be a number greater than the sum of all the numbers in the sequence. The multiplier m should have no factors in common with the modulus ie, $\gcd(n,m)=1$ relatively prime. Its preferable to take n as a prime number due to its features.
3. **public key** is the resultant value after multiplication followed by modulo arithmetic
4. **Private key** is original super-increasing sequence.
5. **Encryption:** Split message into block size equivalent to no: of items in knapsack. Use non-SIK(public key)to encrypt message.
6. **Decryption:** calculate n^{-1} , which is a multiplicative inverse of $n \bmod m$, i.e. $n \times n^{-1} = 1 \bmod m$. Multiply each of the codes with n^{-1} and apply modulo M . From SIK (private key) calculate plain text back.

Challenge 1

- Perform Knapsack Crypto algorithm using $\{1, 2, 4, 10, 20, 40\}$ and explain how encryption and decryption can be done on the message 100100111100101110 where $m=31$ and $n=110$

SOLUTION

Let $n=110$ and $m=31$. The normal knapsack sequence would become:

$$1 \times 31 \bmod(110) = 31$$

$$2 \times 31 \bmod(110) = 62$$

$$4 \times 31 \bmod(110) = 14$$

$$10 \times 31 \bmod(110) = 90$$

$$20 \times 31 \bmod(110) = 70$$

$$40 \times 31 \bmod(110) = 30$$

public key : {31, 62, 14, 90, 70, 30}

private key: {1, 2, 4, 10, 20, 40}.

SOLUTION

- **message:**100100111100101110
Since knapsack contains six weights so we need to split the message into groups of six:
{100100, 111100, 101110}
corresponds to message the sets of weights with totals are as follows
 $100100 = 31 + 90 = 121$
 $111100 = 31+62+14+90 = 197$
 $101110 = 31+14+90+70 = 205$
Encrypted message{ciphertext}: 121 197 205.
- The person decoding must know the two numbers the modulus and the multiplier(110 and 31) and calculate n^{-1} to be 71.
To decode the message.
 $121 \times 71 \bmod(110) = 11 = 100100 \{1, 2, 4, 10, 20, 40\}$
 $197 \times 71 \bmod(110) = 17 = 111100$
 $205 \times 71 \bmod(110) = 35 = 101110$
Decrypted message(PlainText): 100100111100101110.

Challenge 2

1. Perform Knapsack Crypto
algorithm using $(2, 3, 7, 14, 30, 57, 120, 251)$ and
explain how encryption and decryption can be
done on the message 10010110 where
 $m = 41$ and $n = 491$

Example 2

- Start with (2,3,7,14,30,57,120,251) as the SIK
- Choose $m = 41$ and $n = 491$ (m, n relatively prime, n exceeds sum of elements in SIK)
- Compute “general” knapsack
$$2 \cdot 41 \bmod 491 = 82$$
$$3 \cdot 41 \bmod 491 = 123$$
$$7 \cdot 41 \bmod 491 = 287$$
$$14 \cdot 41 \bmod 491 = 83$$
$$30 \cdot 41 \bmod 491 = 248$$
$$57 \cdot 41 \bmod 491 = 373$$
$$120 \cdot 41 \bmod 491 = 10$$
$$251 \cdot 41 \bmod 491 = 471$$
- “General” knapsack:
(82,123,287,83,248,373,10,471)

Example 2

- **Private key:**

(2,3,7,14,30,57,120,251)

$$m^{-1} \bmod n =$$

$$41^{-1} \bmod 491 = 12$$

- **Public key:**

(82,123,287,83,248,373,10,471),
n=491

- Example: Encrypt 10010110

$$82 + 83 + 373 + 10 = 548$$

- To decrypt, use private key...

- $548 \cdot 12 = 193 \bmod 491$

- Solve (easy) SIK with $S = 193$

- Obtain plaintext 10010110

Knapsack Weakness

- **Trapdoor:** Convert SIK into “general” knapsack using modular arithmetic
- **One-way:** General knapsack easy to encrypt, hard to solve; SIK easy to solve
- This knapsack cryptosystem is **insecure**
 - Broken in 1983 with Apple II computer
 - The attack uses **lattice reduction**
- “General knapsack” is not general enough!
 - This special case of knapsack is easy to break

The image features the RSA logo in a bold, white, sans-serif font. The letters are centered against a vibrant red background that is filled with dynamic, diagonal light streaks, creating a sense of motion and energy. The entire composition is framed by a white border, which is itself set against a dark red background.

RSA



RSA

- RSA algorithm is asymmetric cryptography algorithm(two different keys i.e. **Public Key** and **Private Key**-a key pair).
- Proposed by **Rivest**, **Shamir**, and **Adleman** of MIT in 1977 and a paper was published in The Communications of ACM in 1978, patented in 1983, expired in 2000.
- **Idea** of RSA is based on the fact, it is **difficult to factorize a large integer**.
- **Secure** due to **cost of factoring large numbers**.
 - **Factorization** takes $O(e^{\log n \log \log n})$ operations (hard)
 - Given the private key it is easy to derive the public key
 - Given the public key it is difficult to derive the private key
- public key consists of two numbers where **one number** is **multiplication of two large prime numbers**. And private key is also derived from the same two prime numbers.
- encryption strength totally lies on the key size(preferably 1024 or 2048 bits) and strength of encryption increases exponentially.

RSA Algorithm

-
1. Select two very large (100+ digit) prime numbers **P** and **Q**.
 2. Calculate : **$N = P * Q$** and **$\phi(N) = (p-1)(q-1)$**
 3. Select the **Public key**(Encryption Key) **e** where **$1 < e < \phi(N)$** , such that it is not a factor of $\phi(N)$ [ie (P-1) and (Q-1)]. Or **$\gcd(e, \phi(N)) = 1$** ie **$\gcd(e, (p-1)(q-1)) = 1$**
 4. Choose any large integer, **D**, which is **private key**(Decryption Key) such that **$E * D = 1 \pmod{\phi(N)}$** and $0 \leq d \leq N$
 - Alice **encrypts** M as **$C \equiv P^e \pmod{n}$**
 - Bob **decrypts** by computing **$P \equiv C^d \pmod{n}$**
 - Bob makes (e,n) public and (p,q,d) secret

Note: message M must be smaller than the modulus N (block if needed)

RSA Example 1

- Perform RSA encryption on message $m = 88$ and $n=187$
- 1. Select primes: $p=17$ & $q=11$ where $n = pq = 17 \times 11 = 187$
- 2. Compute $\phi(n) = (p-1)(q-1) = 16 \times 10 = 160$
- 3. Select e : $\gcd(e, 160) = 1$; choose $e=7$
- 4. Determine d : $de=1 \pmod{160}$ and $d < 160$ Value is $d=23$ since $23 \times 7 = 161 = 10 \times 160 + 1$

Publish public key $\{e, n\} = \{7, 187\}$

Keep secret private key $(p,q,d) = \{17, 11, 23\}$

- given message $M = 88$ (nb. $88 < 187$)
- encryption:

$$C = 88^7 \pmod{187} = 11$$

decryption:

$$M = 11^{23} \pmod{187} = 88$$

RSA Example2

- Perform RSA encryption on message $m = 7$, and $n=33$
 1. Select primes **$p=11$, $q=3$** .
 2. **$n = pq = 11*3 = 33$**
 $\varphi(N)=(p-1)(q-1)= 10*2 = 20$
 3. Choose $e=3$. Check $\gcd(e, p-1) = \gcd(3, 10) = 1$ (i.e. 3 and 10 have no common factors except 1), and check $\gcd(e, q-1) = \gcd(3, 2) = 1$ therefore $\gcd(e, \phi) = \gcd(e, (p-1)(q-1)) = \gcd(3, 20) = 1$
 4. Compute d such that $ed \equiv 1 \pmod{\phi}$
i.e. compute $d = (1/e) \pmod{\phi} = (1/3) \pmod{20}$
i.e. find a value for d such that ϕ divides $(ed-1)$
i.e. find d such that 20 divides $3d-1$.
Simple testing ($d = 1, 2, \dots$) gives $d = 7$
Check: $ed-1 = 3*7 - 1 = 20$, which is divisible by ϕ .
 5. Public key = $(n, e) = (33, 3)$
Private key = $(n, d) = (33, 7)$.

RSA Challenge

- One of the first description of RSA was in the paper.
- Martin Gardner: [Mathematical games](#) Scientific American, 1977 and in this paper RSA inventors presented the following challenge.
- **Decrypt the cryptotext:**
 - 9686 9613 7546 2206 1477 1409 2225 4355 8829
0575 9991 1245 7431 9874 6951 2093 0816 2982
2514 5708 3569 3147 6622 8839 8962 8013
3919 9055 1829 9451 5781 515
 - **Encrypted using the RSA cryptosystem with**
 - n : 114 381 625 757 888 867 669 235 779 976 146
612 010 218 296 721 242 362 562 561 842 935 706
935 245 733 897 830 597 123 513 958 705 058 989
075 147 599 290 026 879 543 541.
 - and with $e = 9007$
 - The problem was solved in 1994 by first factorizing n into one 64-bit prime and one 65-bit prime, and then computing the **plaintext**
 - **THE MAGIC WORDS ARE SQUEMISH OSSIFRAG**

RSA Challenge

- Take two primes $p = 653$ and $q = 877$ and message $x = 12345$. Encrypt message using RSA.

Realistic Example for RSA

Example of the design and of the use of RSA cryptosystems. By choosing $p = 41, q = 61$ we get $n = 2501, \phi(n) = 2400$

- By choosing $d = 2087$ we get $e = 23$

Plaintext: KARLSRUHE

Encoding: 100 017 111 817 200 704

- Since $10^3 < n < 10^4$, the numerical plaintext is divided into blocks of 3 digits \Rightarrow 6 plaintext integers are obtained

Encryption:

$$100^{23} \bmod 2501, 17^{23} \bmod 2501, 111^{23} \bmod 2501 \\ 817^{23} \bmod 2501, 200^{23} \bmod 2501, 704^{23} \bmod 2501$$

provides cryptotexts: 2306, 1893, 621, 1380, 490, 313

Decryption:

$$2306^{2087} \bmod 2501 = 100, 1893^{2087} \bmod 2501 = 17 \\ 621^{2087} \bmod 2501 = 111, 1380^{2087} \bmod 2501 = 817 \\ 490^{2087} \bmod 2501 = 200, 313^{2087} \bmod 2501 = 704$$



when $x \in Z_n$ and $\gcd(x, n) = 1$



Encryption

$$e_K(x) = y = x^b \bmod n$$

where $x \in Z_n$

Decryption

$$d_K(x) = y^a \bmod n$$

$$\begin{aligned} y^a &\equiv (x^b)^a \bmod n \\ &\equiv (x^{ab}) \bmod n \\ &\equiv (x^{t\phi(n)+1}) \bmod n \\ &\equiv (x^{t\phi(n)} x) \bmod n \\ &\equiv x \end{aligned}$$

$$\begin{aligned} ab &\equiv 1 \bmod \phi(n) \\ ab - 1 &= t\phi(n) \\ ab &= t\phi(n) + 1 \end{aligned}$$

From Fermat's theorem

Why RSA works?



Why RSA works?

Because of Euler's Theorem:

- $a^{\phi(N)} \bmod N = 1$
 - where $\gcd(a, N) = 1$
- in RSA have:
 - $N = p \cdot q$
 - $\phi(N) = (p-1)(q-1)$
 - carefully chosen e & d to be inverses $\bmod \phi(N)$
 - hence $e \cdot d = 1 + k \cdot \phi(N)$ for some k
- hence :
$$C^d = (M^e)^d = M^{1+k \cdot \phi(N)} = M^1 \cdot (M^{\phi(N)})^k = M^1 \cdot (1)^k = M^1 = M \bmod N$$

How to design a good RSA cryptosystem?

1. How to choose large primes p, q ?

- Choose randomly a large integer p , and verify, using a randomized algorithm, whether p is prime. If not, check $p + 2, p + 4, \dots$
- From the Prime Number Theorem it follows that there are approximately

$$\frac{2^d}{\log 2^d} - \frac{2^{d-1}}{\log 2^{d-1}}$$

- d bit primes. (A probability that a 512-bit number is prime is 0.00562.)

How to design a good RSA cryptosystem?

2. What kind of relations should be between p and q ?

2.1 Difference $|p-q|$ should be neither too small nor too large.

2.2 $\gcd(p-1, q-1)$ should not be large.

2.3 Both $p-1$ and $q-1$ should contain large prime factors.

2.4 Quite ideal case: q, p should be safe primes - such that also $(p-1)/2$ and $(q-1)/2$ are primes (83,107,10¹⁰⁰ – 166517 are examples of safe primes).

3. How to choose e and d ?

3.1 Neither d nor e should be small.

3.2 d should not be smaller than $n^{1/4}$. (For $d < n^{1/4}$ a polynomial time algorithm is known to determine d .)

RSA Security

- Computing d from e and n is believed to be hard (requires factoring n to find p, q). Security depends on the difficulty of factoring n
 - Factor $n \Rightarrow \Phi(n) \Rightarrow$ compute d from $(e, \Phi(n))$
- Computing P from $P^e \pmod n$ is believed to be hard (discrete logarithm).
- Numbers up to 663 bits have been factored.
- A theoretical attack exists using a quantum computer.
 - Shor's algorithm solves both the discrete logarithm and factoring.
- The length of $n=pq$ reflects the strength. Minimal 2048 bits recommended for current usage
 - 700-bit n factored in 2007
 - 768 bit factored in 2009

RSA Attacks

- 3 approaches to attack RSA:
 - brute force key search (infeasible given size of numbers)
 - mathematical attacks (based on difficulty of computing $\phi(N)$, by factoring modulus N)
 - timing attacks (on running of decryption)
- Small message/exponent attack
 - If $m^e < n$, then m is easy to find.
 - m should be padded with random data.
- Factoring
 - If p and q have only small factors, then n is easy to factor.
 - If p is close to q then n is easy to factor.

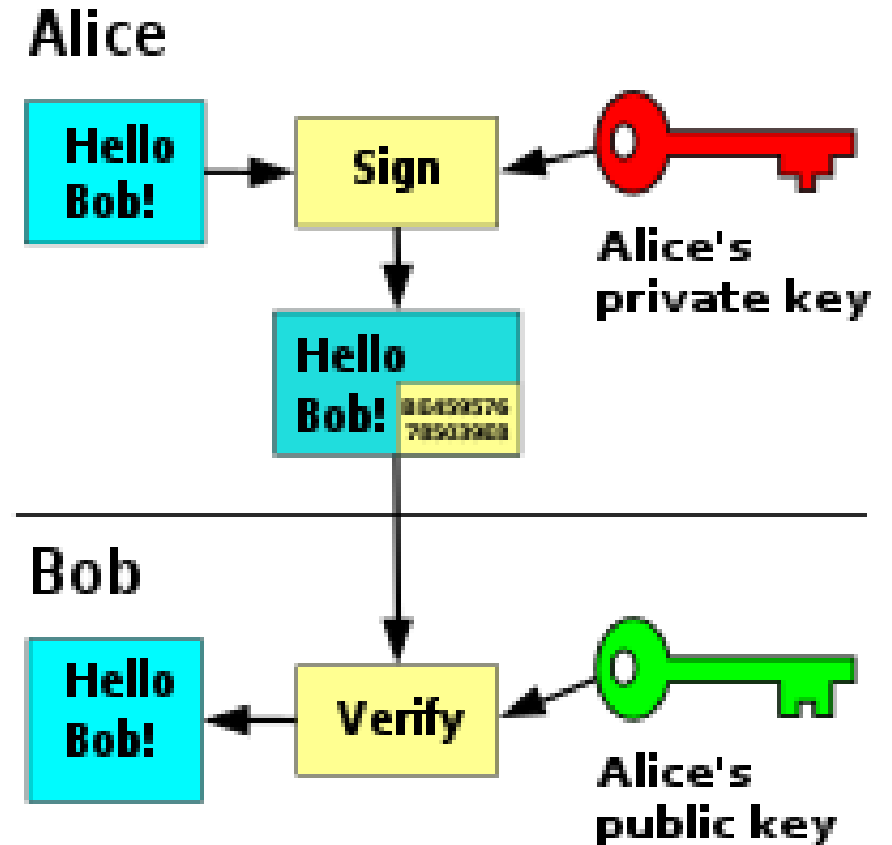


Digital Signatures

Digital Signature

- A Digital Signature is the result of **encrypting** the Hash of the data to be exchanged.
- A **digital signature** is a mathematical scheme for verifying the authenticity of digital messages or documents. A valid digital signature gives a recipient reason to believe that the message was created by a known sender (**authentication**), that the sender cannot deny having sent the message (**non-repudiation**), and that the message was not altered in transit (**integrity**)
- It increases transparency of online interactions and develop trust between customers using **Hash Function** and **Public Key Cryptography**

Eg: DSA(Digital Signature Algorithm)

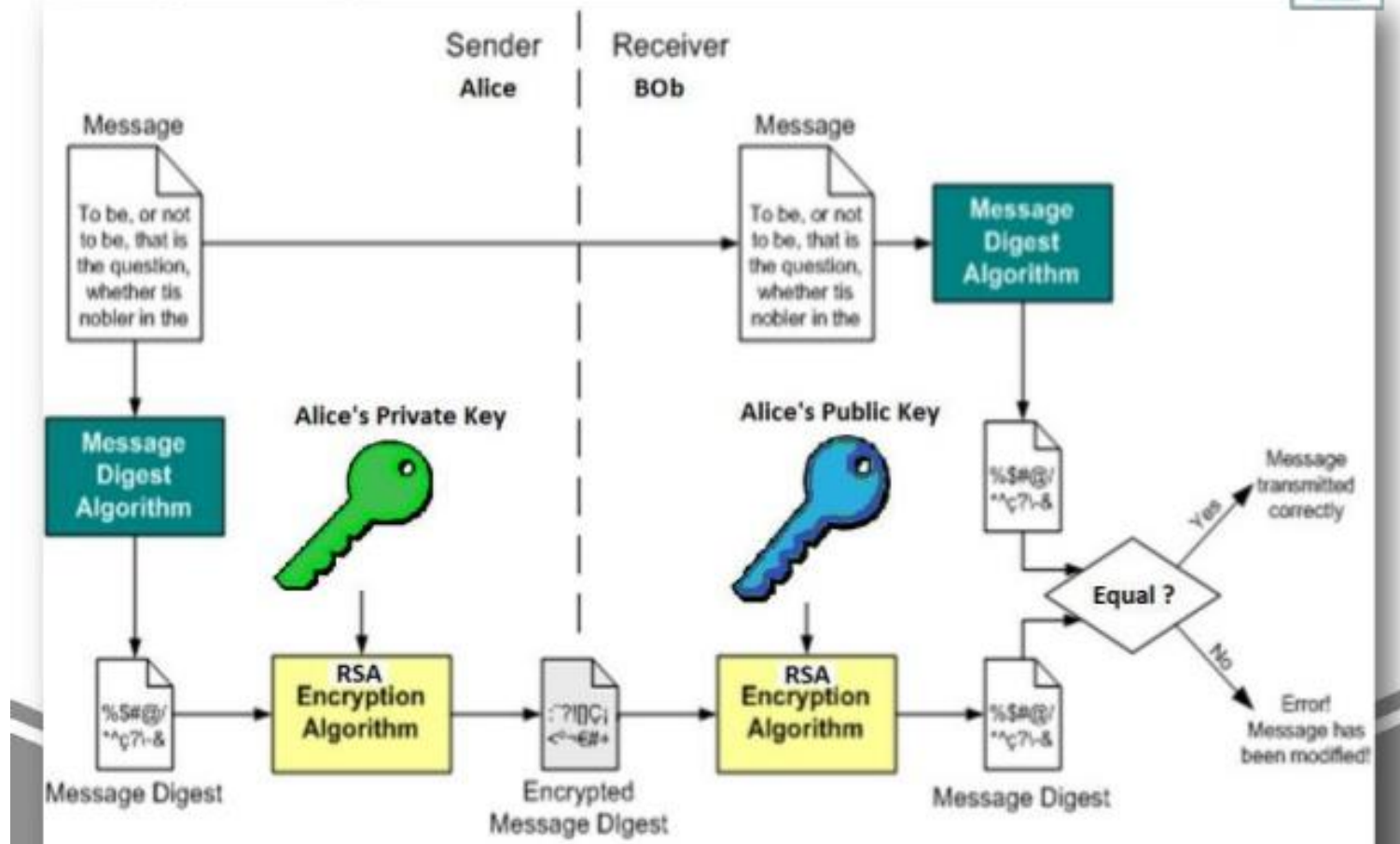




RSA Digital Signature

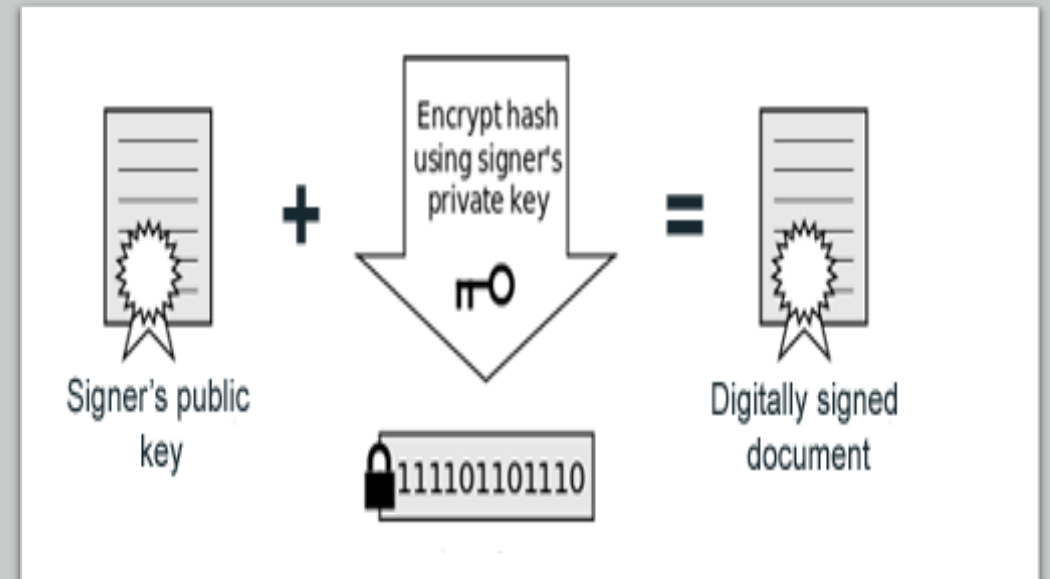
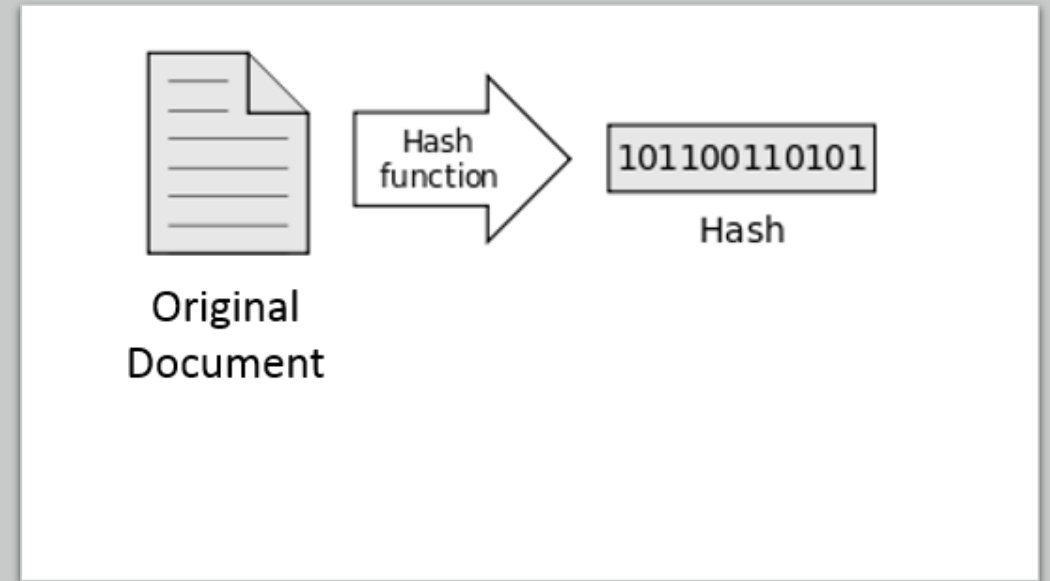
- Encryption:
 $C \equiv P^d \pmod{n}$
- Decryption:
 $P \equiv C^e \pmod{n}$

Digital Signature on RSA



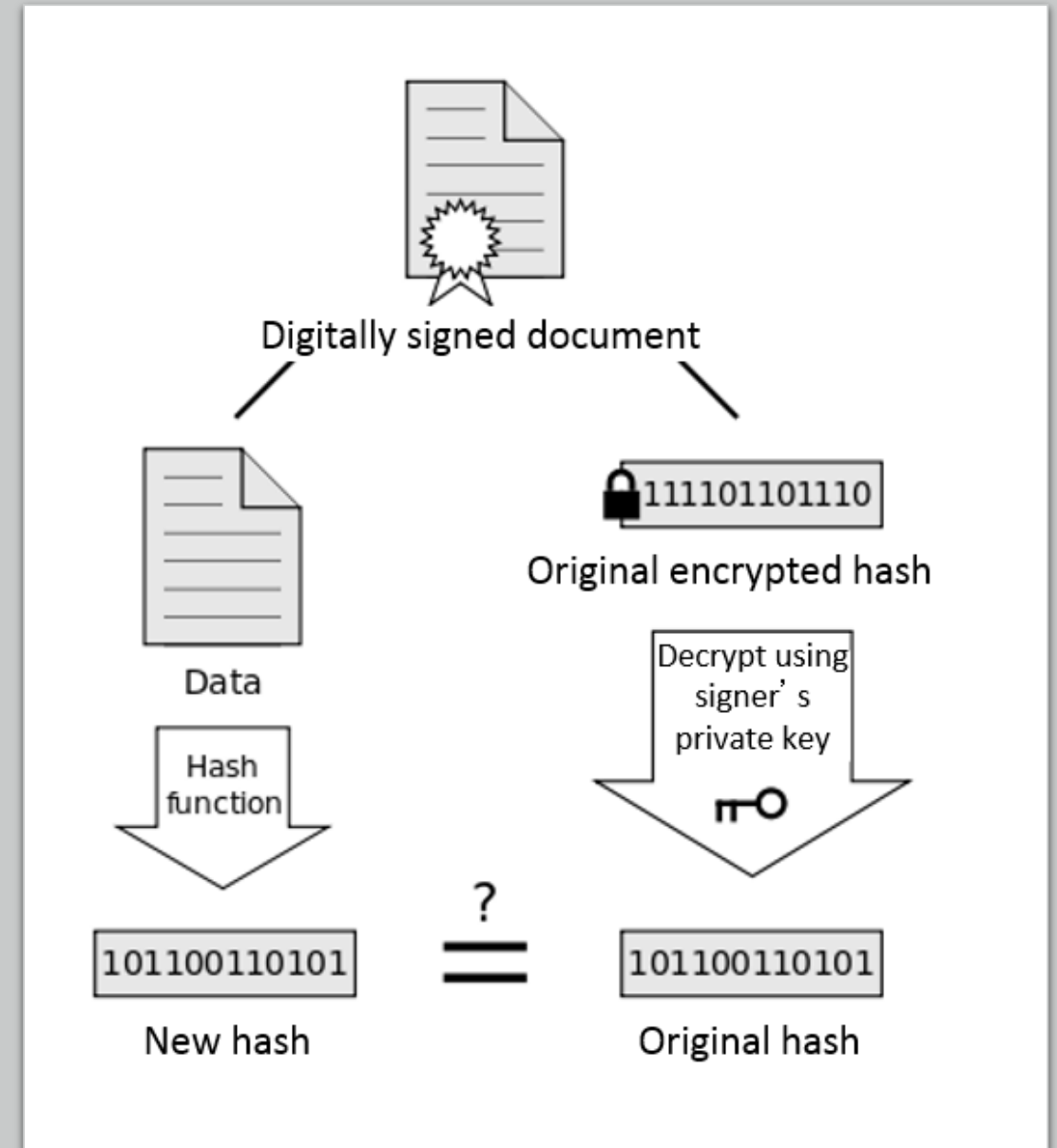
How digital signature Works in real life?

1. The document's hash is created using a mathematical algorithm. This hash is specific to this particular document; even the slightest change would result in a different hash.
2. The hash is encrypted using the signer's private key. The encrypted hash and the signer's public key are combined into a digital signature, which is appended to the document.
3. The digitally signed document is ready for distribution.

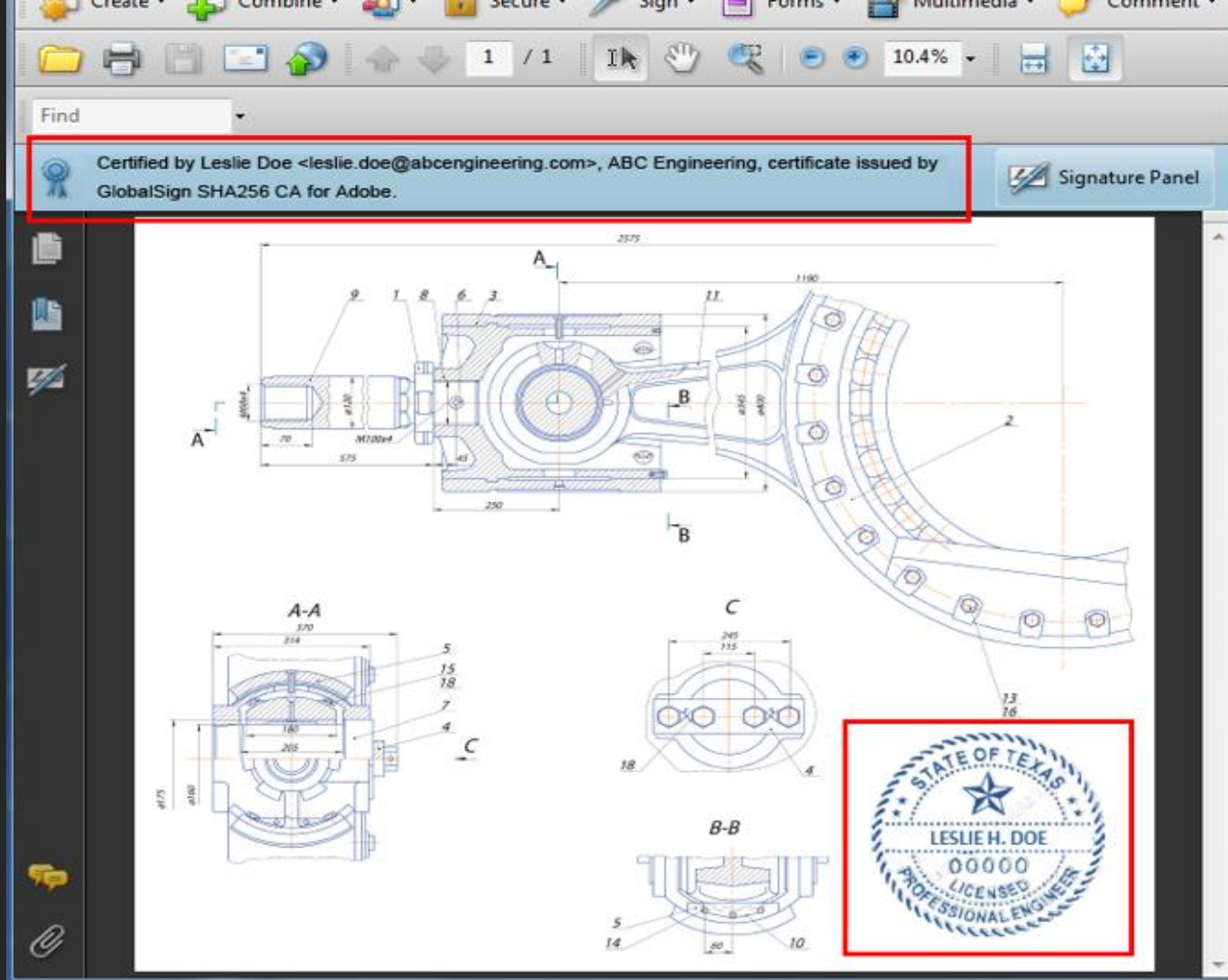


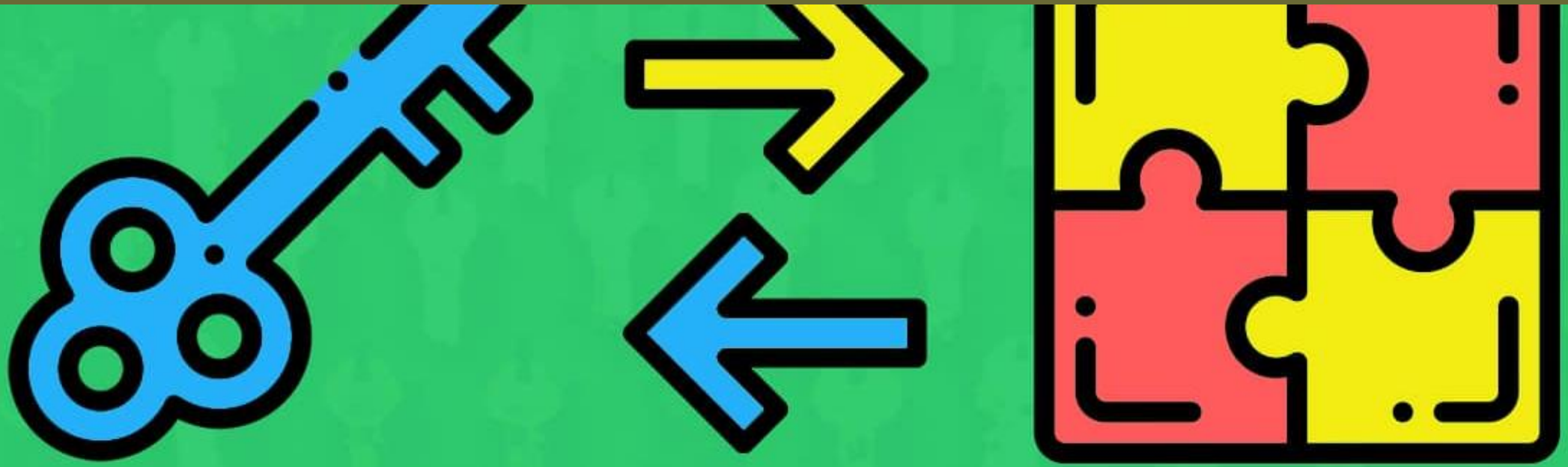
Verifying Signature

1. Document uses the signer's public key (which was included in the digital signature with the document) to decrypt the document hash.
2. The program calculates a new hash for the document. If this new hash matches the decrypted hash from Step 1, the program knows the document has not been altered and displays messaging along the lines of, "The document has not been modified since this signature was applied." The program also validates that the public key used in the signature belongs to the signer and displays the signer's name.



EXAMPLES

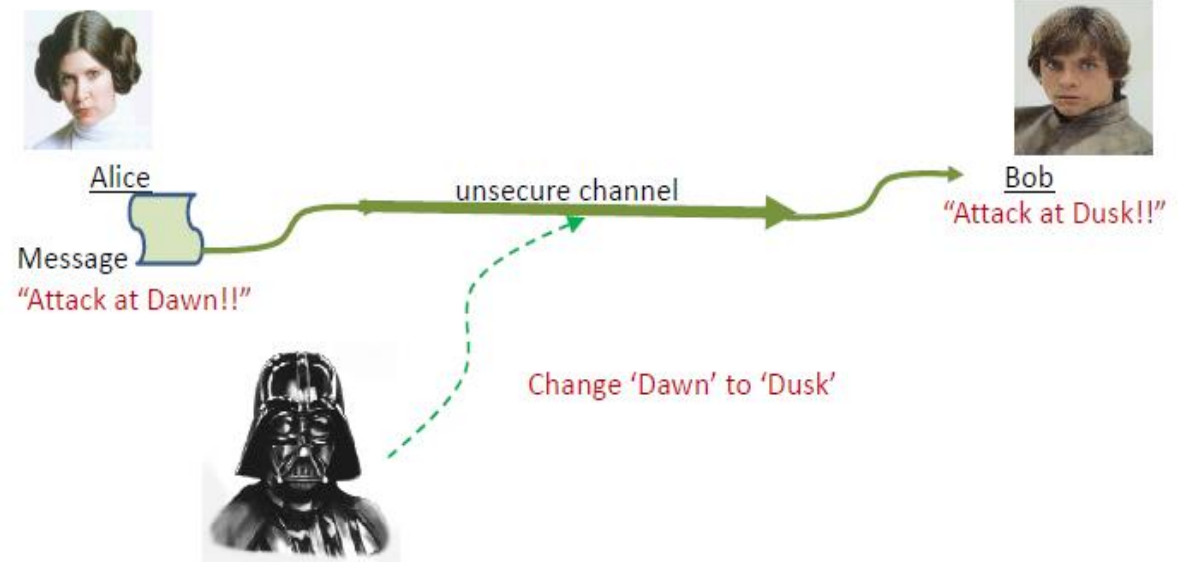




HASH FUNCTION

Ensuring Integrity

- How can Bob ensure that Alice's message has not been modified?

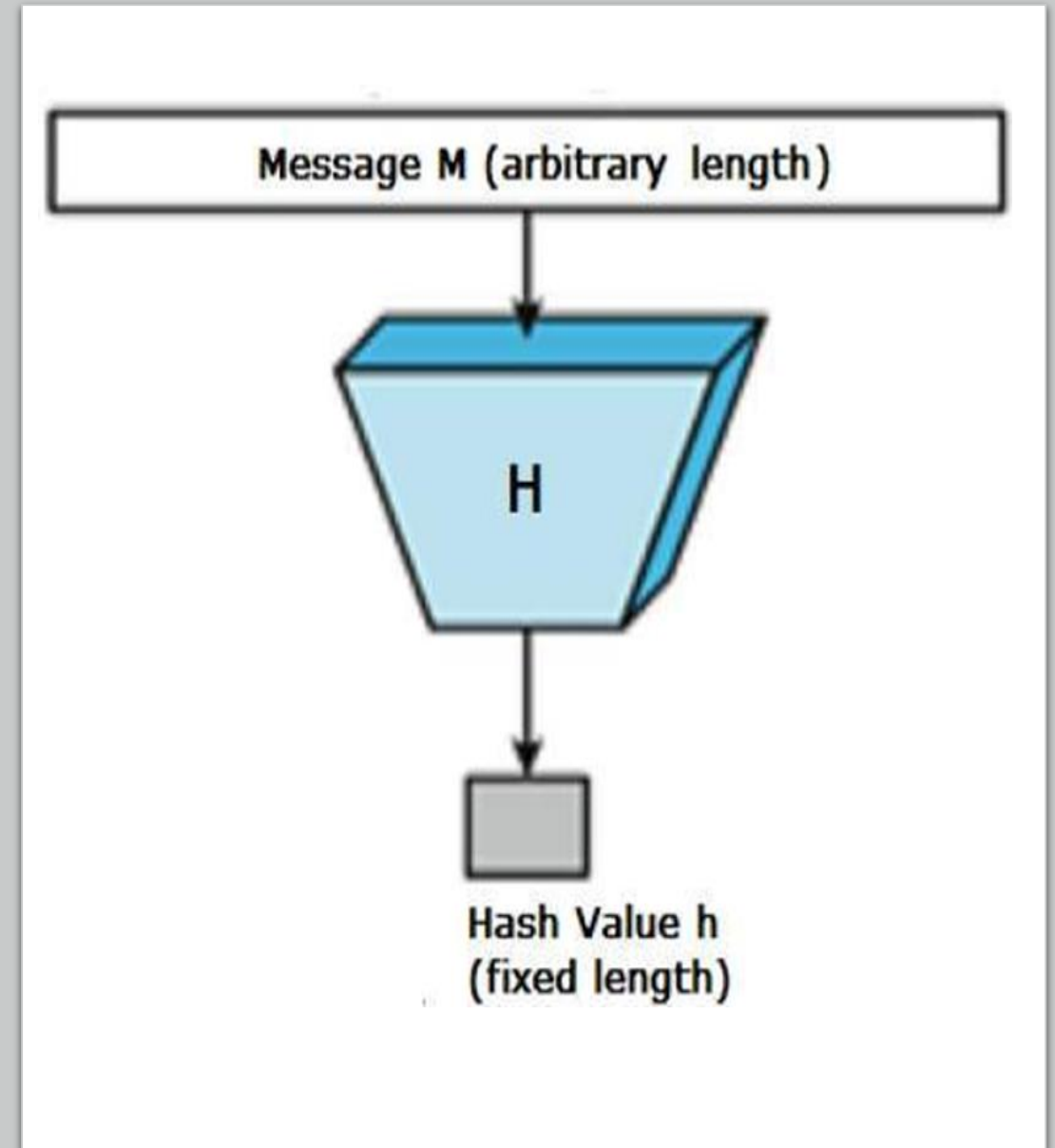


SOLUTION

- **Cryptographic hash function**

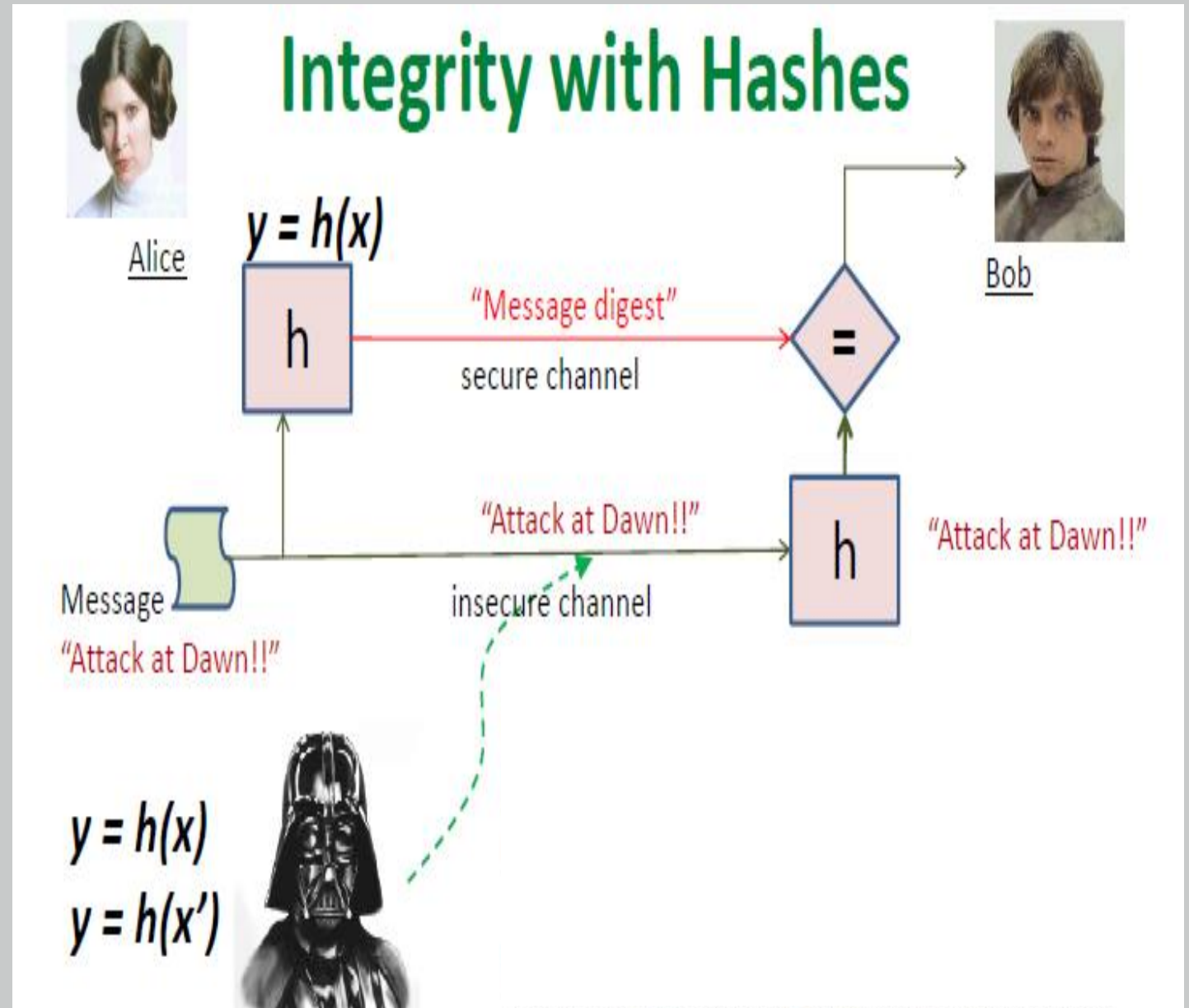
A hash function H is a mathematical transformation that accepts message block M of any length and gives a fixed-length (short) hash value $h=H(M)$ as output.

- Output or value produced by hash function is called as **checksum or message digest**.
- Even a small change in the message will result in a completely new message digest
- Typically of 160 bits, irrespective of the message size.
- Hash functions are specially designed to resist such collisions



MESSAGE DIGEST WORKING

- Alice passes the message through a hash function, which produces a fixed length message digest. The message digest is representative of Alice's message.
- Bob re-computes a message hash and verifies the digest with Alice's message digest.
- Mallory does not have access to the digest y . Even if she modifies Alice's message from x to x' , the modification can be detected unless **$h(x) = h(x')$**





Hashing

USES

- Digital Signatures
- Message Fingerprinting
- Error Detection
- Spam Reduction

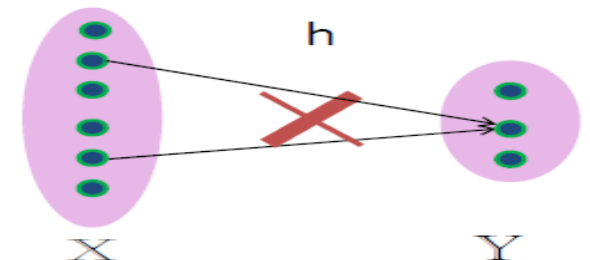
EXAMPLES

- MD2, MD4, MD5 – 128 bits
- SHA-256, 384, 512 bits
- Whirlpool – 512 bits
- Tiger – 192 bits

Properties Of Good Cryptographic Hash Function

(Let the hash of a message m be $h(m)$)

1. **Deterministic:** No matter how many times you parse a particular input m through a hash function you will always get the **same result** $h(m)$.
2. **Compression:** For any size input m , the **output length** $h(m)$ should be **small**.
3. **Efficiency/Quick Computation:** For any m , hash function should be capable of **returning** the hash $h(m)$ of an input **quickly**.
4. **One-wayness /Pre-Image Resistance:** Given $h(m)$, there is **no way to find an m** . It should be difficult to invert the hashes.
5. **The Avalanche Effect:** Even if you **make a small change** in your input m the **changes** that will be **reflected** in the hash $h(m)$ will be **huge**.
6. **Collision Resistant:** Given **two different inputs** m and n where $h(m)$ and $h(n)$ are their respective hashes, it is infeasible for $h(m)$ to be **equal** to $h(n)$. It is computationally infeasible to find two values that hash to the same thing ie strong collision resistance .



Applications

- **Password Hashing:** Email provider does not save your password rather, they run the password through a hashing algorithm and save the hash of your password. Every time you attempt to sign in to your email, the email provider hashes the password you enter and compares this hash to the hash it has saved. Only when the two hashes match are you authorized to access your email.
- **Bitcoin blockchain,** 'mining' is essentially conducted by running a series of SHA-256 hashing functions. In cryptocurrency blockchains today, hashing is used to write new transactions, timestamp them, and ultimately to add a reference to them in the previous block.
- **Download Security :** To download the latest version of the Firefox browser from a site other than Mozilla's, since it isn't being hosted on a site you've learned to trust, you'd like to make sure that the installation file you just downloaded is exactly the same as the one Mozilla offers. Using a checksum calculator, you compute a checksum using a particular cryptographic hash function, such as SHA-256, and then compare that to the one published on Mozilla's site. If they're equal, you can be reasonably sure that the download you have is the one Mozilla intended you to have.
- **Message fingerprint**
 - Save the message digest of the data on a tamper-proof backing store
 - Periodically re-compute the digest of the data to ensure it is not changed.
- **Improving signature efficiency**
 - Compute a message digest (using a hash function) and sign that.



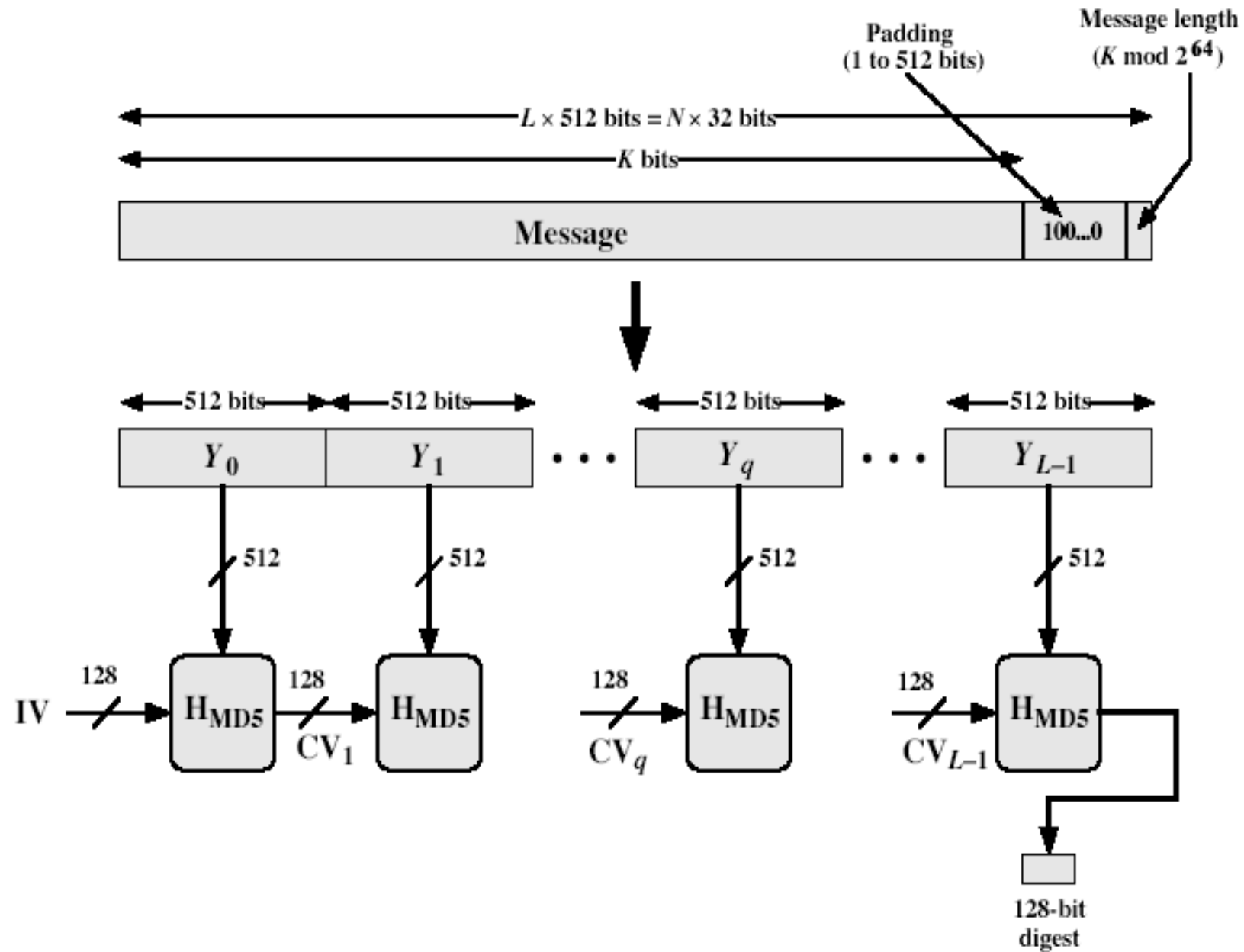
MD-5

- MD5 is the abbreviation of '**Message-Digest algorithm 5**'.
- Designed by **Ronald Rivest** in 1991 to replace an earlier hash function MD4.
- Processes **input** data in **512-bit blocks**.
- **MD5** generated message **digest** of **128 bits**.
- The MD5 algorithm is used as an encryption or fingerprint function for a file.
- Often used to encrypt database passwords, MD5 is also able to generate a file thumbprint to ensure that a file is identical after a transfer .
- An MD5 hash is composed of 32 hexadecimal characters.

PlainText: **hacker**

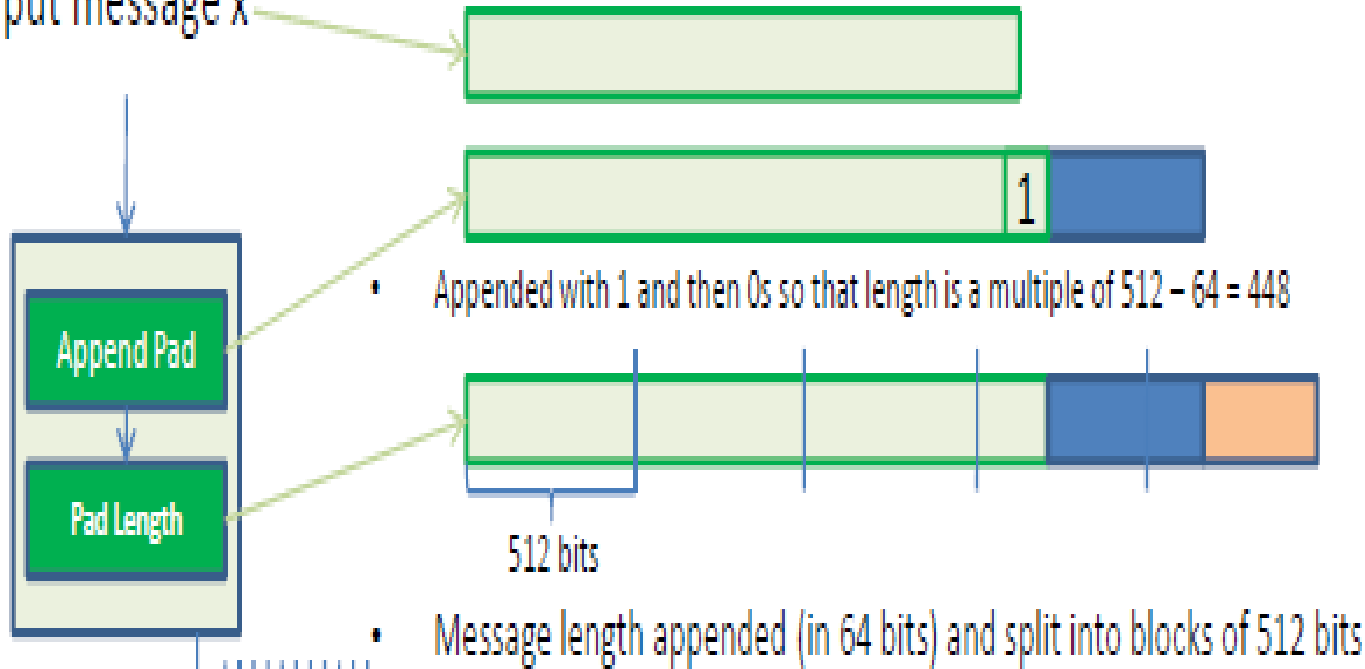
Encrypted hash: **d6a6bc0db10694a2d90e3a69648f3a03lt**

MD5 Overview



How Message Digest-5 works?

input message x



Step 1 Append Padding Bits:

- Bits are padded to make length of original message equal to 64 bit less than exact multiple of 512. ($512 * x - 64$ ie 448,960,1472)

Eg: message= 1000 bits

- Padding bits are single 1 followed by as many 0's
- Padding is always added even if length of original message equal to 64 bit less than exact multiple of 512.

Step 2 Append Length

- Calculate original length of the message excluding padding bits and add it to end of the message as a 64-bit value.
- If the length of the message is greater than 2^{64} , only the low-order 64 bits will be used.
- The resulting message has a length that is an exact multiple

How MD5 works?

Step 3 Forming Input Blocks

- Divide the input message into **blocks** of 512 bits each. Divide it into 16 sub-blocks, denoted as $M_0 \dots M_{15}$. Each sub-block will contain 32-bits.

Step 4 Initializing Chaining Variables

- 4 Chaining variables say A,B,C and D are initialized. Each **of it** is a 32 bit number as shown below.

word A: 01 23 45 67

word B: 89 AB CD EF

word C: FE DC BA 98

word D: 76 54 32 10

Copy chaining variables into 4 corresponding variables a, b, c and d.

MD5 Steps

Step 5 Process message in 16-word blocks. **4 rounds with 16 iterations**(16-sub blocks of messages) in each round.

$$b = b + ((a + \text{Aux Fun}(B, C, D) + M[i] + T[k]) \ll s)$$

1. Four auxiliary functions that take input last three(b,c and d) 32-bit words to produce a 32-bit word output per round.

Round	Iteration	Function
1	(1....16)	$F(B, C, D) = (B \text{ and } C) \text{ or } (\text{not}(B) \text{ and } D)$
2	(17...32)	$G(B, C, D) = (B \text{ and } D) \text{ or } (C \text{ and } \text{not}(D))$
3	(33...48)	$H(B, C, D) = B \text{ xor } C \text{ xor } D$
4	(49...64)	$I(B, C, D) = C \text{ xor } (B \text{ or } \text{not}(D))$

2. Variable **a** is **added** to the 32- bit output.

3. Message sub-block **M[i]** which is 32-bit is **added to output**

4. A constant **t[k]** 32 bit is **added to output**

5. Output is **circular-left shifted** by **s-bits**

6. Variable **b** is **added to output**

7. Resultant **output** becomes **input for next round**.

T[k] is an array of constants derived from sin value **containing 64 elements** where **each element is 32 bit** denoted as **t[1], t[2]...t[64]**. In **4 rounds** we use **16 out of 64 values** for each round.

(b) Table T, constructed from the sine function

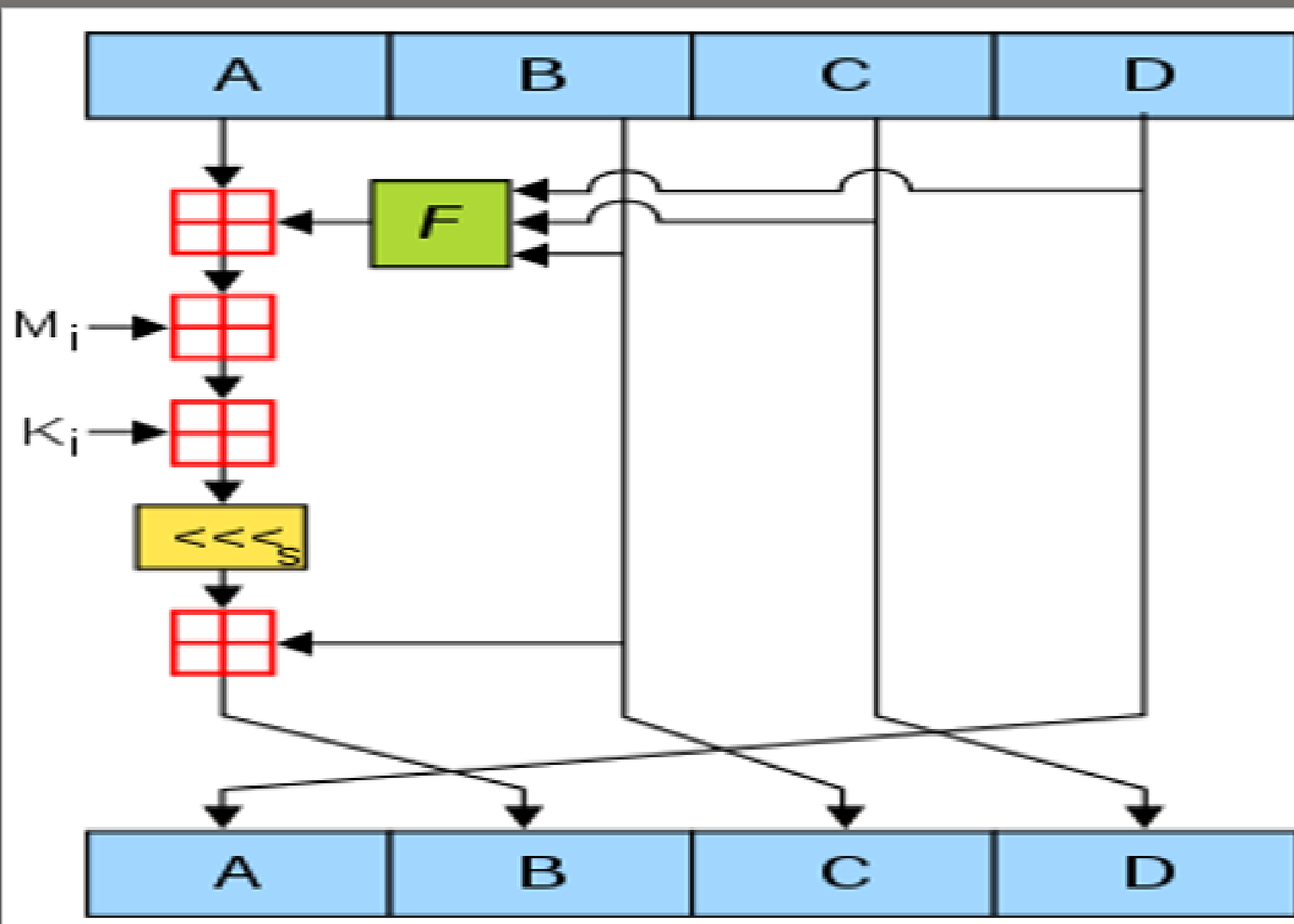
T[1] = D76AA478	T[17] = F61E2562	T[33] = FFFA3942	T[49] = F4292244
T[2] = E8C7B736	T[18] = C040B340	T[34] = 8771F681	T[50] = 432AFF97
T[3] = 242070DB	T[19] = 265E3A51	T[35] = 699D6122	T[51] = AB9423A7
T[4] = C1BDCEEE	T[20] = E9B6C7AA	T[36] = FDE5380C	T[52] = FC93A039
T[5] = F57C0FAF	T[21] = D62F105D	T[37] = A4BEEA44	T[53] = 655B59C3
T[6] = 4787C62A	T[22] = 02441453	T[38] = 4BDECFA9	T[54] = 8F0CCC92
T[7] = A8304613	T[23] = D8A1E681	T[39] = F6BB4B60	T[55] = FFEFF47D
T[8] = FD469501	T[24] = E7D3FBC8	T[40] = BEBFBC70	T[56] = 85845DD1
T[9] = 698098D8	T[25] = 21E1CDE6	T[41] = 289B7EC6	T[57] = 6FA87E4F
T[10] = 8B44F7AF	T[26] = C33707D6	T[42] = EAA127FA	T[58] = FE2CE6E0
T[11] = FFFF5BB1	T[27] = F4D50D87	T[43] = D4EF3085	T[59] = A3014314
T[12] = 895CD7BE	T[28] = 455A14ED	T[44] = 04881D05	T[60] = 4E0811A1
T[13] = 6B901122	T[29] = A9E3E905	T[45] = D9D4D039	T[61] = F7537E82
T[14] = FD987193	T[30] = FCEFA3F8	T[46] = E6DB99E5	T[62] = BD3AF235
T[15] = A679438E	T[31] = 676F02D9	T[47] = 1FA27CF8	T[63] = 2AD7D2BB
T[16] = 49B40821	T[32] = 8D2A4C8A	T[48] = C4AC5665	T[64] = EB86D391

Table T
constructed
from Sin
function

TRUTH
TABLE FROM
ROUND
FUNCTION

(a) Truth table of logical functions

b	c	d	F	G	H	I
0	0	0	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	0	1	1
1	0	1	0	1	0	1
1	1	0	1	1	0	0
1	1	1	1	1	1	0



One MD5
Operation

MD5 PROCESS FOR A SINGLE 512 BIT BLOCK

16 Sub-
blocks, each
of size 32 bit

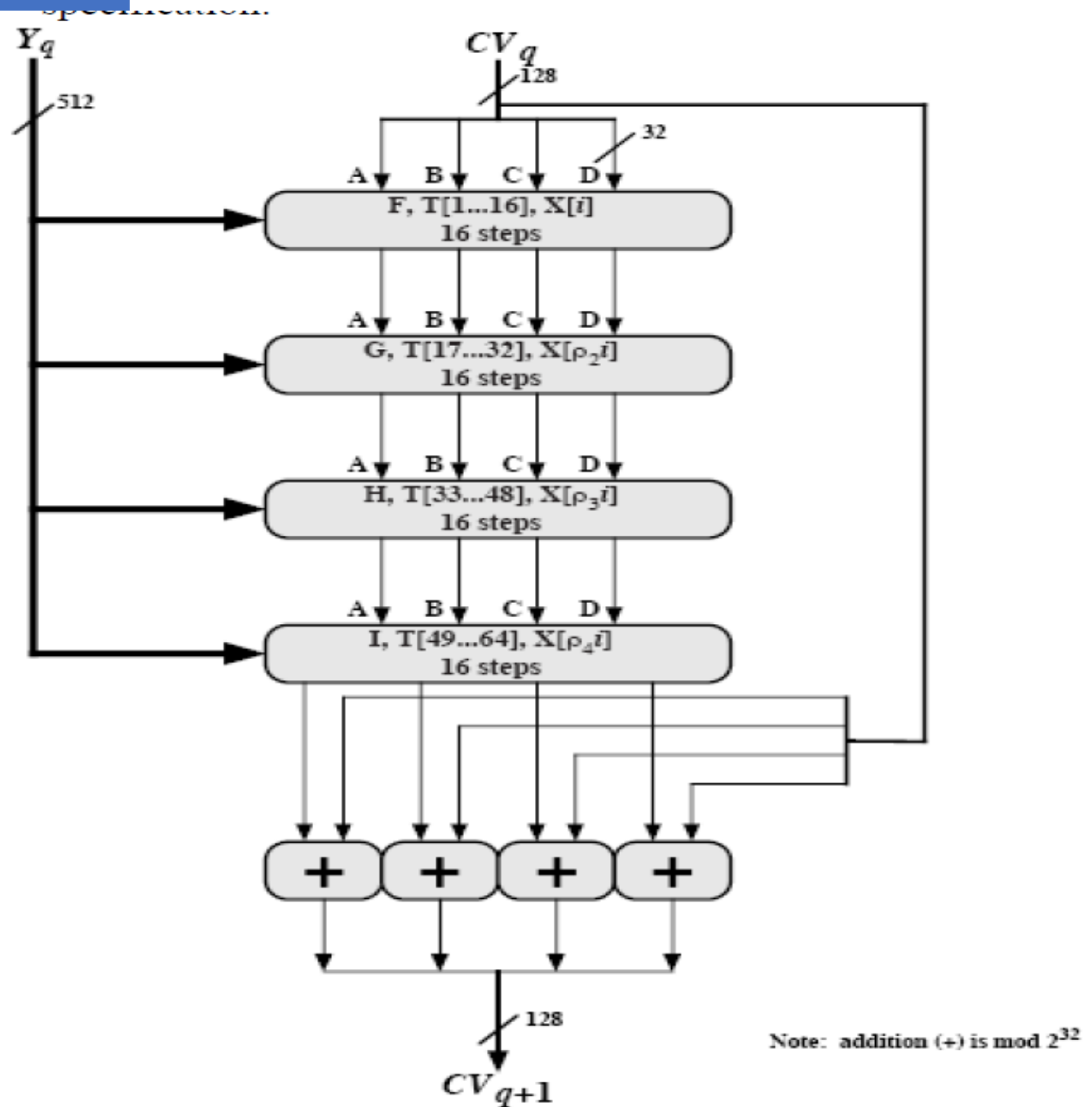


Figure 12.2 MD5 Processing of a Single 512-bit Block



SHA-1

SHA-1(Secure Hash Algorithm 1)

- **Security Hash Algorithm (SHA)**, a popular message compress standard was developed in 1993 by the **National Institute of Standards and Technology (NIST)** and **National Security Agency (NSA)**.
- Processes input data in **512-bit blocks(16 32-bit words)** and generates a **160-bit (5 words of 32 bits)** hash value called as **message digest** by applying **5 x 16 rounds per block**.
- SHA-1 is now considered insecure since 2005.
- To generate the final output, SHA-1 core block occupy **80 clock cycles**.
- Its much similar to MD5.

SHA-1 Process

Step 1 Append Padding Bits:

- Bits are padded to make length of original message equal to 64 bit less than exact multiple of 512. (512*x-64 ie It will be coming like 448,960,1472)
- Padding bits are single 1 followed by as many 0's
- Padding is always added even if length of original message equal to 64 bit less than exact multiple of 512.

Step 2 Append Length

- Calculate original length of the message excluding padding bits and add it to end of the message as a 64-bit value.
- The resulting message has a length that is an exact multiple of 512 bits.

Step 3 Forming Input Blocks

- Divide the input message into **blocks** of 512 bits each. Divide it into 16 sub-blocks, denoted as $M_0 \dots M_{15}$. Each sub-block will contain 32-bits.

SHA-1 Process

Step 4 Initializing Chaining Variables

- 5 Chaining variables say A,B,C,D and E are initialized. Each **of it** is a 32 bit number as shown below. (5*32=160bit o/p)

word A: 01 23 45 67

word B: 89 AB CD EF

word C: FE DC BA 98

word D: 76 54 32 10

Word E: C3 D2 E1 F0

Copy chaining variables into 4 corresponding variables a,b, c, d and e.

- **Step 5 consists of 4 rounds with each round consisting of 20 steps.(total 80)** Each round takes 512 bit block, register a,b,c,d,e and a constant $K[t]$ (where $t=0$ to 79 and 1 to 19 for each round)

$abcde = (e + \text{Process } P + S^5(a) + W[t] + K[t]), a, S^{30}(b), c, d)$

$W[t]$ is a 32 bit word calculated as follows:

First 16 words of $M[t]$ are copied to $W[t]$ and remaining 64 values are derived as follows.

$W[t] = S^1(W[t-16] \text{ XOR } W[t-14] \text{ XOR } W[t-8] \text{ XOR } W[t-3])$ where S^1 indicates circular left shift by 1 position

SHA-1 Process

Process P is logical operation defines as follows:

Round Function

1(1-20) (b and c) or (not(b) and d)

2(21-40) b xor c xor d

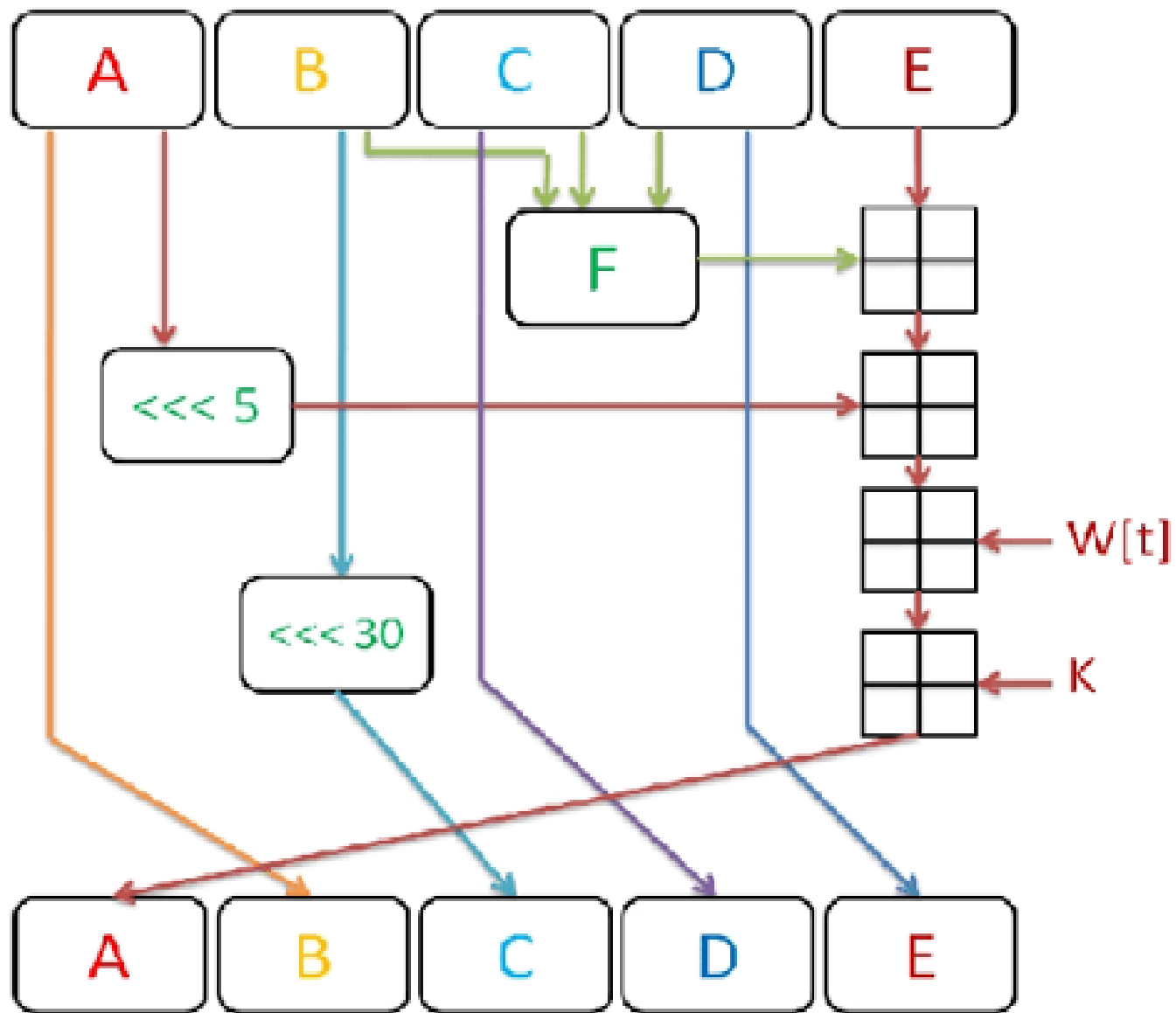
3(41-60) (b and c) or (b and d) or (c and d)

4(61-80) b xor c xor d

K[t] has four constants defined for each round as follows:

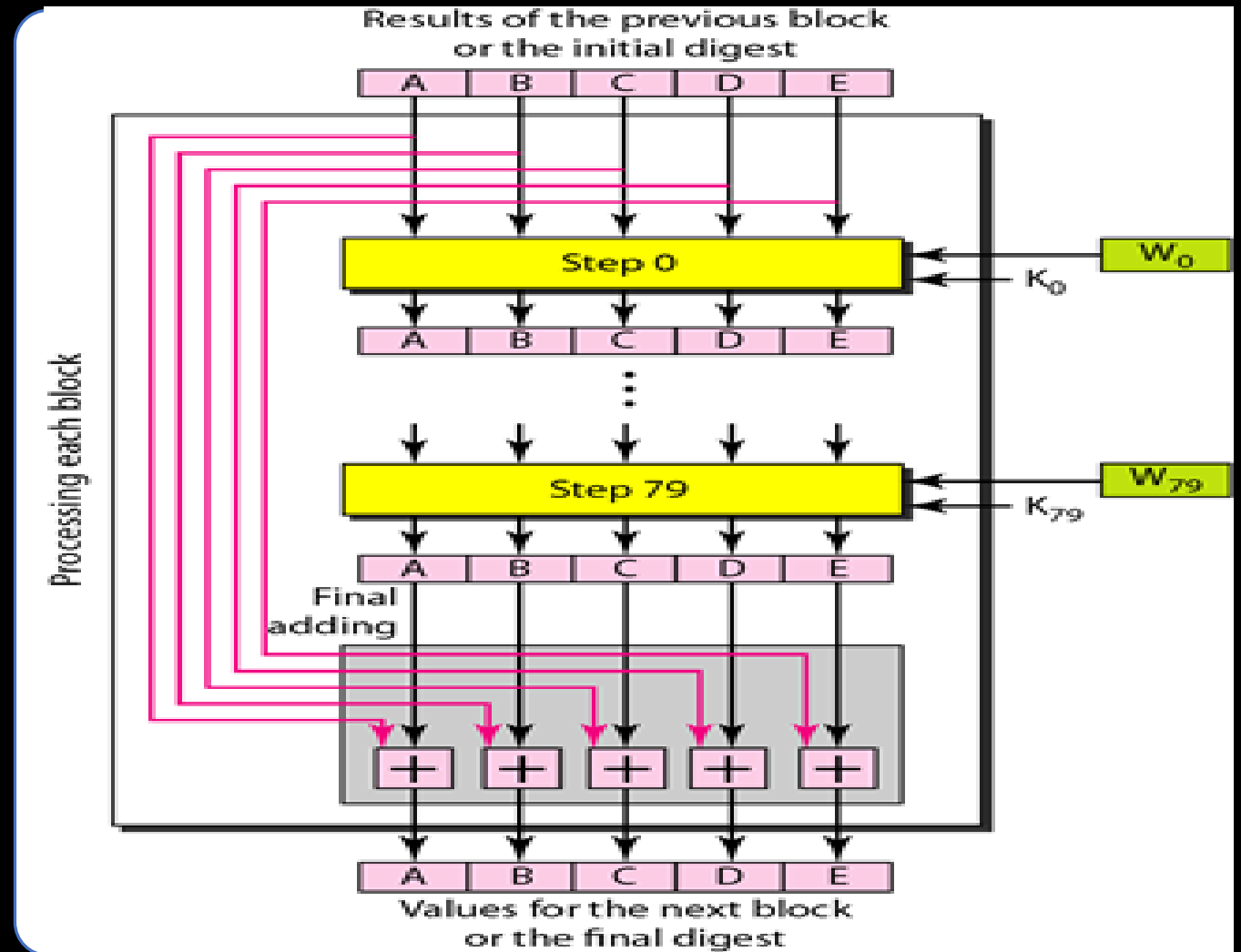
Table 1: Four Rounds Hash Algorithm

Step Number	Hexadecimal	Take integer part of
$0 \leq t \leq 19$	$K_t = 5A827999$	$[2^{30} \times \sqrt{2}]$
$20 \leq t \leq 39$	$K_t = 6ED9EBA1$	$[2^{30} \times \sqrt{3}]$
$40 \leq t \leq 59$	$K_t = 8F1BBCDC$	$[2^{30} \times \sqrt{5}]$
$60 \leq t \leq 79$	$K_t = CA62C1D6$	$[2^{30} \times \sqrt{10}]$

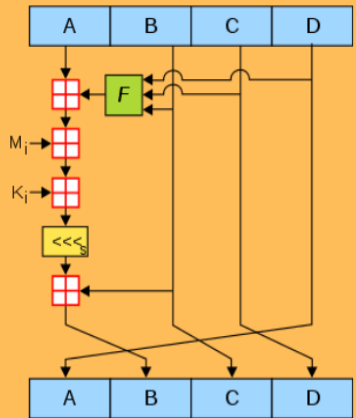


Single SHA-1
iteration

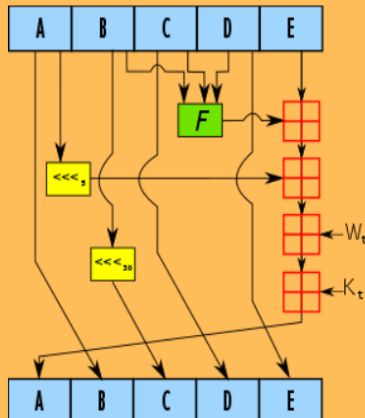
Processing of one block in SHA-1



Comparison Of MD5 and SHA-1




MD5



SHA-1

	MD5	SHA-1
Message -Digest	128	160
To Break original message from MD	2^{128} operations	2^{160} operations
If 2 messages MD are same	2^{60} operations	2^{80} operations
Speed	Faster	Slower
Software Implemenation	Simple	Simple
Attacks	Attempts to compromise some extent	No claims



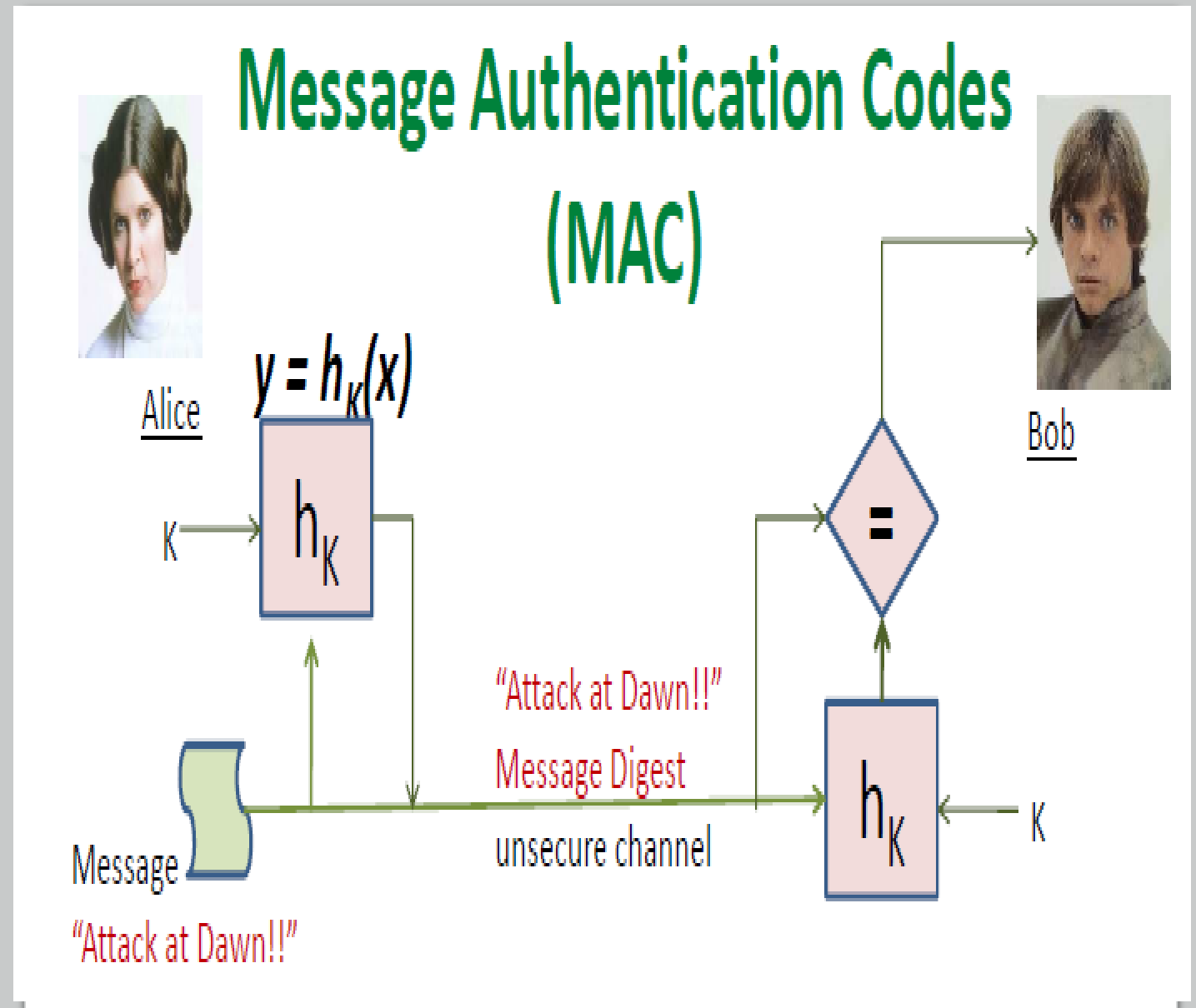
MAC

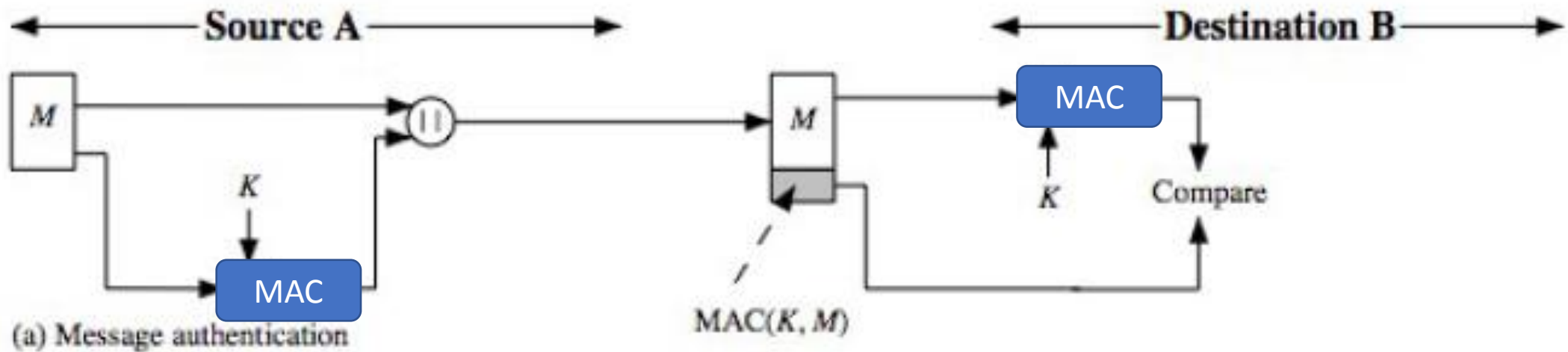
Message Authentication Code

MAC

MAC(Message Authentication Code)

- MACs allow the **message** and the **digest** to be sent over an **insecure channel**
- Similar to message-digest but with involvement of **cryptographic shared symmetric key** between Alice and Bob.
- Provides **Authenticity** and **Integrity**.
- **Integrity:** Messages are not tampered
- **Authenticity:** Bob can verify that the message came from Alice
- Does not provide non-repudiation



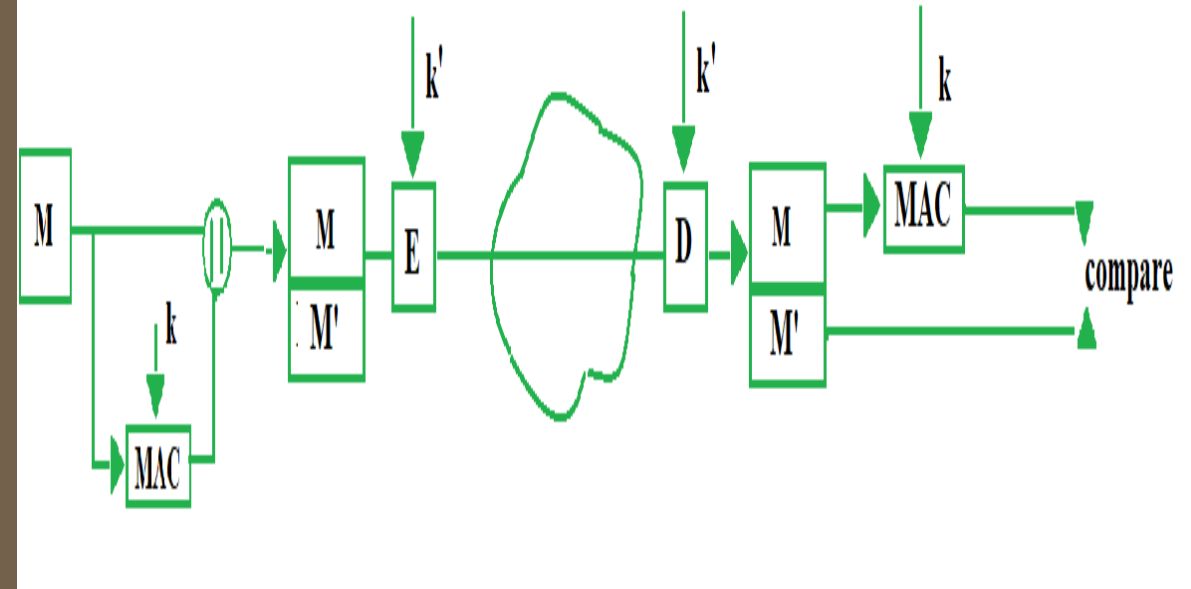
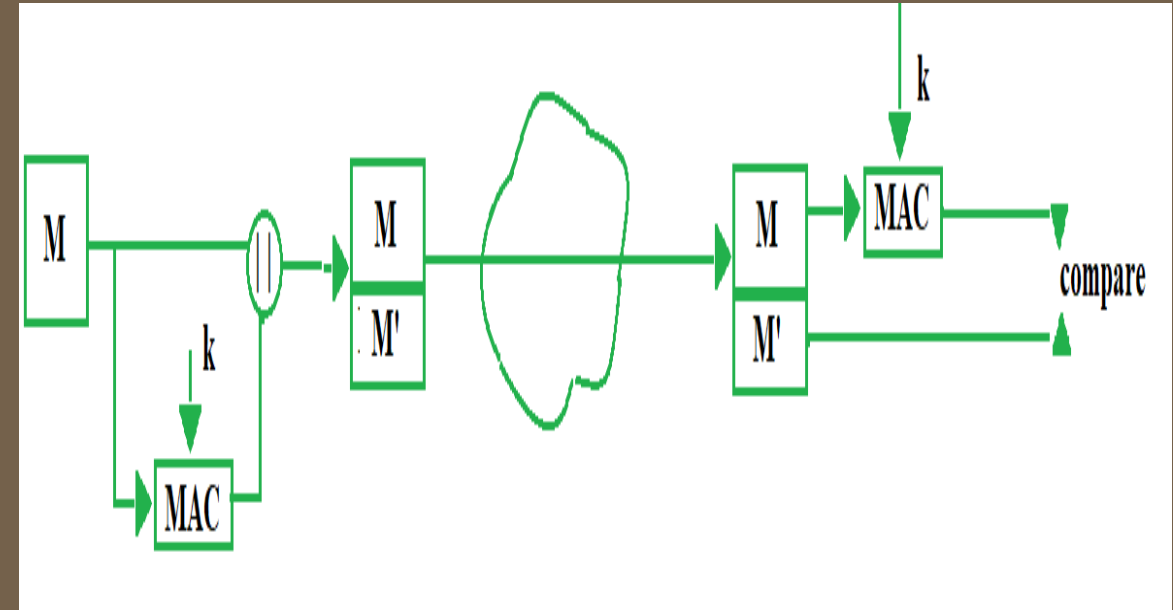


MAC WORKING

- MAC is also known as **cryptographic checksum**, is an **authentication technique** involves the use of a secret key to generate a small fixed-size block of data.
- When A has a message to send to B, it calculates the MAC as a function of the message and the key: $MAC = C(M, K)$. It is appended to the original message and sent.
- On receiver's side, receiver also generates the code and compares it with what he/she received thus ensuring the originality of the message.

Types Of Message Authentication Code (MAC)

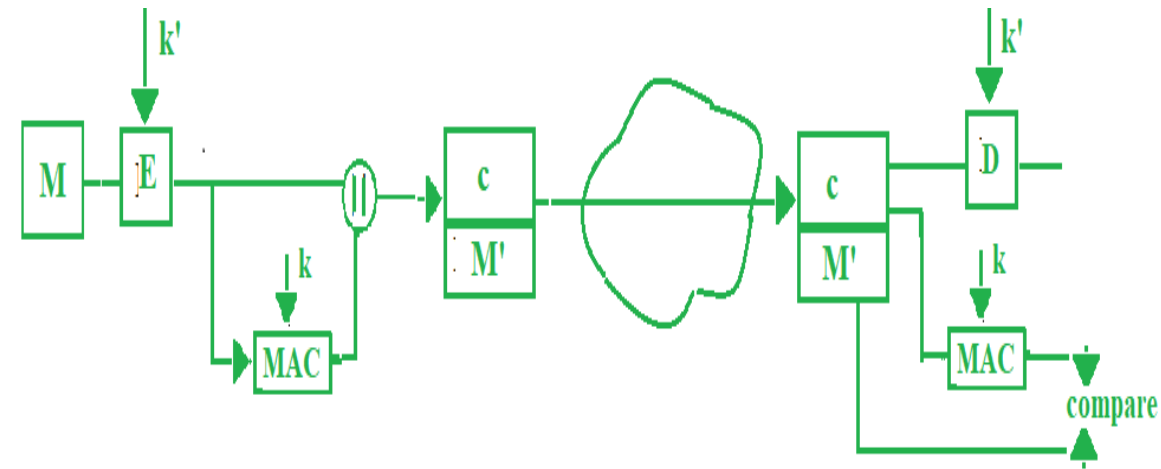
1. **MAC without encryption** : This model can provide authentication but not confidentiality as anyone can see the message.
2. **Internal Error Code**: In this model of MAC, sender encrypts the content before sending it through network for confidentiality. Thus this model provides confidentiality as well as authentication



Types Of Message Authentication Code (MAC)

3. External Error Code: First apply MAC on the encrypted message 'c' and compare it with received MAC value on the receiver's side and then decrypt 'c' if they both are same, else we simply discard the content received. Thus it saves time. $c = E(M, k')$

$$M' = \text{MAC}(c, k)$$





HASH-BASED MESSAGE AUTHENTICATION CODE

HMAC

HMAC

- **Keyed-Hashing for Message Authentication**, a variation on the MAC algorithm, has emerged as an Internet standard for a variety of applications.
- It makes use of existing Message digest algorithms such as SHA-512 ,MD5 etc which doesn't have the involvement of a shared secret key.
- HMAC algorithm uses one-way hash functions to produce unique mac values.
- The main objective of HMAC is to use Hash Functions that are available, without modification
- It is a mandatory security implementation for **Internet Protocol(IP) Security** and **SSL protocol**.

HMAC Algorithm

$$\text{MAC}(\text{text})_t = \text{HMAC}(K, \text{text})_t = H((K0 \oplus \text{opad}) || H((K0 \oplus \text{ipad}) || \text{text}))_t$$

Step 1: Make length of shared symmetric key (K) equal to that of no: of bits(b) in each block.

1.1 If $K < b$: Expand Key K to make it equal to b by appending as many 0 bits to left of K.

Eg: If $K=170$ bits and $b=512$.

1.2 If $K = b$: No action

1.3 If $K > b$: Trim K to make it equivalent to b by passing it through message digests algorithms.

Step 2($K0 \oplus \text{ipad} = S1$): XOR $K0$ (output of step1) with ipad (A string $00110110 = (0x36)_{16}$ repeated $b/8$ times) to produce a variable called S1.

Step 3($S1 || \text{text}$): Append M the original message to S1

Step 4($H(S1 || \text{text})$): Apply the selected Message digest algorithm to the output of step 3.

Step 5($K0 \oplus \text{opad} = S2$): XOR $K0$ (output of step1) with opad (A string $01011010 = (0x5C)_{16}$ repeated $b/8$ times) to produce a variable called S2.

Step 6($H(S1 || \text{text}) || S2$): Append output of step 4 to S2

Step 7($H(H(S1 || \text{text}) || S2)$): Apply the selected Message digest algorithm to the output of step 6.

Steps 1-3:

Determine K_0

Step 4:

$K_0 \oplus \text{ipad}$

Step 5:

$K_0 \oplus \text{ipad} \parallel \text{text}$

Step 6:

$H(K_0 \oplus \text{ipad} \parallel \text{text})$

Step 7:

$K_0 \oplus \text{opad}$

Step 8:

$K_0 \oplus \text{opad} \parallel H(K_0 \oplus \text{ipad} \parallel \text{text})$

Step 9:

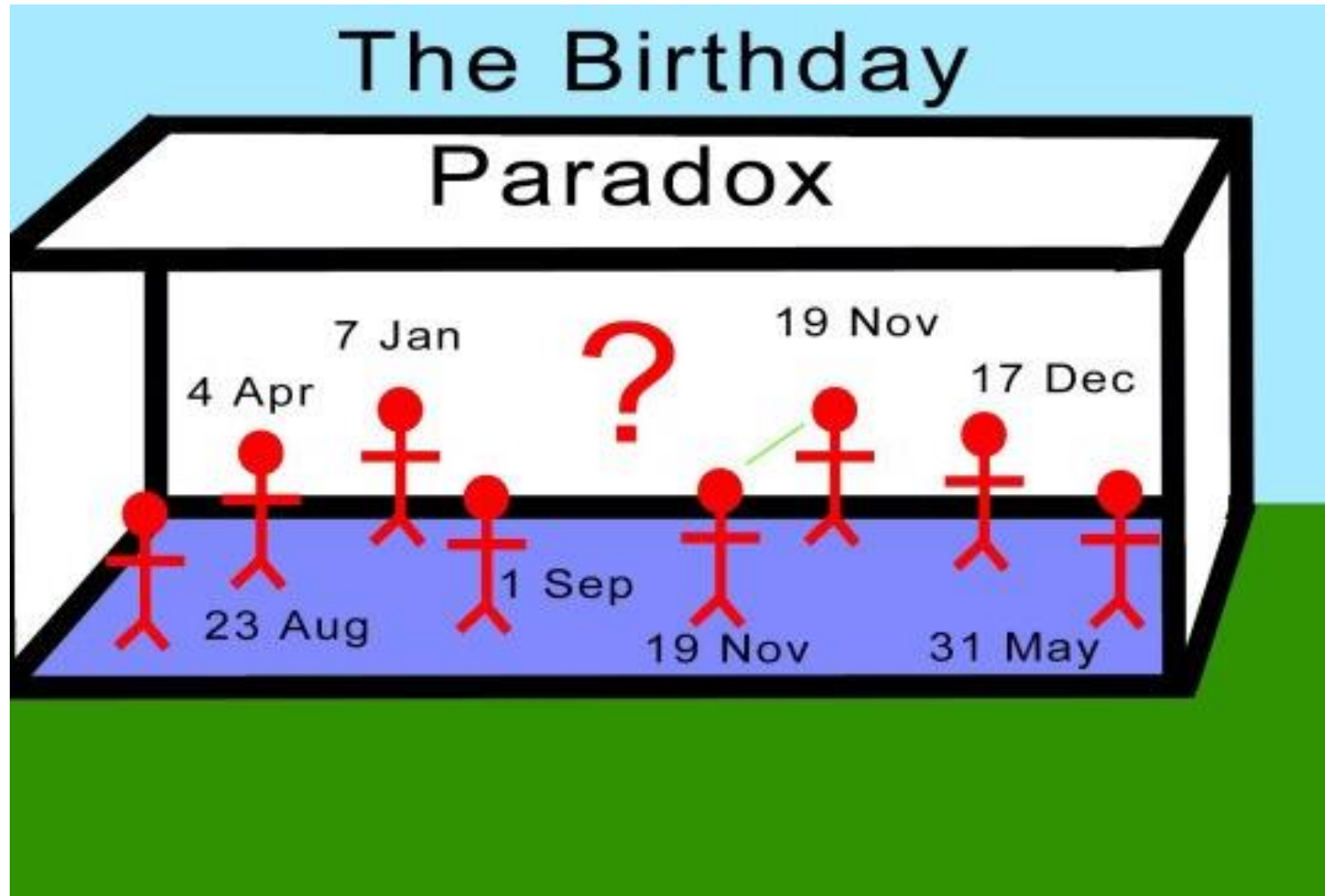
$H(K_0 \oplus \text{opad} \parallel H(K_0 \oplus \text{ipad} \parallel \text{text}))$

Step 10:

$\text{MAC}(\text{text})_t = \text{leftmost 't' bytes of}$
 $H(K_0 \oplus \text{opad} \parallel H(K_0 \oplus \text{ipad} \parallel \text{text}))$

HMAC
CONSTRUCTION

Figure 1: Illustration of the HMAC Construction



Birthday Puzzle
Suppose there are N people in a room. How large must N be before we expect two or more people will have the same birthday?

Birthday Puzzle

- **What is the probability that two persons among n have same birthday?**
- Find the probability that at least two people in a room have the same birthday

Event A: at least two people in the room have the same birthday

Event A': no two people in the room have the same birthday

$$\Pr[A] = 1 - \Pr[A']$$

$$\Pr[A'] = 1 \times \left(1 - \frac{1}{365}\right) \times \left(1 - \frac{2}{365}\right) \times \left(1 - \frac{3}{365}\right) \dots \left(1 - \frac{Q-1}{365}\right)$$

$$= \prod_{i=1}^{Q-1} \left(1 - \frac{i}{365}\right)$$

$$\Pr[A] = 1 - \prod_{i=1}^{Q-1} \left(1 - \frac{i}{365}\right)$$

Birthday Puzzle

- *How many people must be there in a room to make the probability 100% that at-least two people in the room have same birthday?*
- Answer: **367** (since there are 366 possible birthdays, including February 29)
- *How many people must be there in a room to make the probability 50% that at-least two people in the room have same birthday?*
 - ***P(different) can be written as $1 \times (364/365) \times (363/365) \times (362/365) \times \dots \times (1 - (n-1)/365)$***
 - For ***N*** people is: ***$P(N) = [365 \times 364 \times \dots \times (365-N+1)] / 365^N$ where $1 - P(N) = 0.5$. On solving it we get ***N = 23***. Formula for A-choose-B is $A! / [B! (A - B)!]$***
 - For ***N*** people is: ***$P(N) = [365 \times 364 \times \dots \times (365-N+1)] / 365^N$ where $1 - P(N) = 0.5$. On solving it we get ***N = 23***. Formula for A-choose-B is $A! / [B! (A - B)!]$***
 - . The probability that no two people share a birthday is: $(364/365)^{N(N-1)/2}$
 - So the probability that any two people do share one is: $P(N) = 1 - (364/365)^{N(N-1)/2}$
 - $N(N-1)/2 \approx N^2/2$
 - Since there are only 365 different possible birthdays, we expect to find a match, roughly, when $N^2/2 = 365$, when $N = \sqrt{730} \approx 27$
- ***P(same) = 1 - P(different)***

Collisions in Birthdays to Collisions in Hash Functions

If there are 23 people in a room, then the probability that two birthdays collide is 1/2

$$\text{Success Probability } (\varepsilon) \text{ is } \varepsilon = 1 - \prod_{i=1}^{Q-1} \left(1 - \frac{i}{M}\right) \quad |Y| = M$$

Relationship between Q, M, and success

$$Q \approx \sqrt{2M \ln \frac{1}{1-\varepsilon}}$$

Q always proportional to square root of M.

ε only affects the constant factor

$$\text{If } \varepsilon = 0.5 \text{ then } Q \approx 1.17\sqrt{M}$$

Birthday Attacks and Message Digests

- $Q \approx 1.17\sqrt{M}$
- If the size of a message digest is 40 bits then $M = 2^{40}$
- A birthday attack would require 220 queries
- Thus to achieve 128 bit security against collision attacks, hashes of length at-least 256 is required
- a **128-bit hash** which has **2^{128} different possibilities**. By using the birthday paradox, you have a **50% chance to break** the collision resistance at the **$\sqrt{2^{128}} = 2^{64}$ th** instance.
- A hash function $h(x)$ produces an output that is N bits long has 2^N different possible hash values.
- For a good cryptographic hash function, all output values are (more or less) equally likely. Then, since $\sqrt{2} = 2^{1/2}$, the birthday problem immediately implies that if we hash about **$2^{N/2}$** different inputs, we can expect to find a collision, that is, we expect to find two inputs that hash to the same value

Birthday Attack

- **Birthday attack** works thus:
 - Opponent generates $2^{m/2}$ variations of a valid message all with essentially the same meaning
 - Opponent also generates $2^{m/2}$ variations of a desired fake message
 - Two sets of messages are compared to find pair with same hash (probability > 0.5 by birthday paradox)
 - Have user sign the valid message, then substitute the forgery which will have a valid signature
- conclusion is **that need to use larger MACs**
- **No hash function is collision free**, but it usually takes so long to find a collision. Birthday Attack is used to detect collisions in message digest algm

anooja@somaiya.edu

Thank You
