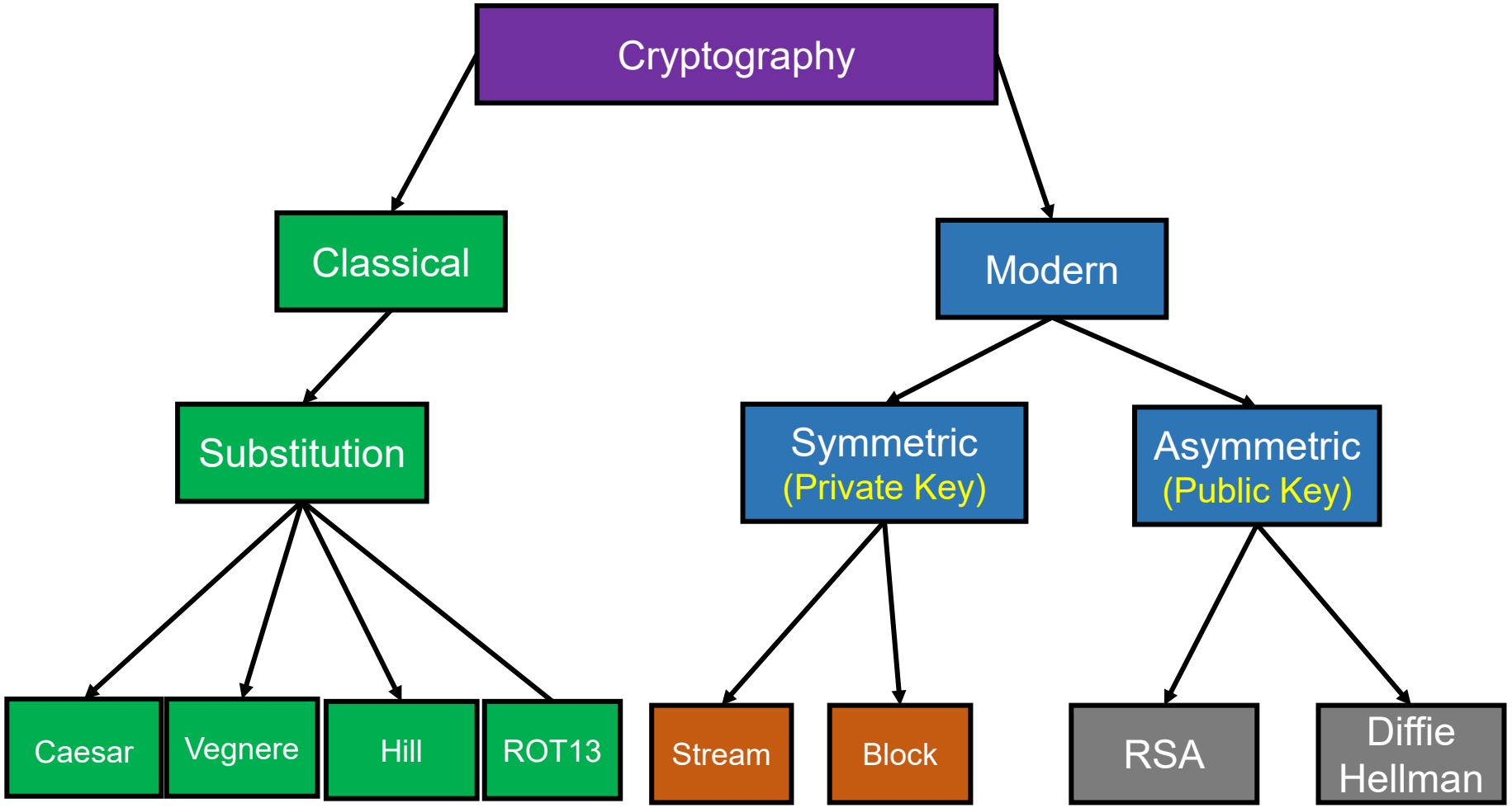
Hashing and Digital signatures

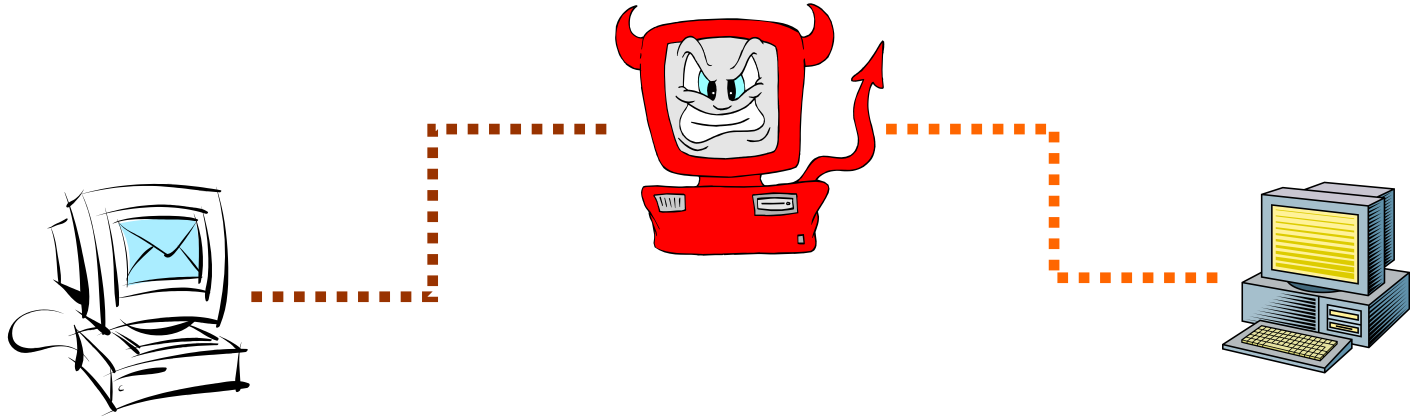
Dr Aftab Ali

COM398

What we have covered



Data Integrity and Source Authentication



- Encryption does not protect data from modification by another party.
  - Why?
- Need a way to ensure that data arrives at destination in its original form as sent by the sender and it is coming from an authenticated source.

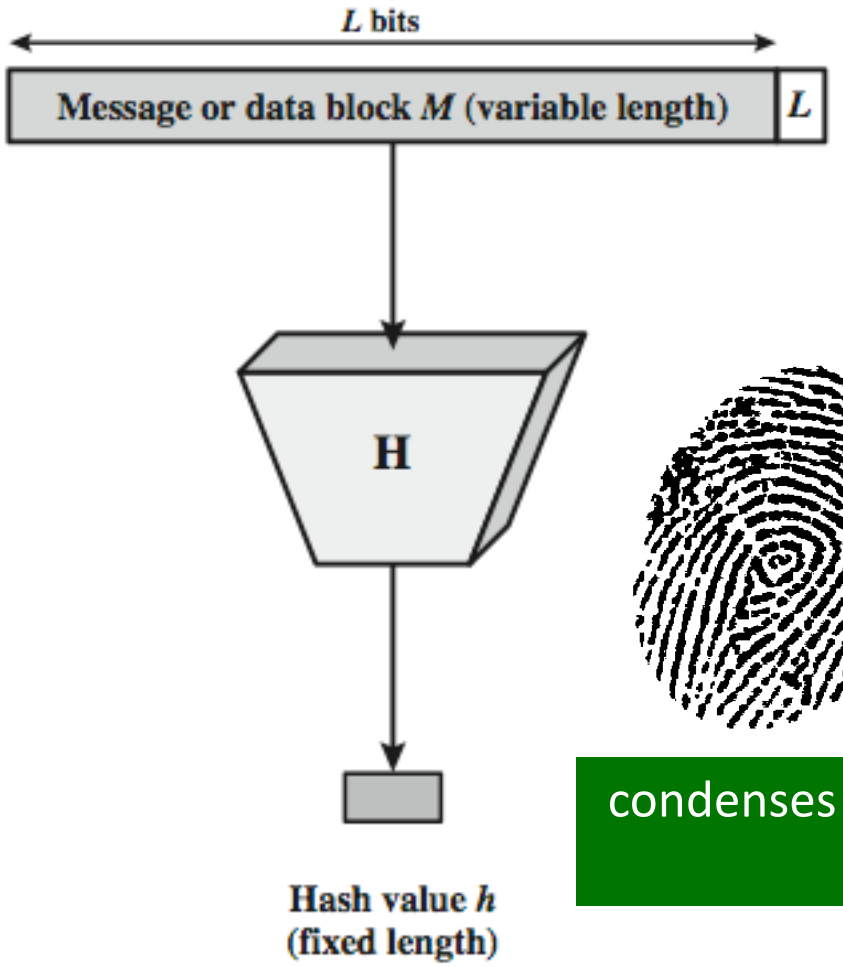
Hash Functions

- A hash function maps a message of an arbitrary length to a m-bit output
  - output known as the fingerprint or the message digest

- What is an example of hash functions?
  - Give a hash function that maps Strings to integers in [0,2^{32}-1]

- Cryptographic hash functions are hash functions with additional security requirements

Hash Functions



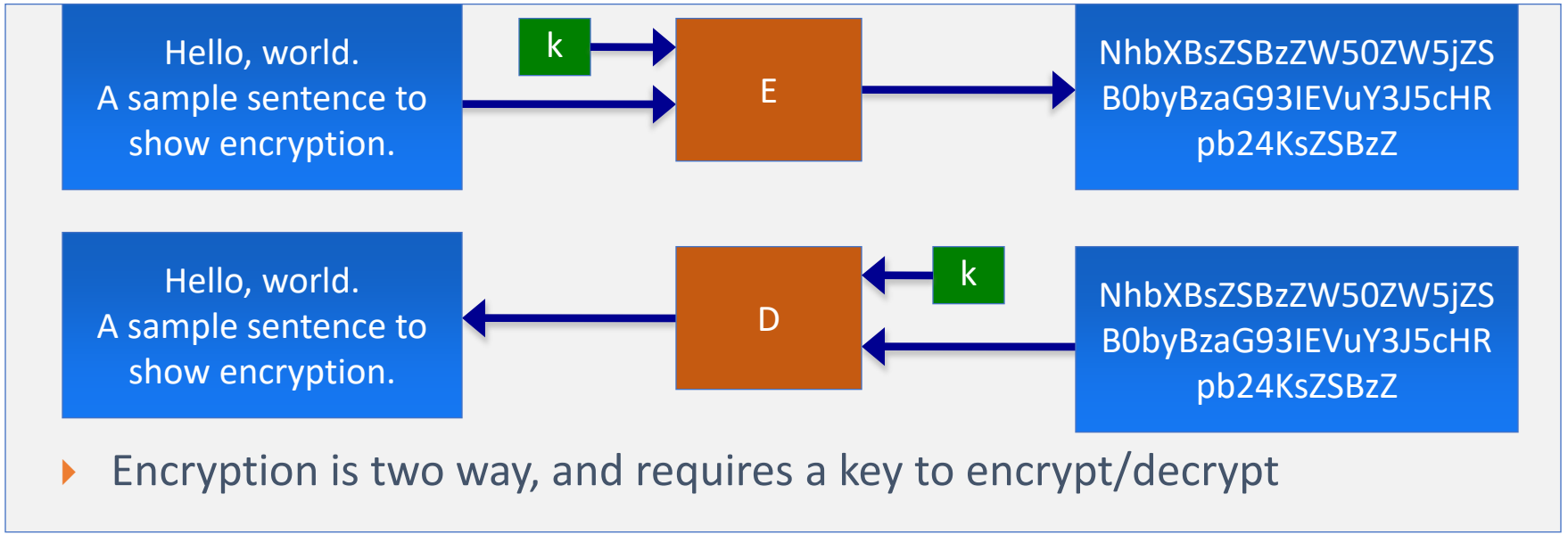
- The hash value represents concisely the longer message
  - may called the *message digest*

- A message digest is as a ``digital fingerprint'' of the original document

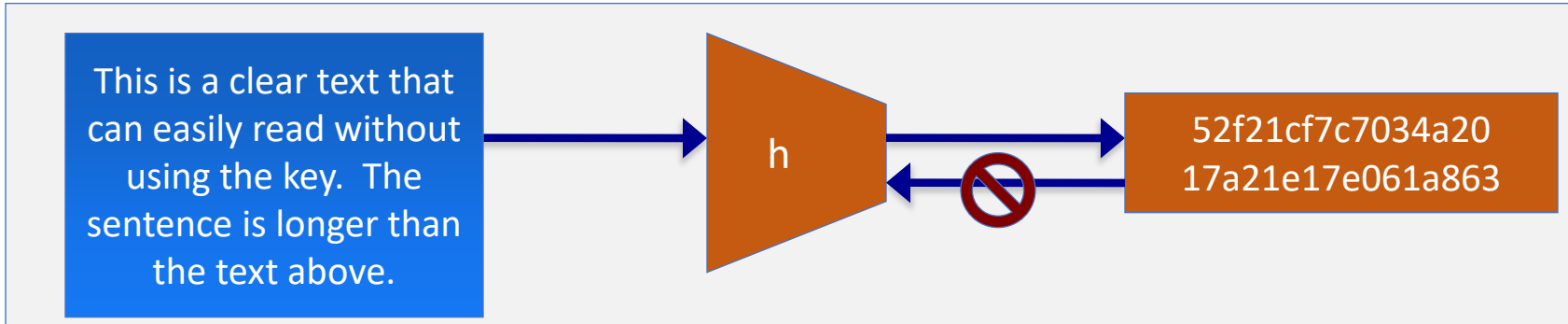
condenses arbitrary message to fixed size

$$h = H(M)$$

Hashing V.S. Encryption



- Encryption is two way, and requires a key to encrypt/decrypt



- Hashing is one-way. There is no 'de-hashing'

Hash Family

- A hash family is a four-tuple (X,Y,K,H), where
  - X is a set of possible messages
  - Y is a finite set of possible message digests
  - K is the keyspace
  - For each $K \in K$, there is a hash function $h_K \in H$. Each h_K: $X \rightarrow Y$
- Alternatively, one can think of H as a function $K \times X \rightarrow Y$

Hash Functions Family

- MD (Message Digest)
  - Designed by Ron Rivest
  - Family: MD2, MD4, MD5
- SHA (Secure Hash Algorithm)
  - Designed by NIST
  - Family: SHA-0, SHA-1, and SHA-2
    - SHA-2: SHA-224, SHA-256, SHA-384, SHA-512
    - SHA-3: New standard in competition
- RIPEMD (Race Integrity Primitive Evaluation Message Digest)
  - Developed by Katholieke University Leuven Team
  - Family : RIPEMD-128, RIPEMD-160, RIPEMD-256, RIPEMD-320

MD5, SHA-1, and RIPEMD-160

	MD5	SHA-1	RIPEMD-160
Digest length	128 bits	160 bits	160 bits
Basic unit of processing	512 bits	512 bits	512 bits
Number of steps	64 (4 rounds of 16)	80 (4 rounds of 20)	160 (5 paired rounds of 16)
Maximum message size	∞	$2^{64} - 1$ bits	$2^{64} - 1$ bits
Primitive logical functions	4	4	5
Additive constants used	64	4	9
Endianness	Little-endian	Big-endian	Little-endian

Cryptographic Hash Function Properties

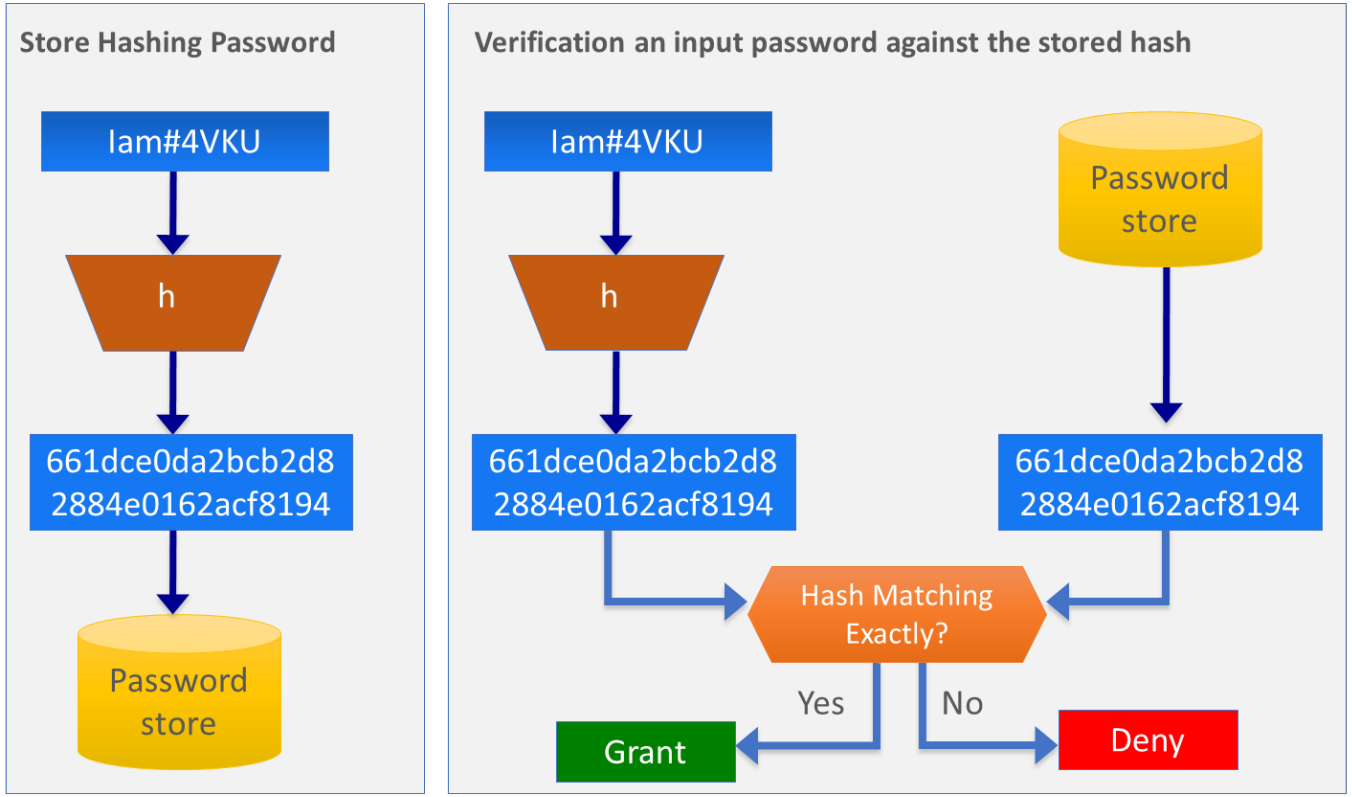
- There are several requirements the hash function has to meet to be considered secure.
- *Speed*
  - *That means the hashing function should be able to produce a hash in a fraction of a second.*
- Avalanche Effect
  - *Hi 3639efcd08abb273b1619e82e78c29a7df02c1051b1820e99fc395dcaa3326b8*
  - *hi 8f434346648f6b96df89dda901c5176b10a6d83961dd3c1ac88b59b2dc327aa4*
- Hash Function Should Be Deterministic
  - Same input, will always result in the same output
- Pre-Image Resistance (One-Way Function)
  - Can't reverse the cryptographic hash function
- Collision Resistance
  - *two different messages shouldn't be able to produce the same hash value.*

Usages of Cryptographic Hash Functions

- Software integrity
  - E.g., tripwire
- Timestamping
  - How to prove that you have discovered a secret on an earlier date without disclosing it?
- Covered later
  - Message authentication
  - One-time passwords
  - Digital signature

Examples of Cryptographic Hash Functions

- Password Verification
  - Nearly all operating systems and websites store passwords in hashes



Examples of Cryptographic Hash Functions

- Cryptocurrencies
  - Cryptographic hash functions are widely used in cryptocurrencies to pass transaction information anonymously.
  - For example, Bitcoin, the original and largest cryptocurrency, uses the SHA-256 cryptographic hash function in its algorithm.
- Signature Generation and Verification
  - Verifying signatures is a mathematical process used to verify the authenticity of digital documents or messages
- Verifying File and Message Integrity

Using Hash Functions for Message Integrity

- Uses a Hash Function h, assuming an authentic (adversary cannot modify) channel for short messages
  - Transmit a message M over the normal (insecure) channel
  - Transmit the message digest h(M) over the secure channel
  - When receiver receives both M' and h, how does the receiver check to make sure the message has not been modified?

- This is insecure. How to attack it?
- A hash function is a many-to-one function, so collisions can happen.

Message Authentication Code

- A MAC scheme is a hash family, used for message authentication
- MAC(K,M) = $H_K(M)$
- The sender and the receiver share secret K
- The sender sends (M, $H_k(M)$)
- The receiver receives (X,Y) and verifies that $H_K(X)=Y$, if so, then accepts the message as from the sender
- To be secure, an adversary shouldn't be able to come up with (X',Y') such that $H_K(X')=Y'$.

Security Requirements for MAC

- Resist the Existential Forgery under Chosen Plaintext Attack
  - Challenger chooses a random key K
  - Adversary chooses a number of messages M_1, M_2, .., M_n, and obtains t_j=MAC(K,M_j) for $1 \leq j \leq n$
  - Adversary outputs M' and t'
  - Adversary wins if $\forall j$ M'$\neq M_j$, and t'=MAC(K,M')

- Basically, adversary cannot create the MAC for a message for which it hasn't seen an MAC

Constructing MAC from Hash Functions

- Let h be a one-way hash function

- MAC(K,M) = h(K || M), where || denote concatenation
  - Insecure as MAC
  - Because of the Merkle-Damgard construction for hash functions, given M and t=h(K || M), adversary can compute M'=M||Pad(M)||X and t', such that h(K||M') = t'

HMAC: Constructing MAC from Cryptographic Hash Functions

$$HMAC_K[M] = Hash[(K^+ \oplus opad) \,||\, Hash[(K^+ \oplus ipad)||M)]]$$

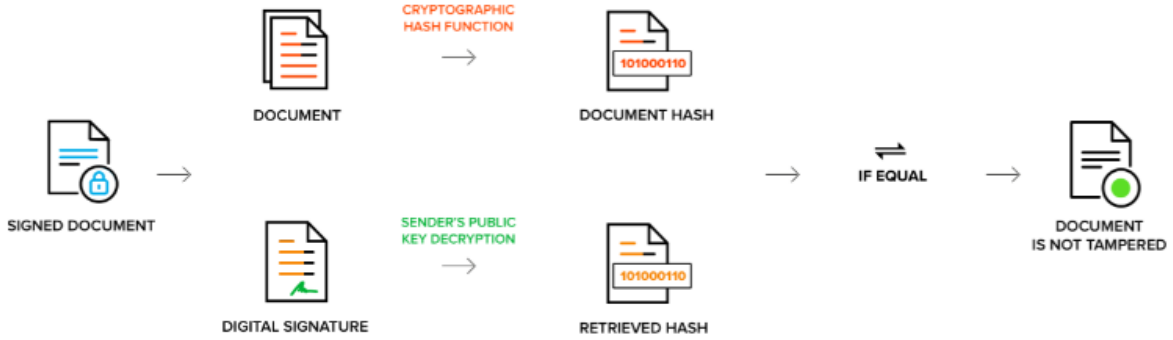
- K^+ is the key padded (with 0) to B bytes, the input block size of the hash function
- ipad = the byte 0x36 repeated B times
- opad = the byte 0x5C repeated B times.

At high level, $HMAC_K[M] = H(K \,||\, H(K \,||\, M))$

Digital Signature

1. Digital Signature is a type of electronic signature that encrypts documents with digital codes that are particularly difficult to duplicate

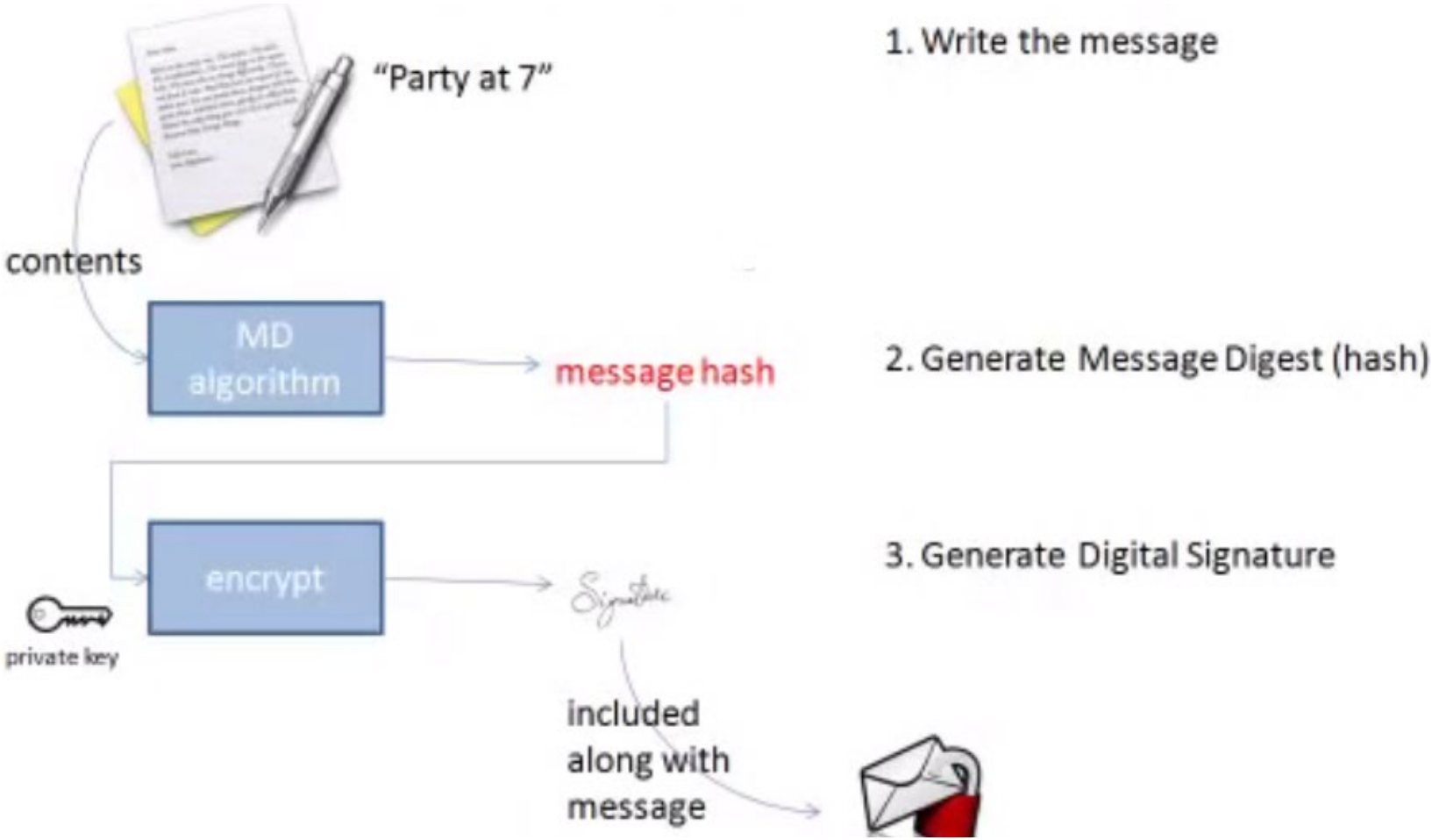
2. In the case of a digital signature message is encrypted with the private key and decrypted with the public key.



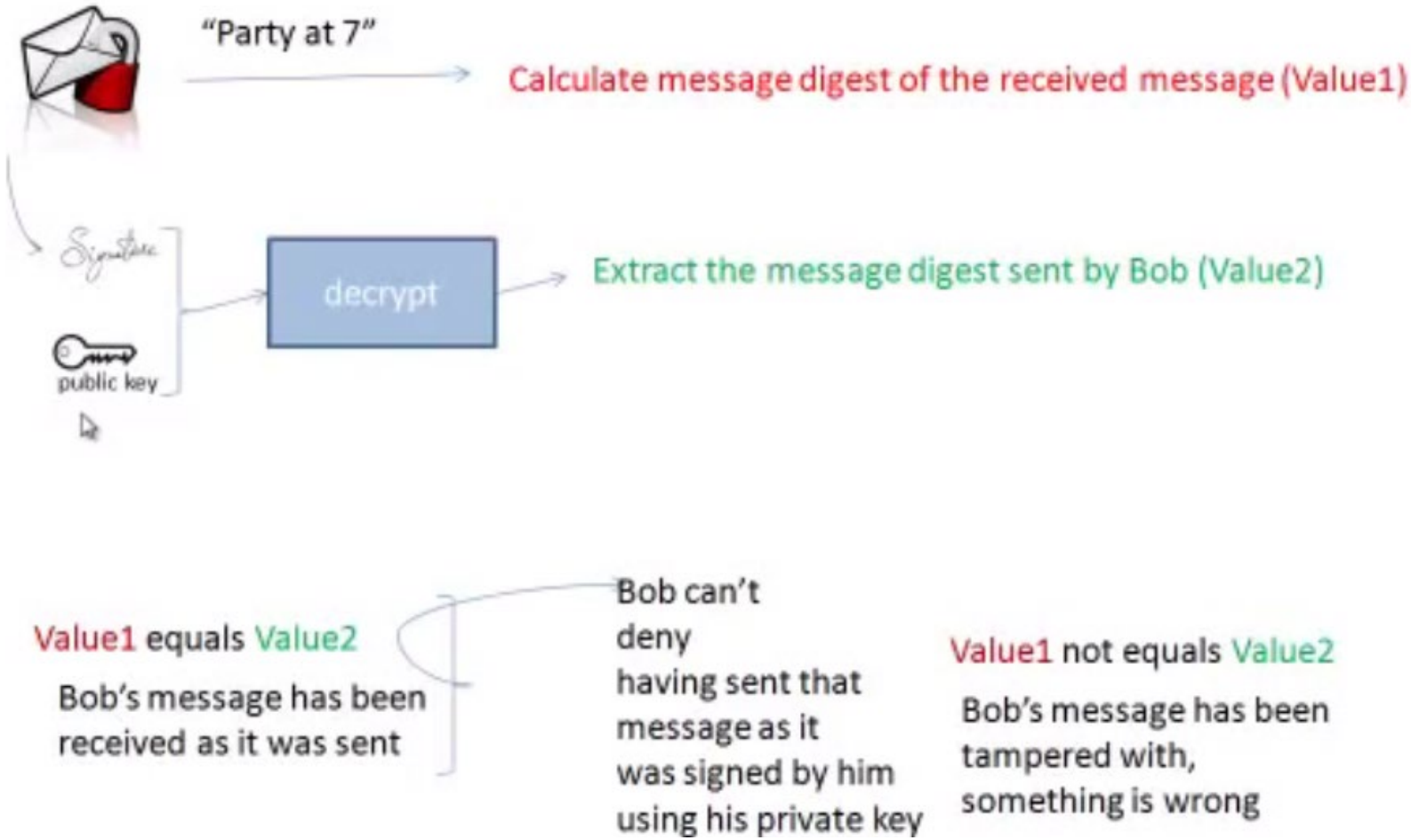
Example


BOB
FRANK
Bob wants to send an important message to Frank
private key
public key
public key
DEVIL

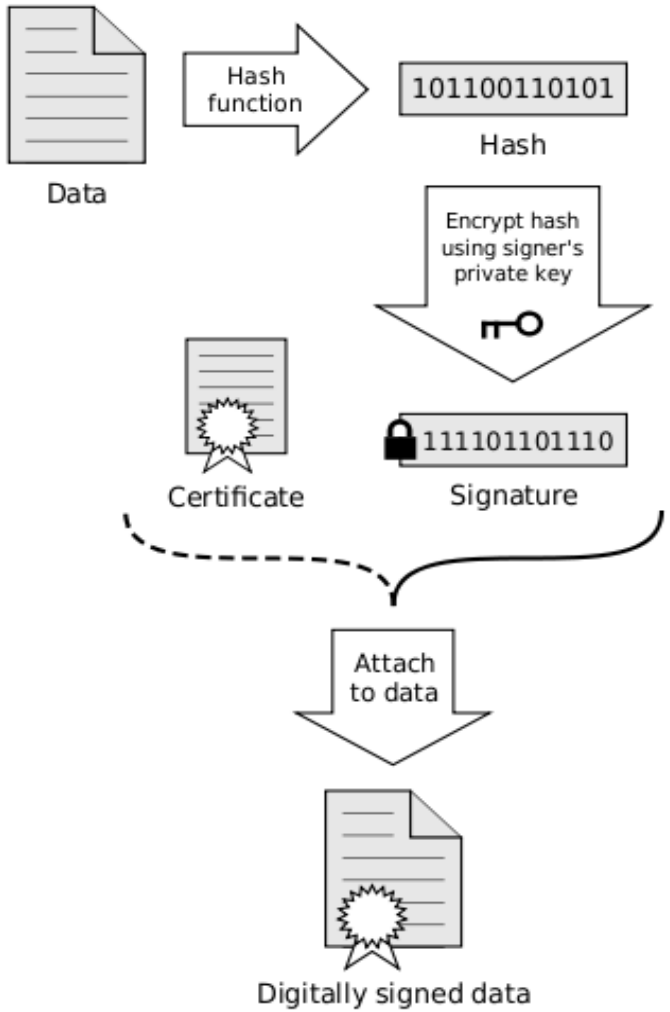
On BOB Side


"Party at 7"
contents
MD algorithm
message hash
encrypt
private key
included along with message
1. Write the message
2. Generate Message Digest (hash)
3. Generate Digital Signature

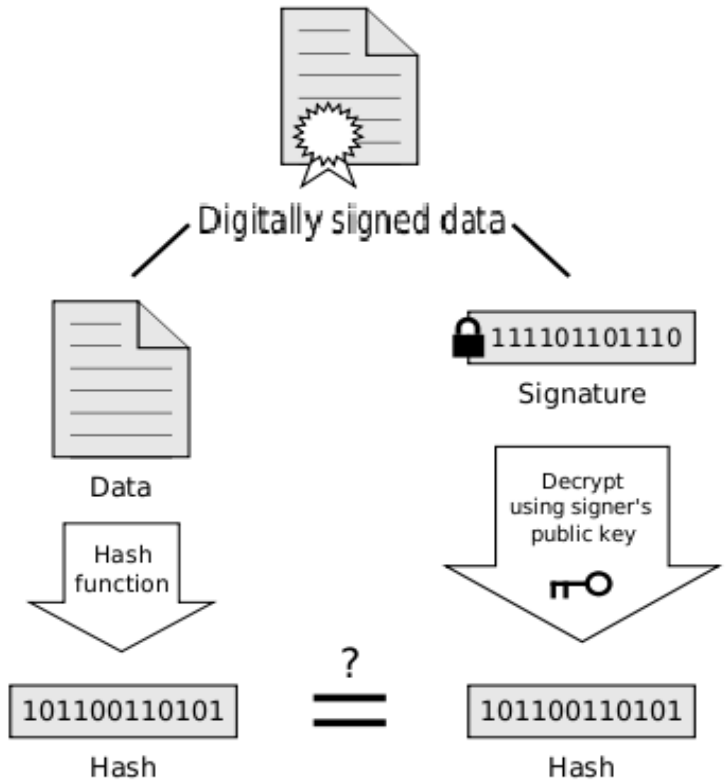
On FRANK side



Signing



Verification



If the hashes are equal, the signature is valid.

Advantages

- Authentication:
  - Identification of person sign.
- Integrity of data:
  - Every change is detected.
- Non Repudiation:
  - Author has encrypted sign on message.
- Speed:
  - Contracts are easily written, completed, and signed by all concerned parties in a little amount of time no matter how far the parties are geographically.

Example

Suppose Alice and Bob are exchanging information and Bob wants to send a signed, but not encrypted message to Alice.

Suppose Bob's RSA setup is as follows:

$$\text{Bob's Public: } E_B(x) = x^5 \pmod{91}$$
$$\text{Bob's Secret: } D_B(y) = y^{29} \pmod{91}$$
$$\text{Bob's Message: } M = 30$$

To securely ***sign*** the message, Bob needs to make a calculation that only he could do. Thus Bob uses his ***secret*** key to sign the message.

$$\text{Bob's signature: } \sigma = D_B(M) = 30^{29} \bmod 91 = 88$$

He sends the message-signature pair to Alice, $(M, \sigma) = (30, 88)$.

Alice can now use Bob's ***public*** information to ***verify*** that this message came from Bob.

$$\text{Alice's verification: } E_B(\sigma) = \sigma^5 = 88^5 \equiv 30 \equiv M \pmod{91}$$

Since $E_B(\sigma) \equiv M \pmod{n}$, Alice has verified that the message is from Bob. This works because $E_B(\sigma) = (\sigma)^5 = (M^{29})^5 = M \pmod{91}$ by Euler's Theorem.