Cybersecurity: Authentication and Digital Signatures

Ozalp Babaoglu

ALMA MATER STUDIORUM – UNIVERSITA' DI BOLOGNA

Prologue

- In the beginning, the main goal of cryptography was *confidentiality*
- In modern usage, especially over public networks like the Internet, we need to add new properties
  - *Integrity*
  - *Authentication*
  - *Digital signatures*

Prologue

- ***Integrity***: the receiver of a message must be able to verify that the *content* of the received message corresponds to that of the sent message
- ***Authentication***: the receiver of a message must be able to verify the *identity* of the sender
- ***Digital signature***: composite property that is necessary when the sender and receiver of a message are mutually non trusting — property that is similar to a paper-and-pen signature

Hash functions

$f: X \mapsto Y$

$|X| = n, \; |Y| = m, \quad n \gg m$

given $x \in X, \quad y = f(x) \in Y$

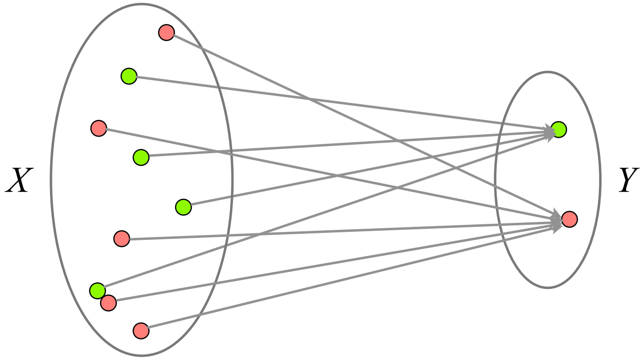
- f: hash function
- x: pre-digest
- y: hash value (also known as *digest*)
- X: domain of f
- Y: range of f
- Used in programming to implement the "dictionary" data structure for fast lookups

Properties of hash functions
Balanced

- f has to be "many-to-one" but it is "balanced"

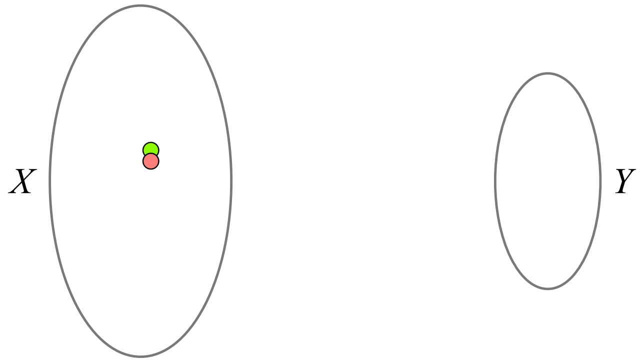
$$X_i = \{x \in X : f(x) = y_i\},\ 1 \leq i \leq m$$

$$|X_1| \approx |X_2| \approx \cdots \approx |X_m|$$



Properties of hash functions
Dispersal

- f is such that values very close together in X are mapped to values far apart in Y



Cryptographic hash functions

A *cryptographic* (or *one-way*) *hash function* is a hash function that satisfies also the following properties:

1. For any $x \in X$, it is easy to compute $f(x)$
2. For any $y \in Y$, it is computationally infeasible to find $x \in X$ such that $f(x) = y$
3. Given any x_1, it is computationally infeasible to find an x_2 different from x_1 such that $f(x_1) = f(x_2)$

Cryptographic hash functions

Consider the *8-bit block parity* hash function:

m=110100101000100111100101000101001010001000010 1

b_1=11010010
b_2=10001001
b_3=11100101
b_4=00010100
b_5=10100010
b_6=00010100
digest=00011100 (column-wise ⊕)

Cryptographic hash functions

- 8-bit block parity satisfies the *balanced* and *dispersal* properties of hash functions
- But does not satisfy the *second* and *third properties* of cryptographic hash functions
- Example (violation of property 2):
  - Given a *digest*, it is trivial to find a *pre-digest* that maps to it

```
         digest(m)=10011100
m=0101001011001001111001010101010010100010000101
```


Cryptographic hash functions

Example (violation of property 3):

$\mathtt{m}_1$=1101001010001001111001010001010010100010000101

Find an $\mathtt{m}_2$ (different from $\mathtt{m}_1$) that has the same digest as $\mathtt{m}_1$

We know what the digest of $\mathtt{m}_1$ is: digest($\mathtt{m}_1$)=00011100

We can invert any *even number* of bits in $\mathtt{m}_1$ that are in the same column and the parity will not change:

```
     11010010       11110010
     10001001       10001101
m1   11100101   m2  11000101   digest(m2)=00011100
     00010100       00110000
     10100010       10100010
     00010100       00110100
```


Cryptographic hash functions: Summary

Hash function properties:

- Arbitrary size input / Fixed-size output
- Efficiently computable
- Balanced / Dispersal

Security properties:

- (Hiding) For any $y \in Y$, it is computationally infeasible to find $x \in X$ such that $f(x) = y$
- (Collision-freedom) Given any x_1, it is computationally infeasible to find another x_2 different from x_1 such that $f(x_1) = f(x_2)$

Comments

- Collision freedom and hiding can be violated trivially through brute force
- Compute the hash of all possible values for pre-digest until you find one that produces the desired digest
- Has to be rendered computationally infeasible by making sure that domain X is very large
- Implication of collision freedom:
  - Given two *digests* $f(x_1)$ and $f(x_2)$ that are equal, then it is safe to assume that the pre-digests are equal: $x_1 = x_2$
  - In other words, the *digest* of an object can serve as a *proxy* for the object

Practical Cryptographic hash functions

- Practical examples:
  - MD2, MD4, MD5 — 128 bits
  - Snefru — 128 bits, 256 bits
  - HAVAL — Variable-size digest
  - SHA-0, SHA-1, SHA-2 — Variable-size digest
  - SHA-3 (won NIST competition in 2012 as the *Keccak* algorithm)

Cryptographic hash functions: MD5

- One of a series of algorithms originally designed by Ron Rivest
- 128-bit digest as defined in IETF RFC 132
- Example:

```
MD5("The quick brown fox jumps over the lazy dog")
  = 9e107d9d372bb6826bd81d3542a419d6

MD5("The quick brown fox jumps over the lazy fog")
  = f0f0996b26d7e959fe3652b4976fc62d
```

http://onlinemd5.com

Cryptographic hash functions: MD5

- Known to be vulnerable to certain collision attacks
- CERT Vulnerability Note VU#836068:
  - "Cryptanalytic research published in 1996 described a weakness in the MD5 algorithm that could result in collision attacks, at least in principle. Further research published in 2004 demonstrated the practical ability for an attacker to generate collisions and in 2005 the ability for an attacker to generate colliding x.509 certificates was demonstrated."

Digital Signatures

Paper-and-pen Signatures

- Only one individual can generate it
- Cannot be falsified by others
- Cannot be reused (on different documents)
- The signed document cannot be modified (after signing)
- Cannot be repudiated by the signer

Digital Signatures

- Must guarantee the same properties as a paper-and-pen signature
- Since any implementation of a digital signature after all is a *string of bits*, it can be copied/duplicated perfectly (while a paper-and-pen signature cannot)

Digital Signatures: Properties

- *Authentic*: Proof that the signer, and no one else, deliberately signed the document
- *Not reusable*: Signature part of a single document and cannot be moved to another document
- *Unalterable*: After it is signed, the document cannot be altered
- *Cannot be repudiated*: The signer cannot claim to not have signed the document

Digital Signatures: Operations

Two operations

- Sign: Generate the signature for message m by A
  - $Sign(m, A) \mapsto \sigma$
- Verify: Verify the signature σ as belonging to A
  - $Verify(\sigma, A) \mapsto \{true, false\}$

Protocol 1: Public-key (Asymmetric) Cryptography

- *A* wants to send *B* the message *m* signed with its signature

```
A: Sign
     s=D(m,kA[priv])
     send <A,m,s>
B: Verify
     receive <A,m,s>
     m*=C(s,kA[pub])
     if m*= m
         then true else false
```


Observations

- The cipher must be commutative
  $D(C(m)) = C(D(m)) = m$
- RSA *is* commutative
- The signed message is not addressed to any specific receiver
- Anyone can verify the signed message
- The message is *not* confidential since the act of verification reveals its content
- The length of the sent message is double the length of the original message

Properties of Protocol 1

- *Authentic*?
  - The signature can be generated by only one party — the one who knows $k_{A[priv]}$, in other words A
- *Not reusable*?
  - The signature cannot be copied/reused since it is a function of the corresponding message
- *Unalterable*?
  - The corresponding message cannot be modified (by anyone other than A) since doing so would require regenerating the signature
- *Cannot be repudiated*?
  - Cannot be repudiated by A since only it could have generated the signature (since only it knows $k_{A[priv]}$)

Protocol 2: Add confidentiality and specific destination

```
A: Sign
     c=C(m,kB[pub])       //encrypt
     s=D(c,kA[priv])      //sign
     send <A,c,s>
B: Verify
     c*=C(s,kA[pub])      //verify
     m*=D(c*,kB[priv])    //decrypt
     if m* makes sense
         then true else false
```

Shortcomings

- Requires the recipient to decide if the decrypted message "makes sense"

Protocol 3: Based on Cryptographic Hash Functions

Let **f()** be a cryptographic hash function

Sign:

```
s=D(f(m),kA[priv]) // sign the digest
c=C(m,kB[pub])     // encrypt
send <A,c,s>
```

Verify:

```
m*=D(c,kB[priv])       // decrypt
if f(m*)=C(s,kA[pub]) // verify
   then true else false
```


Remaining Issues

- A signed message can be *replayed* at a later time: "Transfer $100 from *A* to *B*'s account"
- Need to add *timestamps*
- How do *A* and *B* obtain each other's public keys?
- Simple minded message exchange subject to "***man-in-the-middle***" attack
- More about ***man-in-the-middle*** later

Message Authentication Codes (MAC)

- A short, fixed-length ***digest*** of the message that can be generated only by one ***specific sender***
- Can be used to ***authenticate*** the sender and verify the ***integrity*** of the message
- Obtained through a cryptographic hash function together with a secret key that is shared between the sender and receiver

MAC: Example

- Given a cryptographic hash function $f()$, we can generate the MAC of message m by applying $f()$ to the concatenation of m with a secret key k

$$MAC(m) = f(m \mid k)$$

- Sender sends the tuple: $(m, MAC(m))$
- The receiver computes the MAC of the received message m and compares it to the MAC contained in the message
- If they coincide, the receiver has *authenticated* the sender and *verified* the integrity of the message since no other party could have sent the matching tuple and the contents of the message could not have been altered

MAC using symmetric cryptography

- A and B share a secret key k

1. A sends $(m, f(m \mid k))$ to B
2. B receives (μ, ω)
3. B knows k thus can compute $f(\mu \mid k)$
4. B compares $f(\mu \mid k)$ to ω
5. If $f(\mu \mid k) = \omega$, then B concludes that $\mu = m$ (integrity) and that the sender of m was indeed A (authentication)

MAC: Comments

- It is easy to compute the MAC of a message but it is difficult to compute the message given its MAC
- Example of a "***Keyed Hash Function***"
- Similar to digital signatures but weaker since *non repudiation* is not satisfied (the destination can claim to have received any message it likes)
- Based on a shared secret key with all of its associated shortcomings