

Computer Security

CS 426

Lecture 5

Public-private cryptography

Digests, MAC, Signatures



Center for Education and Research
in Information Assurance and Security

Elisa Bertino
Purdue University
IN, USA
bertino@cs.purdue.edu

Public-private cryptography

2

Public Key Encryption

- Public-key encryption
 - each party has a PAIR (K , K^{-1}) of keys: K is the **public** key and K^{-1} is the **private** key, such that $D_{K^{-1}}[E_K[M]] = M$
 - Knowing the public-key and the cipher, it is computationally infeasible to compute the private key
 - Public-key crypto systems are thus known to be *asymmetric* crypto systems
 - The public-key K may be made publicly available, e.g., in a publicly available directory
 - Many can encrypt, only one can decrypt

3

RSA Algorithm

- Invented in **1978** by Ron Rivest, Adi Shamir and Leonard Adleman
 - Published as R L Rivest, A Shamir, L Adleman, "On Digital Signatures and Public Key Cryptosystems", Communications of the ACM, vol 21 no 2, pp120-126, Feb 1978
- Security relies on the difficulty of factoring large composite numbers
- Essentially the same algorithm was discovered in 1973 by Clifford Cocks, who works for the British intelligence

4

Key generation:

Select 2 large prime numbers of about the same size,
p and q

Compute $n = pq$, and $\Phi(n) = (q-1)(p-1)$

Select a random integer e, $1 < e < \Phi(n)$, s.t.
 $\gcd(e, \Phi(n)) = 1$

Compute d, $1 < d < \Phi(n)$ s.t. $ed \equiv 1 \pmod{\Phi(n)}$

Public key: (e, n)

Private key: d

Note: p and q must remain secret

5

Encryption

Given a message M, $0 < M < n$ $M \in \mathbb{Z}_n - \{0\}$

use public key (e, n)

compute $C = M^e \pmod{n}$ $C \in \mathbb{Z}_n - \{0\}$

Decryption

Given a ciphertext C, use private key (d)

Compute $C^d \pmod{n} = (M^e \pmod{n})^d \pmod{n} = M^{ed} \pmod{n} = M$

6

- $p = 11, q = 7, n = 77, \Phi(n) = 60$
- $d = 13, e = 37$ ($ed = 481; ed \pmod{60} = 1$)
- Let $M = 15$. Then $C \equiv M^e \pmod{n}$
 $C \equiv 15^{37} \pmod{77} = 71$
- $M \equiv C^d \pmod{n}$
 $M \equiv 71^{13} \pmod{77} = 15$

7

n, p, q

- The security of RSA depends on how large n is, which is often measured in the number of bits for n. Current recommendation is 1024 bits for n.
- p and q should have the same bit length, so for 1024 bits RSA, p and q should be about 512 bits.
- p-q should not be small

8

RSA Implementation

- Select p and q prime numbers
- In general, select numbers, then test for primality
- Many implementations use the Rabin-Miller test, (probabilistic test)



9

RSA Implementation

e

- e is usually chosen to be 3 or $2^{16} + 1 = 65537$
- In order to speed up the encryption
 - the smaller the number of 1 bits, the better



10

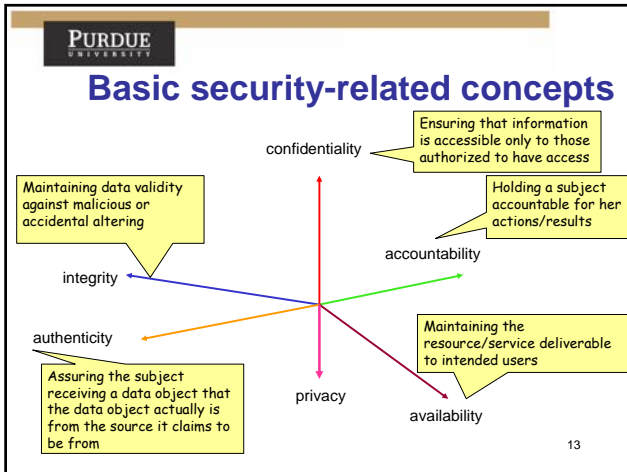
Key Recovery Attack

- One possible attack is based on factoring:
 - $n = pq$
- RSA Challenge: factoring large numbers
 - RSA-193, 640bits (193decimal digits)
 - Factored in November 2, 2005
 - RSA-200, 663bits (200 decimal digits)
 - Factored in May 9, 2005

11

Digests Message Authentication Code Signatures

12



PURDUE UNIVERSITY

Integrity

- Integrity:
 - ☞ **Maintaining data validity against malicious or accidental altering**
- Example: Alice writes a will to distribute her estate upon her death. The will does not have to be encrypted so that anybody can examine it. However its integrity must be preserved (Alice does not want the will to be changed)

14

PURDUE UNIVERSITY

Fingerprint

- To maintain integrity of the will, Alice can put her fingerprint at the bottom of the will.
- A malicious subject can create a false will, but he can not forge Alice's fingerprint.
- A third party can verify the fingerprint on the will with the fingerprint of Alice.

15

PURDUE UNIVERSITY

Message and message digest

- A Message digest (also called Message Detection Code -MDC -) is the equivalent of the fingerprint. It allows to detect if the message was changed.
- To create the message digest, an algorithm (hash function) is applied to it. The output of the function is **a string of fixed length**.

```

graph LR
    A[Message/document] --> B[Hash function]
    B --> C[Message digest]
  
```

16

The size of the digest is fixed

```

$ cat smallfile
This is a very small file with a few characters

$ cat bigfile
This is a larger file that contains more characters.
This demonstrates that no matter how big the input
stream is, the generated hash is the same size (but
of course, not the same value). If two files have
a different hash, they surely contain different data.

$ ls -l empty-file smallfile bigfile linux-kernel
-rw-r--r-- 1 steve steve 0 2004-08-20 08:58 empty-file
-rw-r--r-- 1 steve steve 48 2004-08-20 08:48 smallfile
-rw-r--r-- 1 steve steve 260 2004-08-20 08:48 bigfile
-rw-r--r-- 1 root root 1122363 2003-02-27 07:12 linux-kernel

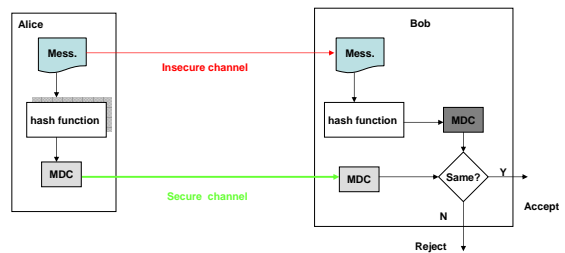
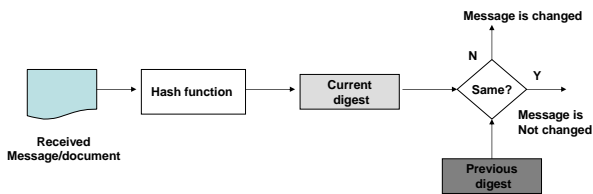
$ md5sum empty-file smallfile bigfile linux-kernel
d41d8cd98f00b204e9800998ecf8427e empty-file
75cdebfeb70a06d42210938da88c42991 smallfile
4a0b7a1676ac027913983f39b085e41a bigfile
c74c32e4d2839fa9acf0aa0c915e022 linux-kernel
    
```

Fingerprints vs. message digest

- The document and the fingerprint are physically linked together.
- The document/message and the digest can be unlinked (and sent) separately.

☞ The digest must be protected from change.

Checking message integrity



MDC: message modification code

Hash functions

- A hash function takes a long string (or 'message') of any length as input and produces a **fixed length** string as output.
- A hash functions $y=H(x)$ has the following properties
 - Deterministic:
 - two identical input must generate the same hash value (if $x_1=x_2$ then $y_1 = y_2$)
 - Minimize the likelihood of collisions
 - If $x_1 \neq x_2$ then the *probability* that $y_1=y_2$ should be low
 - A small change in the input (e.g. one bit) should cause a large change in the output (avalanche effect)
 - Easy to compute and result in good compression

Avalanche effect

- **Small changes to the message cause large difference in the corresponding digest**

```
T -> 0x54 -> 0 1 0 1 0 1 0 0
t -> 0x74 -> 0 1 1 1 0 1 0 0
```



```
$ md5sum file?
75cdbfeb70a06d42210938da88c42991  fi1e1
6fbc37f1eea0f802bd792ea885cd03e2  fi1e2
```

22

Cryptographic hash functions

- A cryptographic hash function is a hash function with certain additional properties:
 - Preimage attack resistant
 - **One-way property**
 - Second Preimage attack resistant
 - Collision resistant
- A cryptographic hash function is suitable to provide **authentication** and **message integrity**

23

Preimage resistance

- Given an hash function h and a hash value y it should be hard to find a message m such that $y=h(m)$
- This property is related to the notion of *one-way function*
- If h is not preimage resistant, then the adversary can fabricate a message m corresponding to y

24

Hash functions

- One-way (hash) functions:
 - Their computation is easy
 - *Computing the inverse function is computationally infeasible*
- One-way function example:
 - Given the integer x , it is easy to compute the function:
 - $y=f(x) = 543*x(\text{mod } 21997)$
 - EX: $x=11000$; $y= 11813$
 - but given y 11813 it is computational infeasible to compute the value of x ($x= f^{-1}(y)$)

25

Second preimage resistance

- Given an input message $m1$ and its hash value $y=h(m1)$, it should be hard to find another input message $m2$ (not equal to $m1$) such that $h(m1) = h(m2)$
- This property ensures that a message can not easily be forged
- If the cryptographic function is not second preimage resistant, an adversary
 - can intercept $m1$ and $y=h(m1)$
 - Creates another message $m2 \neq m1$ but such that $h(m1) = h(m2)$
 - Sends $m2$ and $h(m2)$ to the intended recipient

26

Collision resistance

- It should be hard to find two different messages $m1$ and $m2$ such that $h(m1) = h(m2)$

27

MD2, MD4 and MD5

- Family of one-way hash functions (Ron Rivest)
- MD2: produces a 128-bit hash value, perceived as slower and less secure than MD4 and MD5
- MD4: produces a 128-bit hash of the message, using bit operations on 32-bit operands for fast implementation, specified as Internet standard RFC1320
- MD5: produces a **128-bit output**, specified as Internet standard in RFC1321; till relatively recently was widely used.
- However, a flaw was found in the design of MD5. Thus, cryptographers began recommending the use of other algorithms, such as SHA-1 (Secure Hash Algorithm)

28

Secure Hash Algorithms

- Secure Hash Algorithm(s) are five cryptographic hash functions designed by the National Security Agency (NSA) and published by the NIST as a U.S. Federal Information Processing Standard.
- The five algorithms are denoted *SHA-1*, *SHA-224*, *SHA-256*, *SHA-384*, and *SHA-512*.
- SHA-1 produces a message digest that is 160 bits long; the number in the other four algorithms' names denote the bit length of the digest they produce
- SHA-1 is used in several protocols, including TLS and SSL, PGP, SSH, S/MIME, and IPsec.

Message Digest vs. Checksum

- A checksum is a form of redundancy check, a simple way to protect the integrity of data by detecting errors in data that are sent through space (telecommunications) or time (storage)
 - For an example of a checksum algorithm see <http://en.wikipedia.org/wiki/Checksum>
- In general, it works by adding up the basic components of a message, typically the asserted bits, and storing the resulting value
- Anyone can later perform the same operation on the data, compare the result to the authentic checksum, and (assuming that the sums match) conclude that the message was probably not corrupted
- A checksum cannot be used as a cryptographic hash function because it is not second preimage resistant
 - Given an input message $m1$ and its checksum $h(m1)$ it is easy to find another input message $m2$ (not equal to $m1$) such that $h(m1) = h(m2)$

Why Hash is Not Enough?

- Hash functions can provide data integrity, but no indication about where is data coming from or who generated the hash output:
 - Hash function is public so anybody can generate the hash output
- Message authentication is needed, otherwise anybody can inject traffic
 - Message authentication codes (MACs)
- MACs do not prevent all traffic injections (for example do not prevent replay attacks)



31

Message Authentication Code (MAC)

- Now, how can Bob:
- Verify the integrity of the message AND
- Authenticate the data origin (i.e. that the message was sent by Alice)?



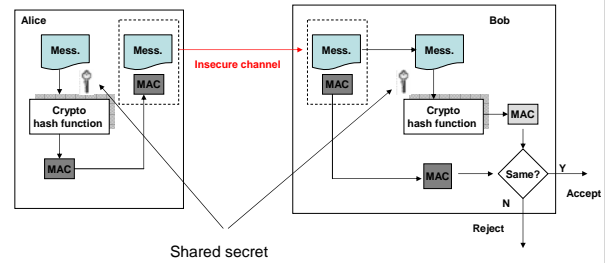
Message Authentication Code

32

MAC

- A message authentication code (MAC) is a short piece of information used to authenticate a message
- $MAC = h(\text{secret}, \text{message})$
- **The difference between a MDC and a MAC is that the MAC includes a secret () shared between the sender and the receiver (Alice and Bob)**
- The MAC value protects both a message's integrity as well as its authenticity, by allowing verifiers (who also possess the secret key) to detect any changes to the message content
- How does the MAC allow Bob to assure himself that the message M is really from Alice?
 - Alice and Bob share the secret key k
 - When Bob receives the message of Alice, Bob re-computes the MAC and confirms that the message M was from Alice

MAC



34

Security of MAC

- An attacker (Eve) intercepts the message M and the MAC $y=h(K|M)$
 - How can Eve forge a message *without knowing the secret key K*?
 - If the size of K is limited, Eve may generate and prepend all possible K to M and generate a digest until the generated digest is equal to y – the one intercepted - (exhaustive search on K)
 - The size of K is normally very large in MAC, but Eve can find a $M': y=h(M') = h(K|M)$. Then she can find K.
- ☞ **The security of a MAC depends on the security of the underlying hash algorithm**

Hashed Message Authentication Code (HMAC)

- A HMAC is a type of message authentication code (MAC) calculated using a cryptographic hash function in combination with a secret key. $HMAC = h(\text{secret key}, \text{msg})$
 - Any iterative (also called nested) cryptographic hash function, such as MD5 or SHA-1, may be used in the calculation of an HMAC
 - An iterative hash function breaks up a message into blocks of a fixed size and iterates over them with a compression function. For example, MD5 and SHA-1 operate on 512-bit blocks. The size of the output of HMAC is the same as that of the underlying hash function (128 or 160 bits in the case of MD5 and SHA-1)
- ☞ **The cryptographic strength of the HMAC depends upon the cryptographic strength of the underlying hash function, on the size and quality of the key and the size of the hash output length in bits**

What is a Digital Signature?

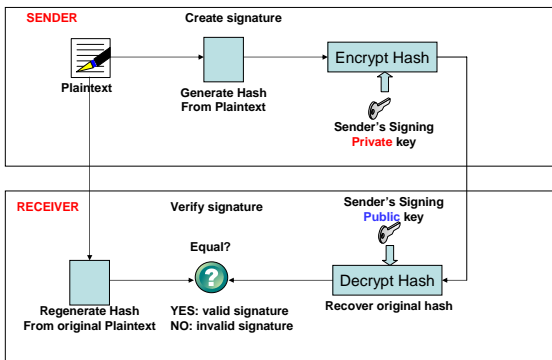
- A digital signature is a **cryptographically based signature**
- Digital signatures provides for:
 - Authentication
 - Integrity
 - Non-repudiation (makes it more difficult)

repudiation refers to the act of disclaiming responsibility of an action
Eg: I repudiate this message and its contents

Digital Signature

- ☞ A digital signature can be used with any kind of message, whether it is encrypted or not
- ☞ Digital signatures use cryptographic hash functions AND public key cryptography

Digital Signature



Digital Signature – how it works

- **The Sender:**
 1. generates a message digest from the message/content.
 2. use a private key (previously obtained from a public-private key authority) to encrypt the digest.
 3. the encrypted hash becomes the sender's digital signature of the message. (Note that it will be different each time you send a message.)
- **The Receiver:**
 1. makes a hash of the received message.
 2. uses the senders' public key to decrypt the message hash.
 3. If the hashes match, the received message is valid:
 - the encryption was done with the sender's private key (authenticity)
 - the message hasn't been altered since it was signed (integrity)
 - BUT No absolute certainty that the purported sender is indeed the signer – i.e., the person who used the private key – since the cryptosystem might have been broken, or the key copied.

PURDUE UNIVERSITY

Digital Signature strength factors

- The strength of a digital signature depends on:
 - the strength of the encryption method used
 - The management of the private key

41

PURDUE UNIVERSITY

Conventional Signature vs. Digital Signature

- Inclusion**
 - A conventional signature is included in the document
 - A digital signature is a separate "document" that can be sent apart of the original document
- Verification method**
 - Conventional signature: compare the signature on the document with a stored one
 - Digital signature: no need of storing the signature
- Relationship**
 - Conventional signature: 1-to-many relationship between the signature and the signed documents
 - Digital signature: 1-to-1 relationship between a signature and the signed document/message
- Duplicity**
 - Conventional signature: a copy of the original document can be distinguished from the original one
 - Digital signature: no such distinction (unless temporal info is used – timestamp)

PURDUE UNIVERSITY

Encryption vs. Digest

Encryption

cleartext → encrypt → ciphertext

hello, world → uyyh..lheyh

this is cleartext that anybody can easily read without the key used by encryption. It's also longer than the box of text above.

this is some really long text that we mean to encrypt, and to keep these pearls of wisdom out of the reach of the bad guy.

We don't really know how anybody could ever break our rot13 encryption, but if the NSA puts its mind to it, perhaps they will manage.

It's not an easy job making up random text for examples.

ciphertext → decrypt → cleartext

Guaf of pyrnagkg gung malokl pna mhyt' emq gupdng gur ut hqz of rapalcpba Vgt nyb ortte guna gur oib. ba gkg notu.

Guaf of fbr emyrl vlt ulj gkg gung f' zma gh rapalcg, nat gh xrc gur emyrl ba pfaez bng ba gur empu ba gur emq tll.

Ji qhag emyrl vltg ulj malob pbyq ruo omes the abgt13 rapalcpba. oib va gur AFN chfg vgt zrag gh eg. crenit' gur jyy znanr.

Vgt abg na mt who zvatat hc enagbz gkg sbe nbcyrf.

output same size as input

Digest

hello, world → hash function → 22c3683d94136c358091ae71a20d4

this is cleartext that anybody can easily read without the key used by encryption. It's also longer than the box of text above.

This is some really long text that we mean to encrypt, and to keep these pearls of wisdom out of the reach of the bad guy.

We don't really know how anybody could ever break our rot13 encryption, but if the NSA puts its mind to it, perhaps they will manage.

It's not an easy job making up random text for examples.

hash function → 14184923636145602a39d4003466

always / 128 bits

hash function → 447e806acc48eaac348bada7843ba

Source: <http://www.unixwiz.net/techtips/guide-crypto-hashes.html#whatis>

43

PURDUE UNIVERSITY

Summary

	Secret Key Setting	Public Key Setting
Secrecy / Confidentiality	Stream cipher Block cipher + encryption modes AES	Public key encryption: RSA
Authenticity / Integrity	MAC:HMAC	Digital Signatures RSA

44

- AES and DES calculators:
<http://www.unsw.adfa.edu.au/~lpb/src/>
- Cryptography:
- William Stalling website:
 - http://www.pearsonhighered.com/educator/academic/product/0,,0131873164,00%2ben-USS_01DBC.html
 - <http://www.williamstallings.com/Crypto/Crypto4e.html>