

# **CSE 410/565: Computer Security**

Instructor: Dr. Ziming Zhao

# Last Class

- Diffie-Hellman Key Agreement Protocol
  - The first practical method for establishing a shared secret over an unsecured communication channel
- Cryptographic hash functions

# Password Hashing

# Property of Hash Functions (SHA-3)

- Fast and efficient
  - Perform many hash operations in a short amount of time
- This is a desirable property for many applications, but not for ***password hashing***, where the goal is to slow down attackers who are trying to crack the passwords
  - Simple hashing is vulnerable to attacks such as brute force, as attackers can easily generate hashes for common passwords and compare them to the hashes in a database.

# Password-based Key Derivation Function (PBKDF)

- PBKDF is a crucial component of secure password storage.
- It is a function that transforms a password into a derived key, which can be used as an encryption key or as input to another cryptographic function.
- PBKDF is designed to be slow and computationally expensive, making it difficult for attackers to crack the passwords through brute force attacks.

# How does PBKDF work?

- PBKDF works by taking a password, a salt, an iteration count, and a desired output length as inputs.
- The salt is a random value that is added to the password before it is hashed. This helps to prevent dictionary attacks, as an attacker would have to generate hashes for every possible combination of password and salt.
- The iteration count determines the number of times the PBKDF function will run. This makes the PBKDF process slower and more computationally expensive, increasing the difficulty of cracking the password through brute force or dictionary attacks.

# Common PBKDF Algorithms

- PBKDF is implemented using a variety of algorithms, each with its own strengths and weaknesses.
- Some of the most commonly used PBKDF algorithms include:
  - PBKDF2 (Password-Based Key Derivation Function 2)
  - Bcrypt
  - Scrypt
  - Argon2
- Each algorithm uses a different approach to generate the derived key, and choosing the right algorithm can depend on the specific security requirements and constraints of the application.

# Example: Argon2

- Winner of the 2015 Password Hashing Competition
- Argon2 takes several parameters, including the password, the salt, and the desired complexity (time and memory cost) of the hash. The function returns the hashed password.

```
import argon2

# Choose the desired complexity
time_cost = 6
memory_cost = 2**16 # 64 MB

# Hash the password
hashed_password = argon2.hash("password", salt="salt", time_cost=time_cost,
memory_cost=memory_cost)
```

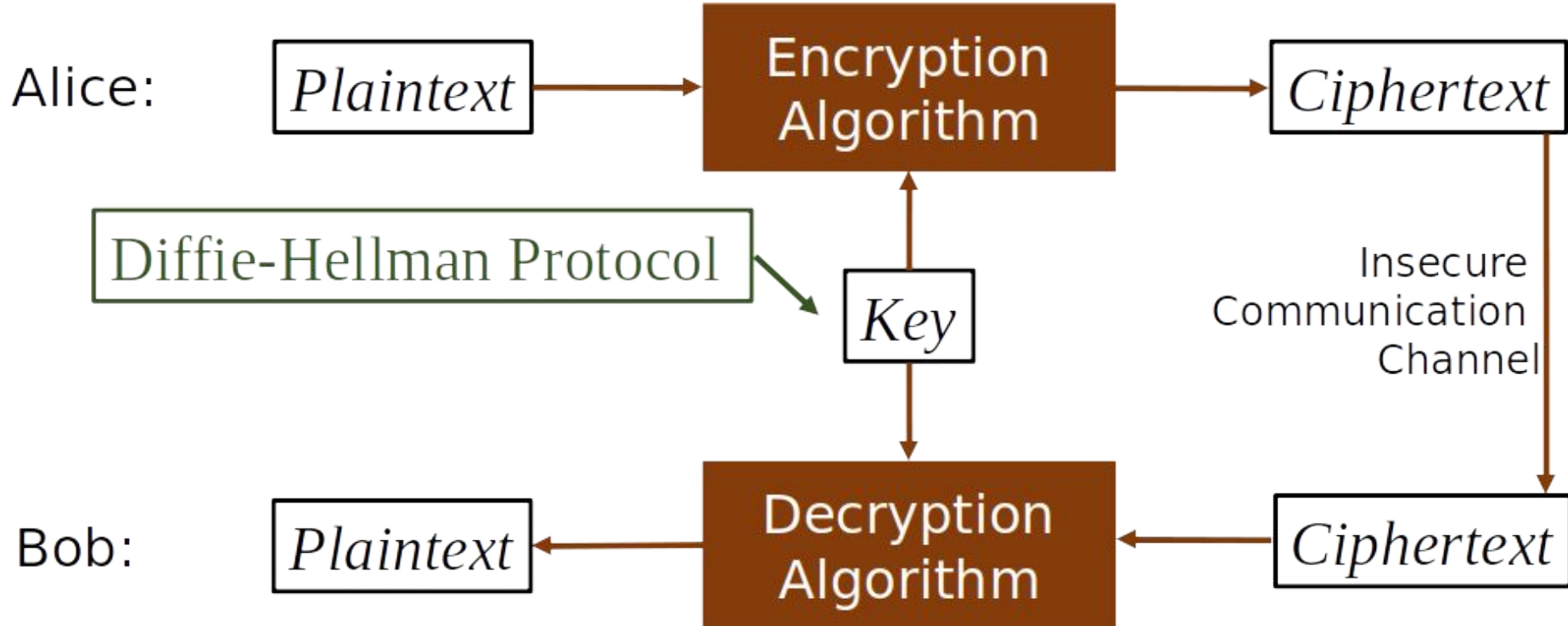


# **Public Key Cryptography**

# Symmetric key crypto

- What we already know
  - symmetric key cryptography enables confidentiality
    - achieved through secret key encryption
  - symmetric key cryptography enables authentication and integrity
    - achieved through MACs
- In all of the above the sender and receiver must share a secret key
  - need a secure channel for key distribution
  - not possible for parties with no prior relationship
  - more powerful public-key cryptography can aid with this

# Diffie-Hellman protocol



# Public Key Cryptography

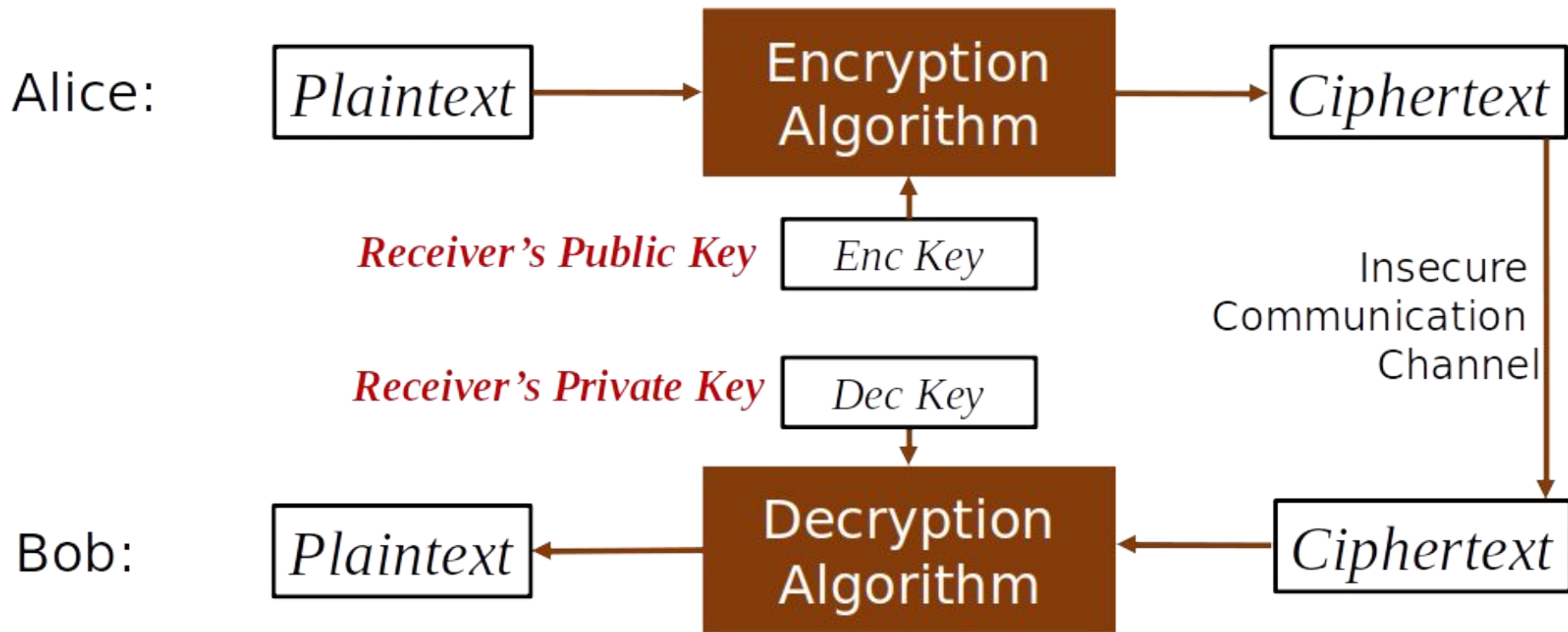
- Public-key encryption

- a party creates a public-private key pair
  - the public key is  $pk$
  - the private or secret key is  $sk$
- the public key is used for encryption and is publicly available
- the private key is used for decryption only

$$\text{Dec}_{sk}(\text{Enc}_{pk}(m)) = m$$

- knowing the public key and the encryption algorithm only, it is computationally infeasible to find the secret key
- public-key crypto systems are also called asymmetric

# Public Key Cryptography



# Public Key Cryptography

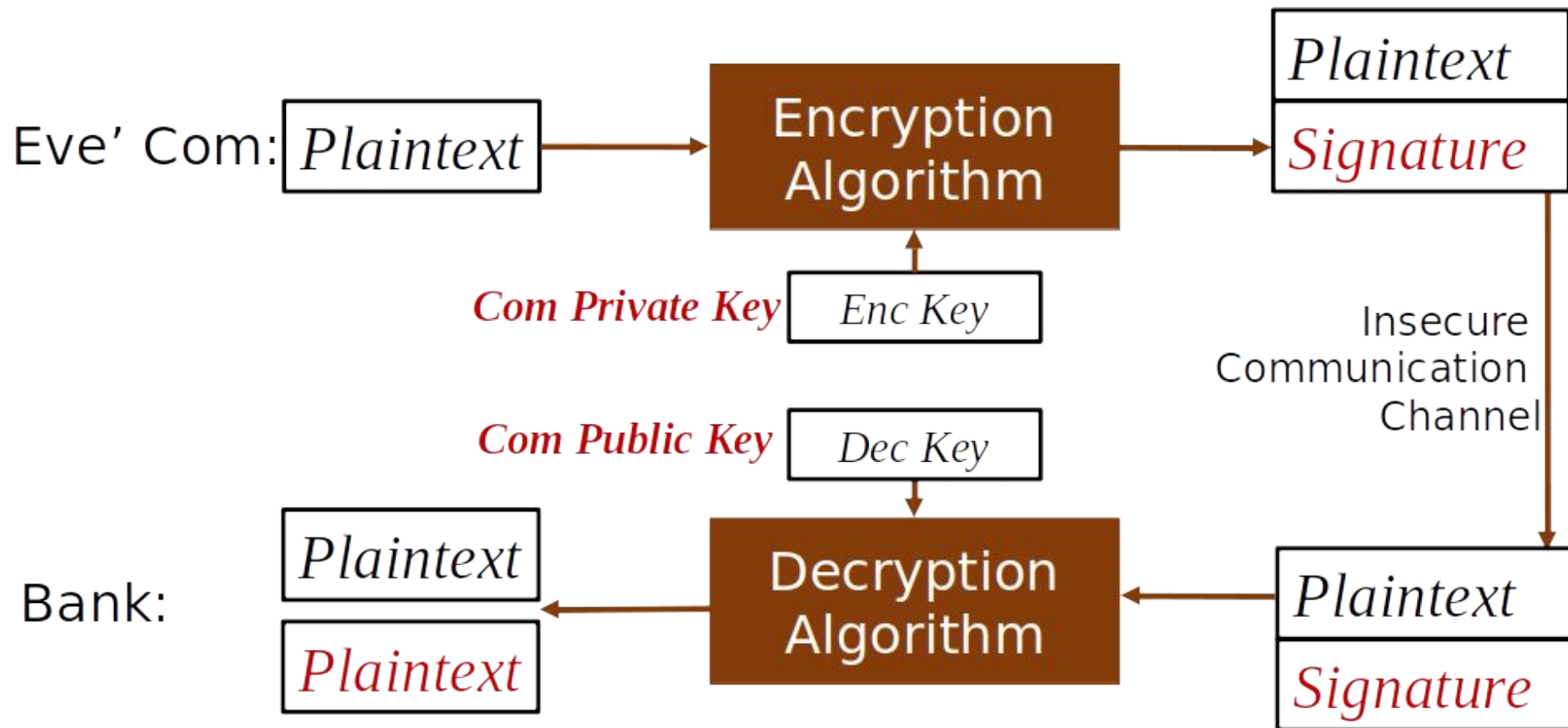
- Digital signatures

- a party generated a public private signing key pair
- private key is used to sign a message
- public key is used to verify a signature on a message
- can be viewed as one-way message authentication

- (Public-key) Key agreement or key distribution

- prior to the protocols the parties do not share a common secret
- after the protocol execution, they hold a key not known to any eavesdropper

# Digital Signature



# How Public-Key Cryptography Works

- Public-key constructions often use number theory and are based on a **special function  $f$**  with the following properties
  - given  $f$  and  $x$ , it is easy to compute  $f(x)$
  - given  $f(x)$ , it is hard to compute  $x$
  - given  $f(x)$  and an additional secret  $t$ , it is easy to find  $x$
  - function  $f$  is called a **one-way trapdoor function** and  $t$  is called the **trapdoor** of  $f$



# How Public-Key Cryptography Works

- Given such a function  $f$ , we **construct encryption** as follows:
  - $f$  is equivalent to encryption  $\text{Enc}_{pk}$
  - the private key serves the purpose of the trapdoor
  - given  $f(x) = \text{Enc}_{pk}(x)$  and the trapdoor  $sk$ , decryption of  $x$  is easy

# Public-Key Encryption

- Similar to symmetric encryption, we can formulate a number of attacks on public-key encryption
  - ciphertext only attack
  - known plaintext attack
  - chosen plaintext attack
  - chosen ciphertext attack

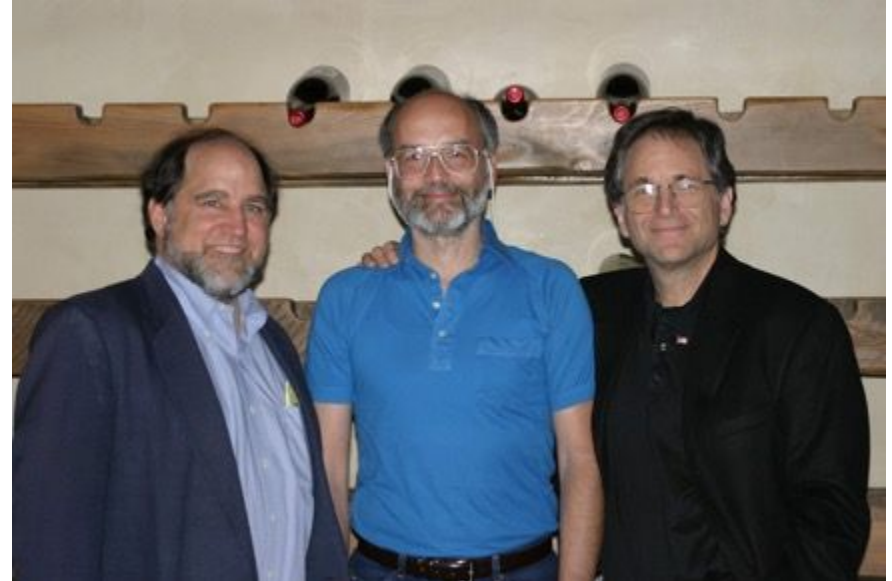
# Public-Key Encryption

- Almost all public-key encryption algorithms use **number theory and modular arithmetic**
  - **RSA** is based on the hardness of factoring large numbers
  - **ElGamal** is based on the hardness of solving discrete logarithm problem
- **RSA is the most commonly used public-key encryption algorithm**  
invented by Rivest, Shamir, and Adleman in 1978
  - sustained many years of attacks on it
  - relies on the fact that **factoring large numbers is hard**
    - let  $n = pq$ , where  $p$  and  $q$  are large primes
    - given only  $n$ , it is hard to find  $p$  or  $q$ , which are used as a trapdoor

# RSA



~1980



2003

Recipients of the 2002 ACM  
A.M. Turing Award

# RSA Cryptosystem

- RSA key generation
  - generate two large prime numbers  $p$  and  $q$  of the same length
  - compute  $n = pq$
  - choose a small prime number  $e$
  - compute the smallest  $d$  such that  $ed \bmod (p - 1)(q - 1) = 1$
  - here  $\varphi(n) = (p - 1)(q - 1)$  is Euler's totient function
- Public key is  $(e, n)$
- Private key is  $d$

# Plain RSA Encryption

- Encryption

- given a message  $m$  such that  $0 < m < n$
- given a public key  $pk = (e, n)$
- encrypt as  $c = \text{Enc}_{pk}(m) = m^e \bmod n$

- Decryption

- given a ciphertext  $c$  ( $0 < c < n$ )
- given a public key  $pk = (e, n)$  and the corresponding private key  $sk = d$
- decrypt as  $m = \text{Dec}_{sk}(c) = c^d \bmod n$

# Plain RSA Encryption

- Example of Plain RSA

- key generation

- $p = 11, q = 7, n = pq = 77, \phi(n) = 60$
    - $e = 37 \Rightarrow d = 13$  (i.e.,  $ed = 481$ ;  $ed \bmod 60 = 1$ )
    - public key is  $pk = (37, 77)$  and private key is  $sk = 13$

- encryption

- let  $m = 15$
    - $c = \text{Enc}(m) = m^e \bmod n = 15^{37} \bmod 77 = 71$
    - decryption
      - $m = \text{Dec}(c) = c^d \bmod n = 71^{13} \bmod 77 = 15$

# Security of RSA

- Existing attacks on RSA
  - brute force search (try all possible keys)
  - number theoretic attacks (factor  $n$ )
    - complicated factoring algorithms that run in sub-exponential (but super-polynomial) time in the length of  $n$  exist
    - a 768-bit modulus was factored in 2009
    - 1024-bit moduli could be factored very soon
    - moduli of length 2048 are expected to be secure until 2030
  - special use cases
    - e.g., encrypting small messages with small  $e$
- Plain (or textbook) RSA is not close to secure



# Towards Safe Use of RSA

- Padded RSA

- plain RSA is deterministic
- this is even worse than in case of symmetric encryption
  - anyone can search for  $m$  encrypting various messages
- we can randomize ciphertext by padding each  $m$  with random bits
  - now a message can be at most  $k - t$  bits long
  - random  $t$  bits are added to it such that  $2^t$  work is infeasible

# Towards Safe Use of RSA

- **PKCS #1 v1.5** was a widely used standard for padded RSA
  - PKCS = RSA Laboratories Public-Key Cryptography Standard
  - it is believed to be CPA-secure
- **PKCS #1 v2.0** utilizes OAEP (Optimal Asymmetric Encryption Padding)
  - the newer version mitigates some attacks on v1.5 and is known to be CCA-secure
  - in OAEP, we use plain RSA encryption on  $m \oplus g(r) || r \oplus h(m \oplus g(r))$ , where  $h$  and  $g$  are hash functions and  $r$  is randomness

# Towards Safe Use of RSA

- Making factoring infeasible
  - choose  $n$  to be long enough (we can choose any  $n$ !)
  - for a security parameter  $k$ , compute  $n$  with  $|n| = k$
- A good implementation will also have countermeasures against implementation-level attacks
  - timing attacks, special cases of  $e$  and  $d$ , etc

# Other Public-Key Algorithms

- Many popular public key algorithms rely on the difficulty of **discrete logarithm problem**
  - ElGamal encryption and ElGamal signature
  - Digital Signature Algorithm (DSA)
  - Diffie-Hellman key exchange
  - ...
- Given an appropriate setup with  $g$ ,  $p$ , and  $h = g^x \bmod p$ , it is difficult for someone to compute  $x$ 
  - $x$  is called the **discrete logarithm** of  $h$  to the base  $g$
  - groups in which the discrete logarithm problem is hard use prime modulus  $p$  (conventional and elliptic curve settings)

# Symmetric vs Public-Key Encryption

- Public-key operations are orders of magnitude slower than symmetric encryption
  - a multiplication modulo  $n$  requires close to  $O(|n|^2)$  work
  - a full-size exponentiation modulo  $n$  requires close to  $O(|n|^3)$  work
    - it is the cost of multiplication times the exponent size
  - public-key encryption is typically not used to communicate large volumes of data
    - it is rather used to communicate (or agree on) a symmetric key
    - the data itself is sent encrypted with the symmetric key
- In RSA, decryption is significantly slower than encryption, with key generation being the slowest

# Digital Signatures

- A **digital signature scheme** is a method of signing messages stored in electronic form and verifying signatures
- Digital signatures can be used in very similar ways conventional signatures are used
  - paying by a credit card and signing the bill
  - signing a contract
  - signing a letter
- Unlike conventional signatures, we have that
  - digital signatures are not physically attached to messages
  - we cannot compare a digital signature to the original signature

# Digital Signatures

- Digital signatures allows us to achieve the following security objectives:
  - authentication
  - Integrity
  - Non-repudiation
    - note that this is the main difference between signatures and MACs
    - a MAC cannot be associated with a unique sender since a symmetric shared key is used
- What security property do we want from a digital signature scheme?  
How does it relate to that of MACs?

# Digital Signatures

- It is meaningful to consider the following **attack models**
  - key-only attack
  - known message attack
  - chosen message attack
- **Adversarial goals** might be
  - total break
  - selective forgery
  - existential forgery



# Digital Signatures

- A digital signature scheme consists of **key generation**, **message signing**, and **signature verification** algorithms
  - **key generation** creates a public-private key pair  $(pk, sk)$
  - **signing algorithm** takes a messages and uses private signing key to output a signature
  - **signature verification algorithm** takes a message, a signature on it, and the signer's public key and outputs a yes/no answer

# Plain RSA Signatures

- Plain RSA signature is similar to plain RSA encryption
  - create a key pair as before: public  $pk = (e, n)$  and private  $sk = d$
  - signing of message  $m$  using  $sk$  is done as  $\sigma = m^d \bmod n$
  - verification of signature  $\sigma$  on message  $m$  using  $pk$  is performed as  $\sigma^e \bmod n \stackrel{?}{=} m$

# Digital Signatures

- Plain RSA is not a secure signature scheme
  - both existential and selective forgeries are easy
  - the “hash-and-sign” paradigm is used in many constructions to achieve adequate security
    - e.g., compute  $h(m)$  and then continue with plain RSA signing of  $h(m)$
  - this additionally improves efficiency
  - the hash function must satisfy all three security properties
    - preimage resistance
    - weak collision resistance
    - strong collision resistance

# Digital Signatures

- RSA signatures

- key generation

- choose prime  $p$  and  $q$ , compute  $n = pq$
- choose prime  $e$  and compute  $d$  s.t.  $ed \bmod (p-1)(q-1) = 1$
- signing key is  $d$ , verification key is  $(e, n)$

- message signing

- given  $m$ , compute  $h(m)$
- output  $\sigma = h(m)^d \bmod n$

- signature verification

- given  $m$  and  $\sigma$ , first compute  $h(m)$
- check whether  $\sigma^e \bmod n \stackrel{?}{=} h(m)$

# Digital Signature Standard (DSS)

- Digital Signature Standard (DSS) or Digital Signature Algorithm (DSA) was adopted as a standard in 1994
  - its design was influenced by prior ElGamal and Schnorr signature schemes
  - it assumes the difficulty of the discrete logarithm problem
  - no formal security proof exists

# Digital Signature Standard (DSS)

- DSS was published in 1994 as [FIPS PUB 186](#)
  - it was specified to hash the message using SHA-1 before signing
  - it was specified to produce a 320-bit signature on a 160-bit hash
- The current version is [FIPS PUB 186-4](#) (2013)
  - DSA can now be used with a 1024-, 2048-, or 3072-bit modulus
  - the message size is 320, 448, or 512 bits
- **Signing** and **signature verification** involve:
  - hashing the message
  - computing a couple of modulo exponentiations on both longer and shorter sizes

# Digital Signature Standard (DSS)

- Thorough evaluation of security of a signature scheme is crucial
  - often a message can be encrypted and decrypted once and long-term security for the key is not required
  - signatures can be used on legal documents and may need to be verified many years after signing
  - choose the key length to be secure against future computing speeds

# Bit Security

- All constructions studied so far rely on the fact that an **adversary is limited in computational power**
  - if it has more resources than we anticipate, cryptographic algorithms can be broken
- Today, **112–128-bit security is considered sufficient**
  - this means approximately that for 128-bit security,  $2^{128}$  operations are needed to violate security with high probability
- This translates into the following parameters
  - **symmetric key encryption**: the key size is at least 112 bits
  - **hash functions**: the hash size is at least 224 bits
  - **public key encryption**: the modulus is at least 2048 bits long



# The Big Picture

- How we address **security goals** using different tools

Security goal	Symmetric key setting	Public key setting
Secrecy / confidentiality	block ciphers with encryption modes (AES); stream ciphers	public key encryption (RSA, ElGamal, etc.)
Authenticity / integrity	message authentication codes (CBC-MAC, HMAC)	digital signatures (RSA, DSA, etc.)

# Conclusions

- Proper use of cryptographic tools requires great care
- Safe use of such algorithms involves
  - familiarity with known attacks
  - adequate choice of parameters
  - including countermeasures against known attacks on implementations
  - using a cryptographically strong source of randomness
- No security by obscurity!