CSCI-GA.3205 Applied Cryptography & Network Security

Department of Computer Science
New York University

PRESENTED BY DR. MAZDAK ZAMANI
mazdak.zamani@NYU.edu

Public key encryption
ElGamal encryption
Digital signatures

06

NYU

Asymmetric encryption

- Public-key encryption is sometimes called *asymmetric encryption* to denote the fact that the encryptor uses one key, pk, and the decryptor uses a different key, sk.
- The basic idea of public-key encryption is that the receiver, Bob in this case, runs a key generation algorithm G, obtaining a pair of keys:

$$(pk, sk) \overset{\text{R}}{\leftarrow} G()$$

- The key pk is Bob's public key, and sk is Bob's secret key. As their names imply, Bob should keep sk secret, but may publicize pk.

Asymmetric encryption

- To send Bob an encrypted email message, Alice needs two things: Bob's email address, and Bob's public key pk.
- So let us assume now that Alice has Bob's email address and public key pk. To send Bob an encryption of her email message m, she computes the ciphertext:

$$c \overset{R}{\leftarrow} E(pk, m)$$

- She then sends c to Bob, using his email address. At some point later, Bob receives the ciphertext c, and decrypts it, using his secret key:

$$m \leftarrow D(sk, c)$$

Public-key encryption scheme

Definition 11.1. *A public-key encryption scheme $\mathcal{E} = (G, E, D)$ is a triple of efficient algorithms: a* ***key generation algorithm*** *G, an* ***encryption algorithm*** *E, a* ***decryption algorithm*** *D.*

- *G is a probabilistic algorithm that is invoked as $(pk, sk) \overset{R}{\leftarrow} G()$, where pk is called a* ***public key*** *and sk is called a* ***secret key****.*
- *E is a probabilistic algorithm that is invoked as $c \overset{R}{\leftarrow} E(pk, m)$, where pk is a public key (as output by G), m is a message, and c is a ciphertext.*
- *D is a deterministic algorithm that is invoked as $m \leftarrow D(sk, c)$, where sk is a secret key (as output by G), c is a ciphertext, and m is either a message, or a special* reject *value (distinct from all messages).*

Attack Game (semantic security)

Attack Game 11.1 (semantic security). For a given public-key encryption scheme $\mathcal{E} = (G, E, D)$, defined over $(\mathcal{M}, \mathcal{C})$, and for a given adversary $\mathcal{A}$, we define two experiments.

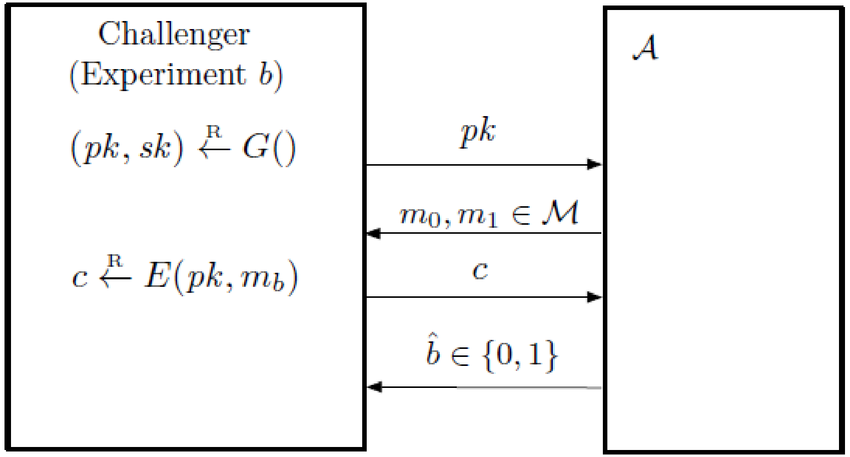
Experiment b $\quad (b = 0, 1)$:

- The challenger computes $(pk, sk) \overset{R}{\leftarrow} G()$, and sends pk to the adversary.
- The adversary computes $m_0, m_1 \in \mathcal{M}$, of the same length, and sends them to the challenger.
- The challenger computes $c \overset{R}{\leftarrow} E(pk, m_b)$, and sends c to the adversary.
- The adversary outputs a bit $\hat{b} \in \{0, 1\}$.

Experiment b of Attack Game (semantic security)

If W_b is the event that $\mathcal{A}$ outputs 1 in Experiment b, we define $\mathcal{A}$'s **advantage** with respect to $\mathcal{E}$ as

$$\mathsf{SSadv}[\mathcal{A}, \mathcal{E}] := \Big| \Pr[W_0] - \Pr[W_1] \Big|. \quad \square$$



Challenger
(Experiment b)
$(pk, sk) \overset{R}{\leftarrow} G()$
$c \overset{R}{\leftarrow} E(pk, m_b)$
$\mathcal{A}$
pk
$m_0, m_1 \in \mathcal{M}$
c
$\hat{b} \in \{0,1\}$

Definition 11.2 (semantic security). *A public-key encryption scheme $\mathcal{E}$ is* ***semantically secure*** *if for all efficient adversaries $\mathcal{A}$, the value $\mathsf{SSadv}[\mathcal{A}, \mathcal{E}]$ is negligible.*

Implications of semantic security

- We first show that any semantically secure public-key scheme must use a randomized encryption algorithm.
- We also show that in the public-key setting, semantic security implies CPA security.
    - This was not true for symmetric encryption schemes: the one-time pad is semantically secure, but not CPA secure.

The need for randomized encryption

Let $\mathcal{E} = (G, E, D)$ be a semantically secure public-key encryption scheme defined over $(\mathcal{M}, \mathcal{C})$ where $|\mathcal{M}| \geq 2$. We show that the encryption algorithm E must be randomized, otherwise the scheme cannot be semantically secure.

To see why, suppose E is deterministic. Then the following adversary $\mathcal{A}$ breaks semantic security of $\mathcal{E} = (G, E, D)$:

The need for randomized encryption

- $\mathcal{A}$ receives a public key pk from its challenger.
- $\mathcal{A}$ chooses two distinct messages m_0 and m_1 in $\mathcal{M}$ and sends them to its challenger. The challenger responds with $c := E(pk, m_b)$ for some $b \in \{0, 1\}$.
- $\mathcal{A}$ computes $c_0 := E(pk, m_0)$ and outputs 0 if $c = c_0$. Otherwise, it outputs 1.

Because E is deterministic, we know that $c = c_0$ whenever $b = 0$. Therefore, when $b = 0$ the adversary always outputs 0. Similarly, when $b = 1$ it always outputs 1. Therefore

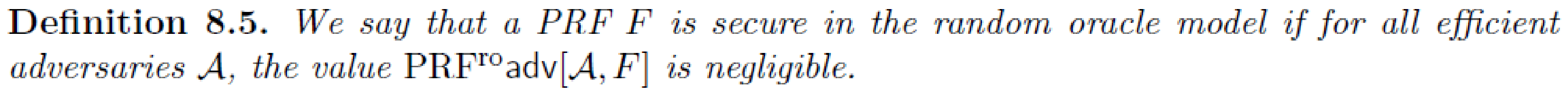
$$\mathsf{SSadv}[\mathcal{A}, \mathcal{E}] = 1$$

Random oracles

- The idea is that we simply model a hash function H *as if* it were a truly random function O. The random oracle is implemented using an associative array Map : $G^2 \rightarrow K$.
- If H maps M to T , then O is chosen uniformly at random from the set Funs[M; T].
- We can translate any attack game into its random oracle version:
- The function O is called a **random oracle** and security in this setting is said to hold in the random oracle model.

Attack Game (PRF in the random oracle model)

- We have a PRF F that uses a hash function H as an oracle,
- We denote by F^{O} the function that uses the random oracle O in place of H.



Definition 8.5. *We say that a PRF F is secure in the random oracle model if for all efficient adversaries $\mathcal{A}$, the value* $\mathrm{PRF}^{\mathrm{ro}}\mathsf{adv}[\mathcal{A}, F]$ *is negligible.*

- Let F be a PRF defined over (K; X; Y) that uses a hash function H defined over (M; T) as an oracle.
- For a given adversary A, we define two experiments, Experiment 0 and Experiment 1. For b = 0; 1, we define:

Attack Game (PRF in the random oracle model)

- Experiment b:
  - $\mathcal{O} \xleftarrow{R} \text{Funs}[\mathcal{M}, \mathcal{T}]$.
  - The challenger selects $f \in \text{Funs}[\mathcal{X}, \mathcal{Y}]$ as follows:

    if $b = 0$: $k \xleftarrow{R} \mathcal{K}$, $f \leftarrow F^{\mathcal{O}}(k, \cdot)$;
    if $b = 1$: $f \xleftarrow{R} \text{Funs}[\mathcal{X}, \mathcal{Y}]$.

  - The adversary submits a sequence of queries to the challenger.
    - F-query: respond to a query $x \in \mathcal{X}$ with $y = f(x) \in \mathcal{Y}$.
    - $\mathcal{O}$-query: respond to a query $m \in \mathcal{M}$ with $t = \mathcal{O}(m) \in \mathcal{T}$.
  - The adversary computes and outputs a bit $\hat{b} \in \{0, 1\}$.

For $b = 0, 1$, let W_b be the event that $\mathcal{A}$ outputs 1 in Experiment b. We define $\mathcal{A}$'s **advantage** with respect to F as

$$\text{PRF}^{\text{ro}}\text{adv}[\mathcal{A}, F] := \Big|\Pr[W_0] - \Pr[W_1]\Big|. \quad \square$$

Semantic security against chosen plaintext attack

- In the public-key setting, the adversary can encrypt any message he likes, without knowledge of any secret key material.
- The adversary does so by using the given public key and never needs to issue encryption queries to the challenger.
- In contrast, in the symmetric key setting, the adversary cannot encrypt messages on his own.

Attack Game (CPA security)

Attack Game 11.2 (CPA security). For a given public-key encryption scheme $\mathcal{E} = (G, E, D)$, defined over $(\mathcal{M}, \mathcal{C})$, and for a given adversary $\mathcal{A}$, we define two experiments.

Experiment b $\quad (b = 0, 1)$:

- The challenger computes $(pk, sk) \overset{R}{\leftarrow} G()$, and sends pk to the adversary.
- The adversary submits a sequence of queries to the challenger.

  For $i = 1, 2, \ldots,$ the ith query is a pair of messages, $m_{i0}, m_{i1} \in \mathcal{M}$, of the same length.

  The challenger computes $c_i \overset{R}{\leftarrow} E(pk, m_{ib})$, and sends c_i to the adversary.
- The adversary outputs a bit $\hat{b} \in \{0, 1\}$.

If W_b is the event that $\mathcal{A}$ outputs 1 in Experiment b, then we define $\mathcal{A}$'s **advantage** with respect to $\mathcal{E}$ as

$$\text{CPAadv}[\mathcal{A}, \mathcal{E}] := \Big|\Pr[W_0] - \Pr[W_1]\Big|. \quad \Box$$

NYU

Semantic security against chosen plaintext attack

Definition 11.4 (CPA security). *A public-key encryption scheme $\mathcal{E}$ is called* ***semantically secure against chosen plaintext attack****, or simply* ***CPA secure****, if for all efficient adversaries $\mathcal{A}$, the value $\text{CPAadv}[\mathcal{A},\mathcal{E}]$ is negligible.*

Theorem 11.1. *If a public-key encryption scheme $\mathcal{E}$ is semantically secure, then it is also CPA secure.*

In particular, for every CPA adversary $\mathcal{A}$ that plays Attack Game 11.2 with respect to $\mathcal{E}$, and which makes at most Q queries to its challenger, there exists an SS adversary $\mathcal{B}$, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that

$$\text{CPAadv}[\mathcal{A},\mathcal{E}] = Q \cdot \text{SSadv}[\mathcal{B},\mathcal{E}].$$

Encryption based on a trapdoor function scheme

Our encryption scheme is called $\mathcal{E}_{\text{TDF}}$, and is built out of several components:

- a trapdoor function scheme $\mathcal{T} = (G, F, I)$, defined over $(\mathcal{X}, \mathcal{Y})$,
- a symmetric cipher $\mathcal{E}_{\text{s}} = (E_{\text{s}}, D_{\text{s}})$, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$,
- a hash function $H : \mathcal{X} \to \mathcal{K}$.

NYU

Encryption based on a trapdoor function scheme

The message space for $\mathcal{E}_{\text{TDF}}$ is $\mathcal{M}$, and the ciphertext space is $\mathcal{Y} \times \mathcal{C}$. We now describe the key generation, encryption, and decryption algorithms for $\mathcal{E}_{\text{TDF}}$.

- The key generation algorithm for $\mathcal{E}_{\text{TDF}}$ is the key generation algorithm for $\mathcal{T}$.
- For a given public key pk, and a given message $m \in \mathcal{M}$, the encryption algorithm runs as follows:

$$\begin{aligned} E(pk, m) := \quad & x \xleftarrow{\text{R}} \mathcal{X}, \quad y \leftarrow F(pk, x), \quad k \leftarrow H(x), \quad c \xleftarrow{\text{R}} E_{\text{s}}(k, m) \\ & \text{output } (y, c). \end{aligned}$$

- For a given secret key sk, and a given ciphertext $(y, c) \in \mathcal{Y} \times \mathcal{C}$, the decryption algorithm runs as follows:

$$\begin{aligned} D(sk,\ (y, c)\ ) := \quad & x \leftarrow I(sk, y), \quad k \leftarrow H(x), \quad m \leftarrow D_{\text{s}}(k, c) \\ & \text{output } m. \end{aligned}$$

NYU

Encryption based on a trapdoor function with RSA

- This scheme is parameterized by two quantities: the length l of the prime factors of the RSA modulus, and the encryption exponent e, which is an odd, positive integer.
- Let us assume that X is a fixed set into which we may embed $\mathbb{Z}_N$, for every RSA modulus n generated by RSAGen($l; e$) (for example, we could take X = $\{0, 1\}^{2l}$).
- The scheme also makes use of a symmetric cipher $\xi = (E_s; D_s)$ defined over (**K; M; C**), as well as a hash function **H : X → K**.

Encryption based on a trapdoor function with RSA

- The basic RSA encryption scheme is ξ_{RSA} = (G; E; D), with message space **M** and ciphertext space X × C, where
  - the key generation algorithm runs as follows:

$$\begin{aligned} G() := \quad & (n,d) \xleftarrow{R} \text{RSAGen}(\ell, e), \quad pk \leftarrow (n,e), \quad sk \leftarrow (n,d) \\ & \text{output } (pk, sk); \end{aligned}$$

  - for a given public key $pk = (n,e)$, and message $m \in \mathcal{M}$, the encryption algorithm runs as follows:

$$\begin{aligned} E(pk, m) := \quad & x \xleftarrow{R} \mathbb{Z}_n, \quad y \leftarrow x^e, \quad k \leftarrow H(x), \quad c \xleftarrow{R} E_s(k,m) \\ & \text{output } (y,c) \in \mathcal{X} \times \mathcal{C}; \end{aligned}$$

  - for a given secret key $sk = (n,d)$, and a given ciphertext $(y,c) \in \mathcal{X} \times \mathcal{C}$, where y represents an element of $\mathbb{Z}_n$, the decryption algorithm runs as follows:

$$\begin{aligned} D(sk, \ (y,c)\ ) := \quad & x \leftarrow y^d, \quad k \leftarrow H(x), \quad m \leftarrow D_s(k,c) \\ & \text{output } m. \end{aligned}$$

NYU

Incorrect use of a Trapdoor Function (TDF)

Never encrypt by applying F directly to plaintext:

E(pk, m) :

output $c \leftarrow F(pk, m)$

D(sk, c) :

output $F^{-1}(sk, c)$

Problems:

- Deterministic functions will not be semantically secure if used for public-key encryption from TDFs! Why?

Public-key encryption from TDFs

(G, F, F^{-1}): secure TDF $X \longrightarrow Y$

(E_s, D_s) : symmetric auth. encryption defined over (K,M,C)

$H: X \longrightarrow K$ a hash function

<u>E(pk, m)</u> :

```
    x ⟵ X,            y ⟵ F(pk, x)
    k ⟵ H(x),         c ⟵ E_s(k, m)
    output   (y, c)
```

<u>D(sk, (y,c))</u> :

```
    x ⟵ F^{-1}(sk, y),
    k ⟵ H(x),         m ⟵ D_s(k, c)
    output   m
```

$$\underbrace{\boxed{F(pk, x)}}_{\text{header}}\underbrace{\boxed{E_s(\, H(x),\ m\,)}}_{\text{body}}$$

ElGamal encryption

The encryption scheme is a variant of a scheme first proposed by ElGamal, and we call it $\mathcal{E}_{\mathrm{EG}}$. It is built out of several components:

- a cyclic group $\mathbb{G}$ of prime order q with generator $g \in \mathbb{G}$,
- a symmetric cipher $\mathcal{E}_{\mathrm{s}} = (E_{\mathrm{s}}, D_{\mathrm{s}})$, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$,
- a hash function $H : \mathbb{G}^2 \to \mathcal{K}$.

The message space for $\mathcal{E}_{\mathrm{EG}}$ is $\mathcal{M}$, and the ciphertext space is $\mathbb{G} \times \mathcal{C}$. We now describe the key generation, encryption, and decryption algorithms for $\mathcal{E}_{\mathrm{EG}}$.

ElGamal encryption

- the key generation algorithm runs as follows:

$$\begin{aligned} G() := \quad & \alpha \xleftarrow{R} \mathbb{Z}_q, \quad u \leftarrow g^{\alpha} \\ & pk \leftarrow u, \quad sk \leftarrow \alpha \\ & \text{output } (pk, sk); \end{aligned}$$

- for a given public key $pk = u \in \mathbb{G}$ and message $m \in \mathcal{M}$, the encryption algorithm runs as follows:

$$\begin{aligned} E(pk, m) := \quad & \beta \xleftarrow{R} \mathbb{Z}_q, \quad v \leftarrow g^{\beta}, \quad w \leftarrow u^{\beta}, \quad k \leftarrow H(v, w), \quad c \leftarrow E_s(k, m) \\ & \text{output } (v, c); \end{aligned}$$

- for a given secret key $sk = \alpha \in \mathbb{Z}_q$ and a ciphertext $(v, c) \in \mathbb{G} \times \mathcal{C}$, the decryption algorithm runs as follows:

$$\begin{aligned} D(sk,\ (v,c)\ ) := \quad & w \leftarrow v^{\alpha}, \quad k \leftarrow H(v, w), \quad m \leftarrow D_s(k, c) \\ & \text{output } m. \end{aligned}$$

Thus, $\mathcal{E}_{\mathrm{EG}} = (G, E, D)$, and is defined over $(\mathcal{M}, \mathbb{G} \times \mathcal{C})$.

Note that the description of the group $\mathbb{G}$ and generator $g \in \mathbb{G}$ is considered to be a system parameter, rather than part of the public key.

NYU

Diffie-Hellman protocol (1977) in ElGamal pub-key encryption (1984)

Fix a finite cyclic group G $\left(\text{e.g } G = (Z_p)^*\right)$ of order n

Fix a generator g in G $\left(\text{i.e. } G = \{1, g, g^2, g^3, \ldots, g^{n-1}\}\right)$

<u>**Alice**</u> <u>**Bob**</u>

choose random **a** in {1,…,n} choose random **b** in {1,…,n}

$A = g^a \longrightarrow$

$\longleftarrow B = g^b$

$\mathbf{B^a} = (g^b)^a = \mathbf{k_{AB} = g^{ab}} = (g^a)^b = \mathbf{A^b}$

To encrypt: compute $g^{ab} = A^b$, derive symmetric key k , encrypt message m with k
To decrypt: compute $g^{ab} = B^a$, derive k, and decrypt

The ElGamal system (a modern view)

- G: finite cyclic group of order n
- (E_s, D_s) : symmetric auth. encryption defined over (K,M,C)
- $H: G^2 \longrightarrow K$ a hash function

```
E( pk=(g,u), m) :
    $b \leftarrow Z_n$ ,  $v \leftarrow g^b$ ,  $w \leftarrow u^b$
    $k \leftarrow H(v, w)$ ,  $c \leftarrow E_s(k, m)$
    output  $(v, c)$
```

```
D( sk=a, (v,c) ) :
    $w \leftarrow v^a$
    $k \leftarrow H(v, w)$ ,  $m \leftarrow D_s(k, c)$
    output  $m$
```

NYU

Secrecy vs Integrity

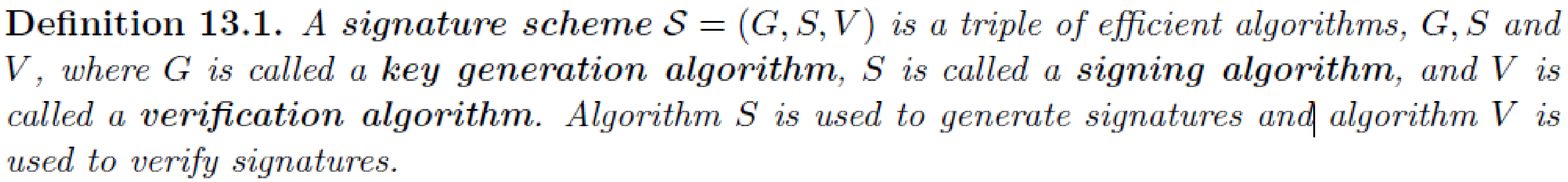
	Private-Key Setting	**Public-Key Setting**
Secrecy	Private-key encryption	Public-key encryption
Integrity	Message authentication codes	Digital signature schemes

Digital signatures

- Functionally, a digital signature is similar to a MAC.
- In a MAC, both the signing and verification algorithms use the same secret key.
- In a signature scheme, the signing algorithm uses one key, sk, while the verification algorithm uses another, pk.

Digital signatures

- Functionally, a digital signature is similar to a MAC.
- In a MAC, both the signing and verification algorithms use the same secret key.
- In a signature scheme, the signing algorithm uses one key, sk, while the verification algorithm uses another, pk.



Definition 13.1. *A signature scheme $\mathcal{S} = (G, S, V)$ is a triple of efficient algorithms, G, S and V, where G is called a* ***key generation algorithm****, S is called a* ***signing algorithm****, and V is called a* ***verification algorithm****. Algorithm S is used to generate signatures and algorithm V is used to verify signatures.*

Digital signatures

- *G is a probabilistic algorithm that takes no input. It outputs a pair (pk, sk), where sk is called a secret* ***signing key*** *and pk is called a public* ***verification key****.*
- *S is a probabilistic algorithm that is invoked as $\sigma \overset{R}{\leftarrow} E(sk, m)$, where sk is a secret key (as output by G) and m is a message. The algorithm outputs a* ***signature*** *σ.*
- *V is a deterministic algorithm invoked as $V(pk, m, \sigma)$. It outputs either* accept *or* reject.
- *We require that a signature generated by S is always accepted by V. That is, for all (pk, sk) output by G and all messages m, we have*

$$\Pr[V(pk,\ m,\ S(sk,\ m)\ ) = \text{accept}] = 1.$$

As usual, we say that messages lie in a finite ***message space*** *$\mathcal{M}$, and signatures lie in some finite* ***signature space*** *Σ. We say that $\mathcal{S} = (G, S, V)$ is defined over $(\mathcal{M}, \Sigma)$.*

NYU

Secure signatures

The definition of a secure signature scheme is similar to the definition of secure MAC. We give the adversary the power to mount a **chosen message attack**, namely the attacker can request the signature on any message of his choice. Even with such power, the adversary should not be able to create an **existential forgery**, namely the attacker cannot output a valid message-signature pair (m, σ) for some new message m. Here "new" means a message that the adversary did not previously request a signature for.

More precisely, we define secure signatures using an attack game between a challenger and an adversary $\mathcal{A}$. The game is described below and in Fig. 13.1.

NYU

Attack Game (Signature security)

Attack Game 13.1 (Signature security). For a given signature scheme $\mathcal{S} = (G, S, V)$, defined over $(\mathcal{M}, \Sigma)$, and a given adversary $\mathcal{A}$, the attack game runs as follows:

- The challenger runs $(pk, sk) \xleftarrow{R} G()$ and sends pk to $\mathcal{A}$.
- $\mathcal{A}$ queries the challenger several times. For $i = 1, 2, \ldots,$ the ith *signing query* is a message $m_i \in \mathcal{M}$. Given m_i, the challenger computes $\sigma_i \xleftarrow{R} S(sk, m_i)$, and then gives σ_i to $\mathcal{A}$.
- Eventually $\mathcal{A}$ outputs a candidate forgery pair $(m, \sigma) \in \mathcal{M} \times \Sigma$.

Attack Game (Signature security)

We say that the adversary wins the game if the following two conditions hold:

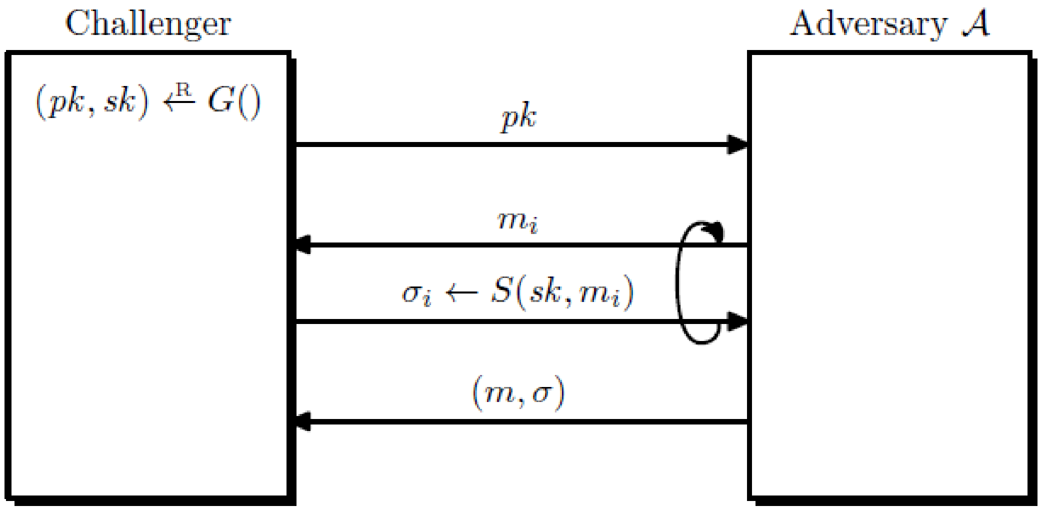
- $V(pk, m, \sigma) = \mathsf{accept}$, and
- m is new, namely $m \notin \{m_1, m_2, \ldots\}$.

We define $\mathcal{A}$'s advantage with respect to $\mathcal{S}$, denoted $\mathsf{SIGadv}[\mathcal{A}, \mathcal{S}]$, as the probability that $\mathcal{A}$ wins the game. Finally, we say that $\mathcal{A}$ is a **Q-query adversary** if $\mathcal{A}$ issues at most Q signing queries.

Definition 13.2. *We say that a signature scheme $\mathcal{S}$ is secure if for all efficient adversaries $\mathcal{A}$, the quantity* $\mathsf{SIGadv}[\mathcal{A}, \mathcal{S}]$ *is negligible.*

In case the adversary wins Attack Game 13.1, the pair (m, σ) it outputs is called an **existential forgery**. Systems that satisfy Definition 13.2 are said to be **existentially unforgeable under a chosen message attack**.

Signature attack game



Challenger
Adversary $\mathcal{A}$
$(pk, sk) \xleftarrow{R} G()$
pk
m_i
$\sigma_i \leftarrow S(sk, m_i)$
(m, σ)

Figure 13.1: Signature attack game (Attack Game 13.1)

Applications of digital signatures

Software distribution:

- Suppose a software company releases a software update for its product.
- Customers download the software update file U Before installing U on their machine.
- Customers want to verify that U really is from the company.
- A MAC system is of no use in this setting because the company does not maintain a shared secret key with each of its customers.

Applications of digital signatures

The signing process works as follows:

- The company generates a secret signing key sk along with some corresponding public key denoted pk and keeps the secret key sk to itself.
- To sign a software update file U, the company runs a signing algorithm S that takes (sk; U) as input and outputs a short signature σ.
- The company then ships the pair (U; σ) to all its customers.
- A customer given the update (U; σ) and the public key pk, checks validity of this message signature pair using a signature verification algorithm V that takes (pk;U; σ) as input.

Applications of digital signatures

Authenticated email:

- Suppose Bob receives an email claiming to be from his friend Alice. Bob wants to verify that the email really is from Alice. A MAC system would do the job but requires that Alice and Bob have a shared secret key. What if they never met before and do not share a secret key?
- Digital signatures provide a simple solution.
- First, Alice generates a public/secret key pair (pk; sk). When sending an email m to Bob, Alice generates a signature σ on m derived using her secret key. She then sends (m; σ) to Bob.
- Bob receives (m; σ) and verifies that m is from Alice in two steps. First, Bob retrieves Alice's public key pk. Second, Bob runs the signature verification algorithm on the triple (pk; m; σ).

Applications of digital signatures

Certificates:

- We could assume that public keys are obtained from a read-only public directory. In practice, however, there is no public directory. Instead, Alice's public key pk is certified by some third party called a *certificate authority* or CA for short.

Applications of digital signatures

To generate a certified public key:

- Alice first generates a public/private key pair (pk; sk) and presents her public key pk to the CA. The CA then verifies that Alice is who she claims to be.
- The CA signs the message m using its own secret key sk_{CA} and sends the pair *Cert* := (m; σ_{CA}) back to Alice. This pair *Cert* is called a **certificate** or pk.
- Bob obtains Alice's certificate from Alice and verifies the CA's signature in the certificate. If the signature is valid, Bob has some confidence that pk is Alice's public key.

Applications of digital signatures

Non-repudiation:

- An interesting property of the authenticated email system above is that Bob now has evidence that the message m is from Alice.
- He could show the pair (m; σ) to a judge who could also verify Alice's signature.
- This property provided by digital signatures is called **non-repudiation**.