Public Key encryption

Presentation by:
V. Balasubramanian
SSN College of Engineering

Objectives

- Public-Key Encryption – An Overview -- Definitions
- Hybrid encryption
- RSA encryption
- The El Gamal Encryption Scheme
- Security Against Chosen-Ciphertext Attacks;

Objectives

- Digital Signatures Schemes
- RSA Signatures -- The Hash-and-Sign Paradigm
- Lamport's One-Time Signature Scheme
- Signatures from Collision – Resistant Hashing
- The Digital Signature Standard
- Certificates and Public-Key Infrastructures;
- Authentication Protocol: SSL and TLS

Introduction

Symmetric Algorithms

- Encryption and Decryption use the same key
- i.e. $K_E = K_D$
- Examples:
  - Block Ciphers : DES, AES, PRESENT, etc.
  - Stream Ciphers : A5, Grain, etc.

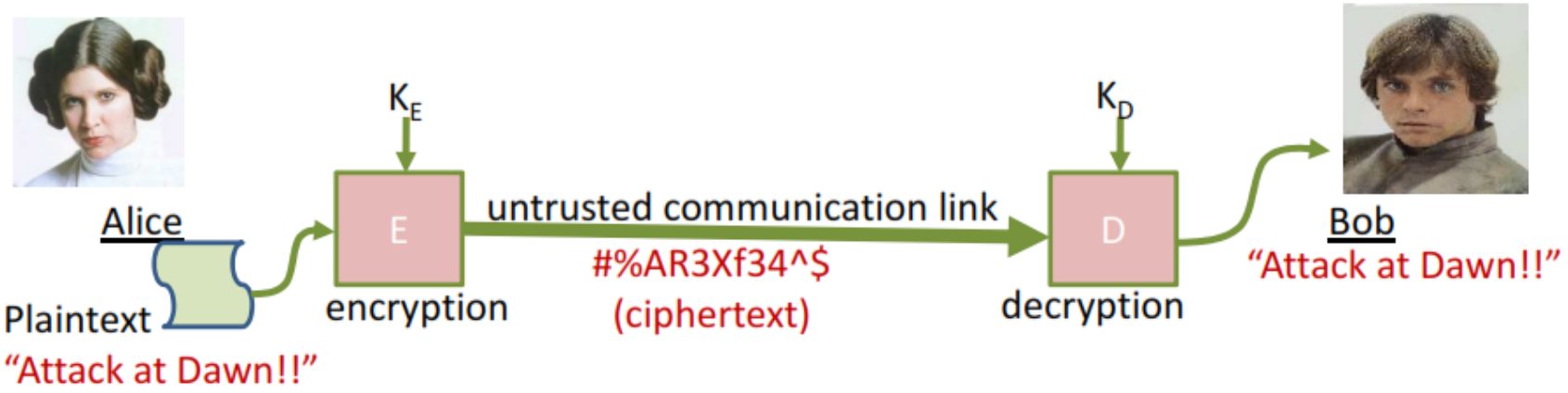
Asymmetric Algorithms

- Encryption and Decryption keys are different
- $K_E \neq K_D$
- Examples:
  - RSA
  - ECC

Public Key Basic Terminology

- Plaintext/Plaintext Space
  - A message m $\in \mathcal{M}$
- Ciphertext c $\in \mathcal{C}$
- **Public/Private Key Pair $(pk, sk) \in \mathcal{K}$**

Asymmetric Key



The Key K is a secret

Encryption Key K_E not same as decryption key K_D

K_E known as Bob's public key;
K_D is Bob's private key

Advantage : No need of secure key exchange between Alice and Bob

Asymmetric key algorithms based on trapdoor one-way functions

OWF

Easy to compute in one direction

Once done, it is difficult to inverse

Press to lock
(can be easily done)

Once locked it is difficult to unlock without a key

Trap door Function

- One way function with a trapdoor
- Trapdoor is a special function that if possessed can be used to easily invert the one way

Locked
(difficult to unlock)

Easily Unlocked

PKI Analogy

- Alice puts message into box and locks it
- Only Bob, who has the key to the lock can open it and read the message

Public Key Encryption Syntax

- Three Algorithms
  - $\mathrm{Gen}(1^n, R)$ (Key-generation algorithm)
    - Input: Random Bits R
    - Output: $(\boldsymbol{pk}, \boldsymbol{sk}) \in \mathcal{K}$
  - $\mathrm{Enc}_{\mathrm{pk}}(m) \in \mathcal{C}$ (Encryption algorithm)
  - $\mathrm{Dec}_{\mathrm{sk}}(c)$ (Decryption algorithm)
    - Input: Secret key sk and a ciphertex c
    - Output: a plaintext message $\mathrm{m} \in \mathcal{M}$

- **Invariant**: $\mathrm{Dec}_{\mathrm{sk}}(\mathrm{Enc}_{\mathrm{pk}}(\mathrm{m}))=\mathrm{m}$

Pk & Sk

Alice must run key generation algorithm in advance an publishes the public key: pk

Assumption: Adversary only gets to see pk (not sk)

Comparison

Public-key encryption addresses (to some extent) the *key-distribution problem*, since communicating parties do not need to secretly share a key in advance of their communication. Two parties can communicate secretly even if *all* communication between them is monitored.

- To avoid use a trusted KDC we could have every pair of users exchange private keys
- How many private keys per person?
  - **Answer**: n-1
  - Need to meet up with n-1 different users in person!
- Key Explosion Problem
  - n can get very big if you are Google or Amazon!

Drawbacks

- The main disadvantage of public-key encryption is that it is roughly 2 to 3 orders of magnitude slower than private-key encryption.
- 1
- It can be a challenge to implement public-key encryption in severely resource-constrained devices like smartcards or radio-frequency identification (RFID) tags.

Public Key Encryption

DEFINITION 11.1 *A* public-key encryption scheme *is a triple of probabilistic polynomial-time algorithms* (Gen, Enc, Dec) *such that:*

1. *The* key-generation algorithm Gen *takes as input the security parameter* 1^n *and outputs a pair of keys* (pk, sk). *We refer to the first of these as the* public key *and the second as the* private key. *We assume for convenience that* pk *and* sk *each has length at least* n, *and that* n *can be determined from* pk, sk.

2. *The* encryption algorithm Enc *takes as input a public key* pk *and a message* m *from some message space (that may depend on* pk*). It outputs a ciphertext* c, *and we write this as* $c \leftarrow \mathsf{Enc}_{pk}(m)$. *(Looking ahead,* Enc *will need to be probabilistic to achieve meaningful security.)*

3. *The deterministic* decryption algorithm Dec *takes as input a private key* sk *and a ciphertext* c, *and outputs a message* m *or a special symbol* $\perp$ *denoting failure. We write this as* $m := \mathsf{Dec}_{sk}(c)$.

It is required that, except possibly with negligible probability over (pk, sk) *output by* $\mathsf{Gen}(1^n)$, *we have* $\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m)) = m$ *for any (legal) message* m.

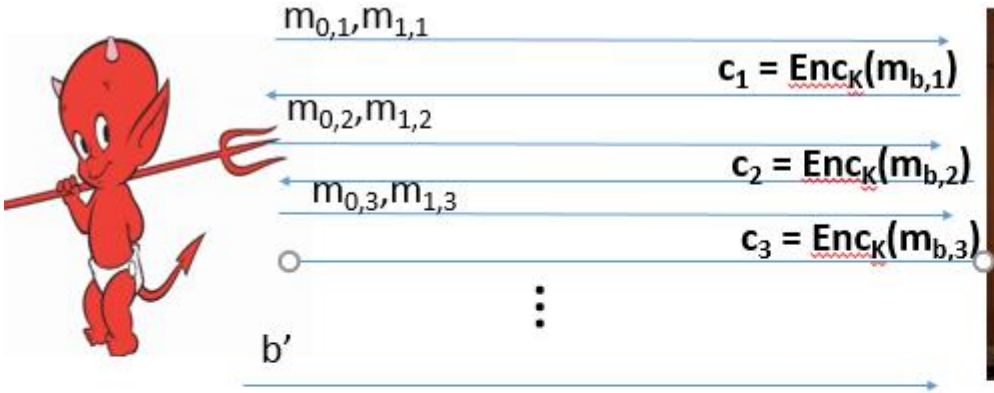
Chosen Plaintext Attack

- Model ability of adversary to control or influence what the honest parties encrypt.

- Historical Example: Battle of Midway (WWII).
  - US Navy cryptanalysts were able to break Japanese code by tricking Japanese navy into encrypting a particular message

- Private Key Cryptography

CPA



Random bit b

K = Gen(.)

$$\forall PPT\ A\ \exists \mu\ (\text{negligible})\ \text{s.t}$$

$$\Pr[A\ \ Guesses\ b' = b] \leq \frac{1}{2} + \mu(n)$$

EAV Secure (indistinguishable encryptions against eavesdroppers)

The eavesdropping indistinguishability experiment
$\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$:

1. $\mathsf{Gen}(1^n)$ is run to generate keys (pk, sk).
2. $\mathcal{A}$ is given pk and outputs valid messages m_0, m_1 of the same length.
3. $b \in \{0,1\}$ is chosen uniformly at random and $c \leftarrow \mathsf{Enc}_{pk}(m_b)$ is given to $\mathcal{A}$.
4. $\mathcal{A}$ outputs a bit b'. The outcome of the experiment is 1 if $b = b'$ and 0 otherwise.

Definition (11.2)

A public-key encryption scheme Π is *EAV-secure* if for every ppt $\mathcal{A}$,

$$\Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n).$$

CPA Security

Proposition (11.3)

If a public key encryption scheme is EAV-secure, then it is CPA-secure.

Theorem (11.4)

No deterministic public key encryption scheme is CPA-secure.

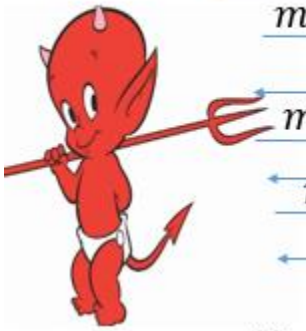
Chosen-Plaintext Attacks

- Model ability of adversary to control or influence what the honest parties encrypt.

- Private Key Crypto
  - Attacker tricks victim into encrypting particular messages

- Public Key Cryptography
  - The attacker already has the public key pk
  - Can encrypt any message s/he wants!
  - CPA Security is critical!

CPA



Public Key: pk

m_0^1, m_1^1

$c_1 = \mathrm{Enc}_{pk}(m_b^1)$

m_0^2, m_1^2

$c_2 = \mathrm{Enc}_{pk}(m_b^2)$

m_0^3, m_1^3

$c_3 = \mathrm{Enc}_{pk}(m_b^3)$

$\vdots$

b'

Random bit b

(pk,sk) = Gen(.)

$$\forall PPT\ A\ \exists \mu\ (\text{negligible})\ \text{s.t}$$

$$\Pr\left[\mathrm{PubK}_{A,\Pi}^{\mathrm{LR-cpa}}(n) = 1\right] \leq \frac{1}{2} + \mu(n)$$

Public Key Cryptography

- **Fact 1:** CPA Security and Eavesdropping Security are Equivalent
  - Key Insight: The attacker has the public key so he doesn't gain anything from being able to query the encryption oracle!
- **Fact 2:** Any deterministic encryption scheme is not CPA-Secure
  - Historically overlooked in many real world public key crypto systems
- **Fact 3:** Plain RSA is **not** CPA-Secure
- **Fact 4:** No Public Key Cryptosystem can achieve Perfect Secrecy!

Insecurity

In contrast to the private-key setting, however, perfectly secret public-key encryption is *impossible*, regardless of how long the keys are or how small the message space is. In fact, an unbounded adversary given pk and a ciphertext c computed via $c \leftarrow \mathsf{Enc}_{pk}(m)$ can determine m with probability 1. A proof of

THEOREM 11.4 *No deterministic public-key encryption scheme is CPA-secure.*

a professor encrypting students' grades. Here, an eavesdropper knows that each student's grade must be one of $\{A, B, C, D, F\}$. If the professor uses a deterministic public-key encryption scheme, an eavesdropper can quickly determine any student's actual grade by encrypting all possible grades and comparing the result to the given ciphertext.

Probablistic

- When public-key encryption was introduced, it is fair to say that the importance of probabilistic encryption was not yet fully realized. The seminal work

Multiple Encryptions

attacker is given access to a "left-or-right" oracle $\mathsf{LR}_{pk,b}$ that, on input a pair of equal-length messages m_0, m_1, computes the ciphertext $c \leftarrow \mathsf{Enc}_{pk}(m_b)$ and returns c. The attacker is allowed to query this oracle as many times as it likes, and the definition therefore models security when multiple (unknown) messages are encrypted using the same public key.

Formally, consider the following experiment defined for a public-key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and adversary $\mathcal{A}$:

The LR-oracle experiment $\mathsf{PubK}^{\mathsf{LR\text{-}cpa}}_{\mathcal{A},\Pi}(n)$**:**

1. $\mathsf{Gen}(1^n)$ *is run to obtain keys* (pk, sk).
2. *A uniform bit* $b \in \{0,1\}$ *is chosen.*
3. *The adversary* $\mathcal{A}$ *is given input* pk *and oracle access to* $\mathsf{LR}_{pk,b}(\cdot,\cdot)$.
4. *The adversary* $\mathcal{A}$ *outputs a bit* b'.
5. *The output of the experiment is defined to be* 1 *if* $b' = b$, *and* 0 *otherwise. If* $\mathsf{PubK}^{\mathsf{LR\text{-}cpa}}_{\mathcal{A},\Pi}(n) = 1$, *we say that* $\mathcal{A}$ succeeds.

DEFINITION 11.5 *A public-key encryption scheme* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *has* indistinguishable multiple encryptions *if for all probabilistic polynomial-time adversaries* $\mathcal{A}$ *there exists a negligible function* negl *such that:*

$$\Pr[\mathsf{PubK}^{\mathsf{LR\text{-}cpa}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n).$$

THEOREM 11.6 *If public-key encryption scheme* Π *is CPA-secure, then it also has indistinguishable* multiple *encryptions.*

Chosen Cipher Text attacks

- Models ability of attacker to obtain (partial) decryption of selected ciphertexts
- Attacker might intercept ciphertext c (sent from S to R) and send c’ instead.
  - After that attacker can observe receiver’s behavior (abort, reply etc…)
- Attacker might send a modified ciphertext c’ to receiver R in his own name.
  - E-mail response: Receiver might decrypt c’ to obtain m’ and include m’ in the response to the attacker

CCA

$\mathcal{A}$ might send a modified ciphertext c' to $\mathcal{R}$ *on behalf of* $\mathcal{S}$. (For example, in the context of encrypted e-mail, $\mathcal{A}$ might construct an encrypted e-mail c' and forge the "`From`" field so that it appears the e-mail originated from $\mathcal{S}$.) In this case, although it is unlikely that $\mathcal{A}$ would be able to obtain the entire decryption m' of c', it might be possible for $\mathcal{A}$ to infer some information about m' based on the subsequent behavior of $\mathcal{R}$. Based on this information, $\mathcal{A}$ might be able to learn something about the original message m.

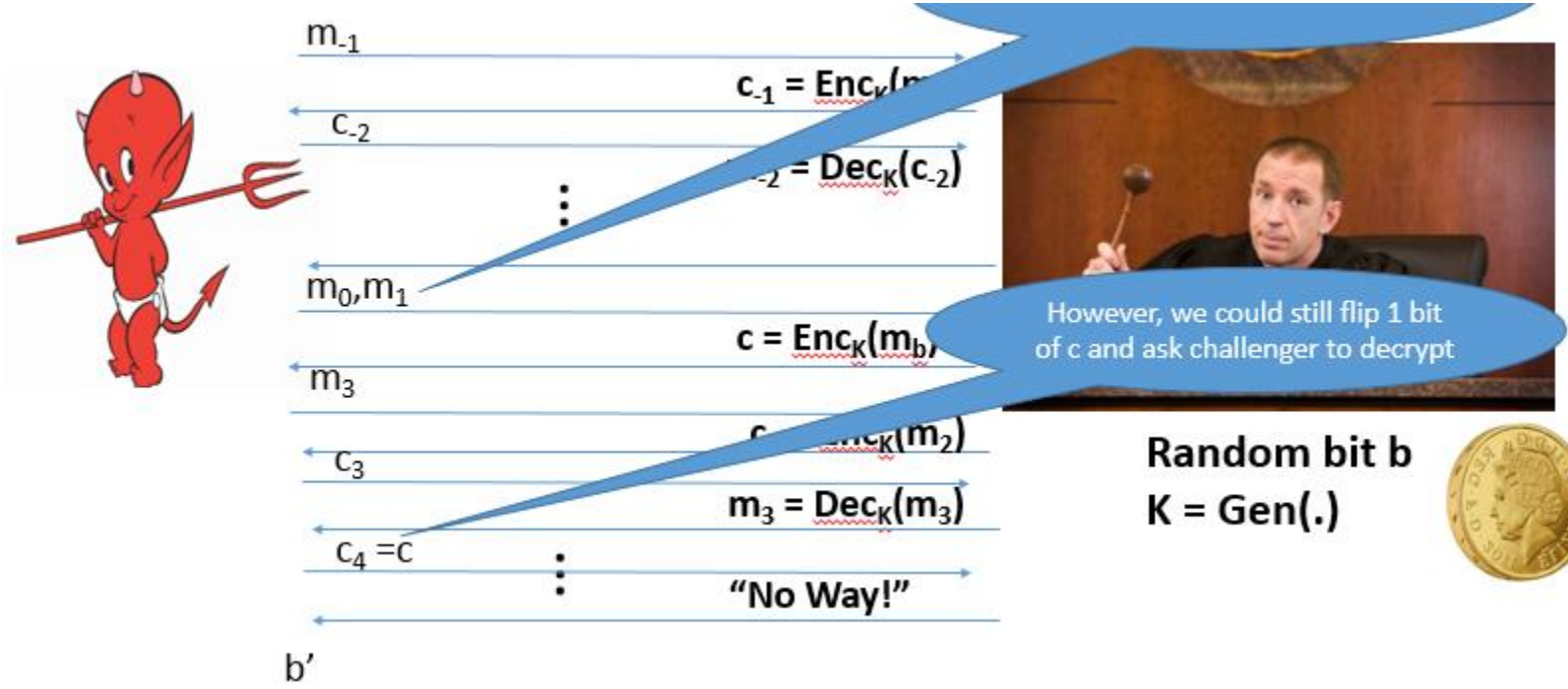
$\mathcal{A}$ might send a modified ciphertext c' to $\mathcal{R}$ *in its own name*. In this case, $\mathcal{A}$ might obtain the entire decryption m' of c' if $\mathcal{R}$ responds directly to $\mathcal{A}$. Even if $\mathcal{A}$ learns nothing about m', this modified message may have a known *relation* to the original message m that can be exploited by $\mathcal{A}$; see the third scenario below for an example.

CCA

The CCA indistinguishability experiment $\mathsf{PubK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n)$:

1. $\mathsf{Gen}(1^n)$ *is run to obtain keys* (pk, sk).
2. *The adversary* $\mathcal{A}$ *is given* pk *and access to a decryption oracle* $\mathsf{Dec}_{sk}(\cdot)$. *It outputs a pair of messages* m_0, m_1 *of the same length. (These messages must be in the message space associated with* pk.)
3. *A uniform bit* $b \in \{0,1\}$ *is chosen, and then a ciphertext* $c \leftarrow \mathsf{Enc}_{pk}(m_b)$ *is computed and given to* $\mathcal{A}$.
4. $\mathcal{A}$ *continues to interact with the decryption oracle, but may not request a decryption of* c *itself. Finally,* $\mathcal{A}$ *outputs a bit* b'.
5. *The output of the experiment is defined to be* 1 *if* $b' = b$, *and* 0 *otherwise.*

CCA


m_{-1}
$c_{-1} = Enc_K(m$
c_{-2}
$= Dec_K(c_{-2})$
m_0, m_1
$c = Enc_K(m_b)$
m_3
c $Enc_K(m_2)$
c_3
$m_3 = Dec_K(m_3)$
$c_4 = c$
"No Way!"
b'
However, we could still flip 1 bit of c and ask challenger to decrypt
Random bit b
K = Gen(.)

CCA

DEFINITION 11.8 *A public-key encryption scheme* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *has* indistinguishable encryptions under a chosen-ciphertext attack (*or is* CCA-secure) *if for all probabilistic polynomial-time adversaries* $\mathcal{A}$ *there exists a negligible function* negl *such that*

$$\Pr[\mathsf{PubK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n).$$

Hybrid Encryption

Private key schemes are more efficient than public key ones.

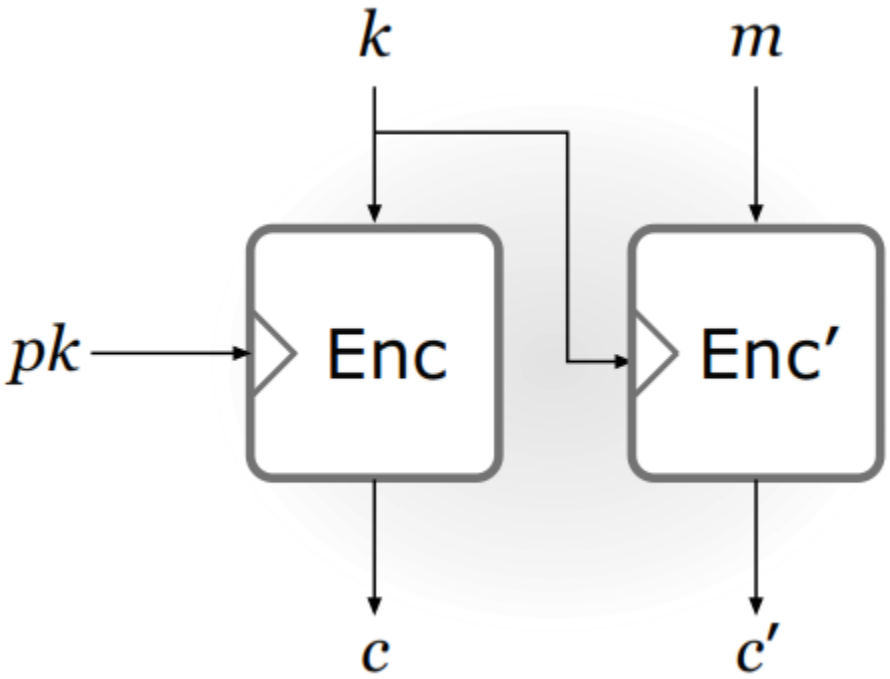
Hyprid approach:

- Choose a key k uniformly at random.
- Encrypt k using pk and send it to Bob.
- Encrypt the message using k and send it to Bob

This is a full-fledged public key scheme, since k is not shared in advance.

Uses a two-stage approach.

Hybrid Scheme



Hybrid Encryption

practice. The basic idea is to use public-key encryption to obtain a shared key k, and then encrypt the message m using a private-key encryption scheme and key k. The receiver uses its long-term (asymmetric) private key to derive k, and then uses private-key decryption (with key k) to recover the original message. We stress that although private-key encryption is used as a component, this is a full-fledged public-key encryption scheme by virtue of the fact that the sender and receiver do not share any secret key *in advance*.

Key Encapsulation Mechanism

Definition (11.9)

A *key-encapsulation mechanism (KEM)* is a triple of ppt algorithms $(\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ such that

1. Gen on input 1^n outputs a pair of keys (pk, sk). Both have length at least n and n can be determined from pk.
2. Encaps on input pk and 1^n outputs a ciphertext c and a key $k \in \{0,1\}^{\ell(n)}$, where ℓ is the key length. We write $(c, k) \leftarrow \mathsf{Encaps}_{pk}(1^n)$.
3. Decaps is deterministic and takes as input sk and c and outputs a key k or $\perp$. We write $k := \mathsf{Decaps}_{sk}(c)$.

KEM

- Sender runs $\text{Encaps}_{pk}(1^n)$ to obtain c and k.
- Then uses a private key encryption scheme ("data encapsulation mechanism") to encrypt m using k yielding c'
- He then sends c and c'.

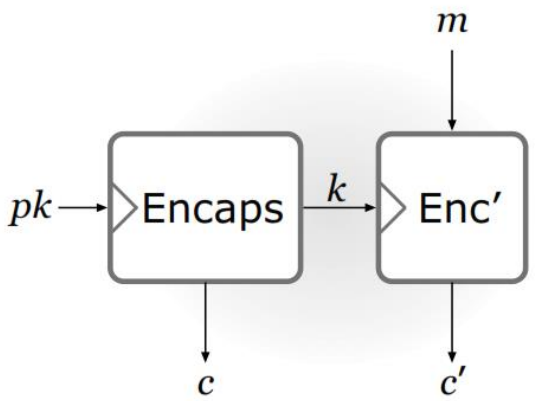
Cost

What is the message length? Let

- α denote the cost of encapsulating an n-bit string and
- β be the cost per bit to encrypt m

Total time per bit:

$$\frac{\alpha + \beta|m|}{|m|} = \frac{\alpha}{|m|} + \beta.$$



Key Encapsulation

Let $\Pi = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ be a KEM with key length n, and let $\Pi' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ be a private-key encryption scheme. Construct a public-key encryption scheme $\Pi^{\mathsf{hy}} = (\mathsf{Gen}^{\mathsf{hy}}, \mathsf{Enc}^{\mathsf{hy}}, \mathsf{Dec}^{\mathsf{hy}})$ as follows:

- $\mathsf{Gen}^{\mathsf{hy}}$: on input 1^n run $\mathsf{Gen}(1^n)$ and use the public and private keys (pk, sk) that are output.
- $\mathsf{Enc}^{\mathsf{hy}}$: on input a public key pk and a message $m \in \{0,1\}^*$ do:
  1. Compute $(c, k) \leftarrow \mathsf{Encaps}_{pk}(1^n)$.
  2. Compute $c' \leftarrow \mathsf{Enc}'_k(m)$.
  3. Output the ciphertext $\langle c, c' \rangle$.
- $\mathsf{Dec}^{\mathsf{hy}}$: on input a private key sk and a ciphertext $\langle c, c' \rangle$ do:
  1. Compute $k := \mathsf{Decaps}_{sk}(c)$.
  2. Output the message $m := \mathsf{Dec}'_k(c')$.

CPA SEcurity

Let $\Pi = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ be a KEM and $\mathcal{A}$ an arbitrary adversary.

The CPA indistinguishability experiment $\mathsf{KEM}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n)$**:**

1. $\mathsf{Gen}(1^n)$ *is run to obtain keys* (pk, sk). *Then* $\mathsf{Encaps}_{pk}(1^n)$ *is run to generate* (c, k) *with* $k \in \{0,1\}^n$.
2. *A uniform bit* $b \in \{0,1\}$ *is chosen. If* $b = 0$ *set* $\hat{k} := k$. *If* $b = 1$ *then choose a uniform* $\hat{k} \in \{0,1\}^n$.
3. *Give* $(pk, c, \hat{k})$ *to* $\mathcal{A}$, *who outputs a bit* b'. *The output of the experiment is defined to be* 1 *if* $b' = b$, *and* 0 *otherwise.*

CPA Security

A KEM Π is CPA-secure if for all ppt $\mathcal{A}$,

$$\Pr[\mathsf{KEM}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n).$$

CCA Security

Let $\Pi = (\text{Gen}, \text{Encaps}, \text{Decaps})$ be a KEM:

The CCA indistinguishability experiment $\text{KEM}^{\text{cca}}_{\mathcal{A},\Pi}(n)$**:**

1. $\text{Gen}(1^n)$ is run to obtain keys (pk, sk). Then $\text{Encaps}_{pk}(1^n)$ is run to generate (c, k) with $k \in \{0,1\}^n$
2. Choose $b \in \{0,1\}$ uniformly at random. If $b = 0$ set $\hat{k} := k$ Else choose $\hat{k} \in \{0,1\}^n$ uniformly at random.
3. Give $(pk, c, \hat{k})$ to $\mathcal{A}$ and access to the oracle $\text{Decaps}_{sk}(.)$. $\mathcal{A}$ may not query c itself.
4. $\mathcal{A}$ outputs a bit b'. The outcome of the experiment is 1 if $b = b'$ and 0 otherwise.

CCA Secure

Definition (11.13)

A KEM Π is CCA-secure if for all ppt $\mathcal{A}$,

$$\Pr[\mathsf{KEM}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n).$$

Theorem (11.14)

If Π is a CCA-secure KEM and Π' is a CCA-secure private key encryption scheme, then Π^{hy} (Construction 11.10) is a CCA-secure public key encryption scheme.

Discrete Log Problem

Let $(G,\cdot)$ be a group

Let $\alpha \in G$ be a primitive element in the group with order n

Define the set

$$\langle \alpha \rangle = \{\alpha^i : 0 \le i \le n-1\}$$

For any unique integer a $(0 \le a \le n-1)$,

let $\alpha^a = \beta$

Denote $a = \log_\alpha \beta$ as the discrete logarithm of β

Given α and a, it is easy to compute β
Given α and β it is computationally difficult to determine what a was

El Gamal algorithm

- In 1985, Taher El Gamal observed that the Diffie–Hellman key-exchange protocol could be adapted to give a public-key encryption scheme.

Elgamal Public key Cryptosystem

- Fix a prime p (and group Z_p)
- Let $\alpha \in Z_p$ be a primitive element
- Choose a secret 'a' and compute $\beta \equiv \alpha^a \bmod p$

Public keys : α, β, p

Private key : a

Encryption

$choose\ \ a\ random\ (\sec ret)\ k \leftarrow Z_p$

$e_k(x) = (y_1, y_2)$

$where\ \ y_1 = \alpha^k \bmod p,$

$y_2 = x \cdot \beta^k \bmod p$

Decryption

$$d_k(x) = y_2 (y_1^a)^{-1} \bmod p$$
$$= x \cdot \beta^k (\alpha^{ka})^{-1} \bmod p$$
$$= x \cdot \alpha^{ka} (\alpha^{ka})^{-1} \bmod p$$
$$\equiv x$$

Illustration

- $p = 2579$, $\alpha = 2$ (α is a primitive element mod p)
- Choose a random $a = 765$
- Compute $\beta \equiv 2^{765} \bmod 2579$

Encryption of message x = 1299

choose a random key $k = 853$

$y_1 = 2^{853} \bmod 2579 = 435$

$y_2 = 1299 \times 949^{853} = 2396$

Decryption of cipher (435, 2396)

$2396 \times (435^{765})^{-1} \bmod p$

$= 1299$

Mathematical Model

Let $\mathcal{G}$ be as in the text. Define a public-key encryption scheme as follows:

- Gen: on input 1^n run $\mathcal{G}(1^n)$ to obtain $(\mathbb{G}, q, g)$. Then choose a uniform $x \in \mathbb{Z}_q$ and compute $h := g^x$. The public key is $\langle \mathbb{G}, q, g, h \rangle$ and the private key is $\langle \mathbb{G}, q, g, x \rangle$. The message space is $\mathbb{G}$.
- Enc: on input a public key $pk = \langle \mathbb{G}, q, g, h \rangle$ and a message $m \in \mathbb{G}$, choose a uniform $y \in \mathbb{Z}_q$ and output the ciphertext

$$\langle g^y,\ h^y \cdot m \rangle.$$

- Dec: on input a private key $sk = \langle \mathbb{G}, q, g, x \rangle$ and a ciphertext $\langle c_1, c_2 \rangle$, output

$$\hat{m} := c_2 / c_1^x.$$

Proof

To see that decryption succeeds, let $\langle c_1, c_2 \rangle = \langle g^y,\ h^y \cdot m \rangle$ with $h = g^x$. Then

$$\hat{m} = \frac{c_2}{c_1^x} = \frac{h^y \cdot m}{(g^y)^x} = \frac{(g^x)^y \cdot m}{g^{xy}} = \frac{g^{xy} \cdot m}{g^{xy}} = m.$$

Example

- $p = 83, q = 2 * p + 1 = 167$

$g = generator = 2^2 mod\ 167 = 4$

$private\ key$:

$select\ x \in \mathrm{Z}_q \in Z_{83}, x = 37$

Public key

$p_k =< p, q, g, h > =< 83,167,4,4^{37} \mathrm{mod} 167 >$

$p_k = \{83,167,4,76\}$

Encryption

- $message\ m = 65.$
- $select\ y \in Z_{167}, y = 71.$
- $cipher\ text = \{4^{71} \bmod 167, 76^{71} \bmod 167\} = \{132,44\}$

Enc: on input a public key $pk = \langle \mathbb{G}, q, g, h \rangle$ and a message $m \in \mathbb{G}$, choose a uniform $y \in \mathbb{Z}_q$ and output the ciphertext

$$\langle g^y,\ h^y \cdot m \rangle.$$

Decryption

To decrypt, the receiver first computes $124 = [132^{37} \bmod 167]$; then, since $66 = [124^{-1} \bmod 167]$, the receiver recovers $m = 65 = [44 \cdot 66 \bmod 167]$. $\diamondsuit$

Dec: on input a private key $sk = \langle \mathbb{G}, q, g, x \rangle$ and a ciphertext $\langle c_1, c_2 \rangle$, output

$$\hat{m} := c_2 / c_1^x.$$

RSA Algorithm

RSA key generation GenRSA

Input: Security parameter 1^n
Output: N, e, d as described in the text

$(N, p, q) \leftarrow \mathsf{GenModulus}(1^n)$
$\phi(N) := (p-1)(q-1)$
choose $e > 1$ such that $\gcd(e, \phi(N)) = 1$
compute $d := [e^{-1} \bmod \phi(N)]$
return N, e, d

RSA Encryption

Let GenRSA be as in the text. Define a public-key encryption scheme as follows:

- Gen: on input 1^n run GenRSA(1^n) to obtain N, e, and d. The public key is $\langle N, e \rangle$ and the private key is $\langle N, d \rangle$.
- Enc: on input a public key $pk = \langle N, e \rangle$ and a message $m \in \mathbb{Z}_N^*$, compute the ciphertext
$$c := [m^e \bmod N].$$
- Dec: on input a private key $sk = \langle N, d \rangle$ and a ciphertext $c \in \mathbb{Z}_N^*$, compute the message
$$m := [c^d \bmod N].$$

Illustration

Say GenRSA outputs $(N, e, d) = (391, 3, 235)$. (Note that $391 = 17 \cdot 23$ and so $\phi(391) = 16 \cdot 22 = 352$. Moreover, $3 \cdot 235 = 1 \bmod 352$.) So the public key is $(391, 3)$ and the private key is $(391, 235)$.

To encrypt the message $m = 158 \in \mathbb{Z}_{391}^*$ using the public key $(391, 3)$, we simply compute $c := [158^3 \bmod 391] = 295$; this is the ciphertext. To decrypt, the receiver computes $[295^{235} \bmod 391] = 158$. $\diamondsuit$

RSA

- Karatsuba multiplication

Plain RSA

N, e

c

$(N, e, d) \leftarrow \underline{RSAGen}(1^n)$

$\underline{pk} = (N, e)$

$\underline{sk} = d$

$m = [c^d \bmod N]$

$c = [m^e \bmod N]$

Security

This scheme is *deterministic*

- Cannot be CPA-secure!

RSA assumption only refers to hardness of computing the e^{th} roots of *uniform* c

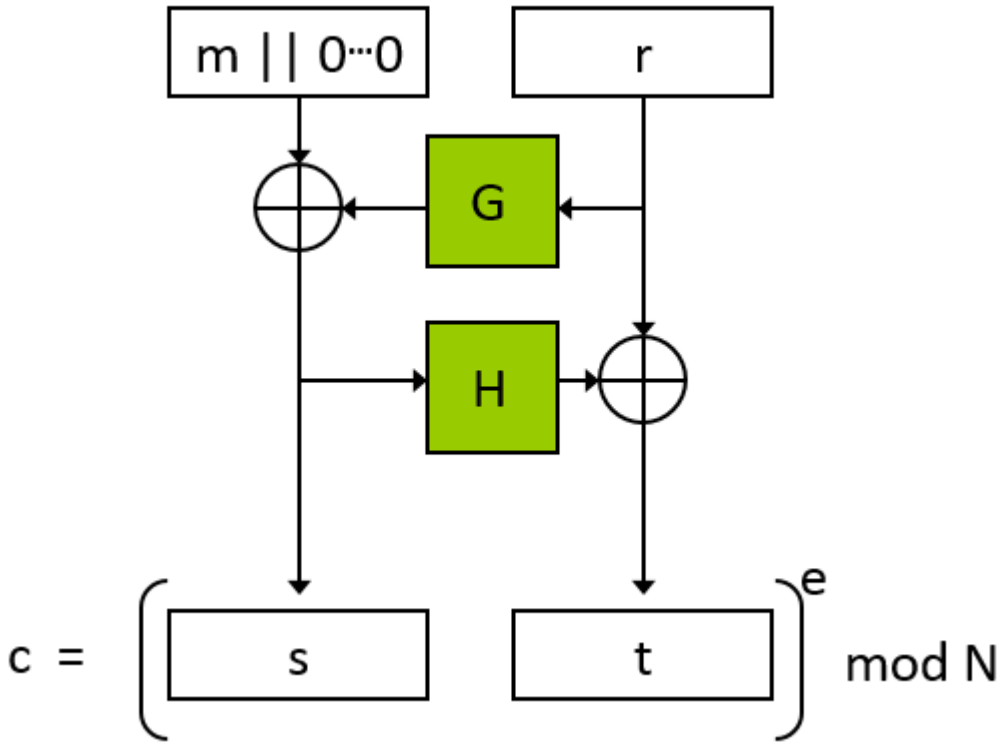
- c is not uniform unless m is
- Easy to recover "small" m from c

Plain RSA should never be used!

RSA assumption only refers to hardness of computing the e^{th} root of c *in its entirety*

- *Partial* information about the e^{th} root may be leaked

Optimal asymmetric encryption padding



Rivest Shamir Adelman

Introduction

- After Whitfield Diffie and Martin Hellman introduced public-key cryptography in their landmark 1976 paper, a new branch of cryptography suddenly opened up.
- As a consequence, cryptologists started looking for methods with which public key encryption could be realized.
- In 1977, Ronald Rivest, Adi Shamir and Leonard Adleman proposed a scheme which became the most widely used asymmetric cryptographic scheme, RSA

Algebraic Structures

Two Algebraic Structures

RSA uses two algebraic structures: a ring and a group.

Encryption/Decryption Ring Encryption and decryption are done using the commutative ring $\mathbf{R} = \langle \mathbf{Z}_n, +, \times \rangle$ with two arithmetic operations: addition and multiplication. In RSA, this ring is public because the modulus n is public. Anyone can send a message to Bob using this ring to do encryption.

Key-Generation Group RSA uses a multiplicative group $\mathbf{G} = \langle \mathbf{Z}_{\phi(n)}^*, \times \rangle$ for key generation. This group supports only multiplication and division (using multiplicative inverses), which are needed for generating public and private keys. This group is hidden from the public because its modulus, $\phi(n)$, is hidden from the public. We will see shortly that if Eve can find this modulus, she can easily attack the cryptosystem.

RSA uses two algebraic structures:
a public ring $\mathbf{R} = \langle \mathbf{Z}_n, +, \times \rangle$ and a private group $\mathbf{G} = \langle \mathbf{Z}_{\phi(n)}^*, \times \rangle$.

Key Generation

RSA_Key_Generation

{

Select two large primes p and q such that $p \neq q$.

$n \leftarrow p \times q$

$\phi(n) \leftarrow (p-1) \times (q-1)$

Select e such that $1 < e < \phi(n)$ and e is coprime to $\phi(n)$

$d \leftarrow e^{-1} \bmod \phi(n)$ // d is inverse of e modulo $\phi(n)$

Public_key $\leftarrow$ (e, n) // To be announced publicly

Private_key $\leftarrow d$ // To be kept secret

return Public_key and Private_key

}

Encryption

```
RSA_Encryption (P, e, n)                    // P is the plaintext in Z_n and P < n
{
   C ← Fast_Exponentiation (P, e, n)        // Calculation of (P^e mod n)
   return C
}
```

RSA Encryption Given the public key $(n, e) = k_{pub}$ and the plaintext x, the encryption function is:

$$y = e_{k_{pub}}(x) \equiv x^e \bmod n \tag{7.1}$$

where $x, y \in \mathbb{Z}_n$.

Decryption

```
RSA_Decryption (C, d, n)                    //C is the ciphertext in Z_n
{
    P ← Fast_Exponentiation (C, d, n)       // Calculation of (C^d mod n)
    return P
}
```

In RSA, p and q must be at least 512 bits; n must be at least 1024 bits.

RSA Decryption Given the private key $d = k_{pr}$ and the ciphertext y, the decryption function is:

$$x = d_{k_{pr}}(y) \equiv y^d \bmod n \qquad (7.2)$$

where $x, y \in \mathbb{Z}_n$.

Proof

Proof of RSA

We can prove that encryption and decryption are inverses of each other using the second version of Euler's theorem discussed in Chapter 9:

If $n = p \times q$, $a < n$, and k is an integer, then $a^{k \times \phi(n)+1} \equiv a \pmod{n}$.

Assume that the plaintext retrieved by Bob is P_1 and prove that it is equal to P.

$$P_1 = C^d \bmod n = (P^e \bmod n)^d \bmod n = P^{ed} \bmod n$$
$$ed = k\phi(n) + 1 \qquad \text{// } d \text{ and } e \text{ are inverses modulo } \phi(n)$$
$$P_1 = P^{ed} \bmod n \rightarrow P_1 = P^{k\phi(n)+1} \bmod n$$
$$P_1 = P^{k\phi(n)+1} \bmod n = P \bmod n \qquad \text{// Euler's theorem (second version)}$$

Example

Example 10.5

Bob chooses 7 and 11 as p and q and calculates $n = 7 \times 11 = 77$. The value of $\phi(n) = (7 - 1)(11 - 1)$ or 60. Now he chooses two exponents, e and d, from $\mathbf{Z}_{60}^*$. If he chooses e to be 13, then d is 37. Note that $e \times d \bmod 60 = 1$ (they are inverses of each other). Now imagine that Alice wants to send the plaintext 5 to Bob. She uses the public exponent 13 to encrypt 5.

Plaintext: 5	$C = 5^{13} = 26 \bmod 77$	Ciphertext: 26

Bob receives the ciphertext 26 and uses the private key 37 to decipher the ciphertext:

Ciphertext: 26	$P = 26^{37} = 5 \bmod 77$	Plaintext: 5

The plaintext 5 sent by Alice is received as plaintext 5 by Bob.

Illustration

Alice

message $x = 4$

Bob

1. choose $p = 3$ and $q = 11$
2. $n = p \cdot q = 33$
3. $\Phi(n) = (3-1)(11-1) = 20$
4. choose $e = 3$
5. $d \equiv e^{-1} \equiv 7 \bmod 20$

$k_{pub}=(33,3)$ ←

$y = x^e \equiv 4^3 \equiv 31 \bmod 33$

$y=31$ →

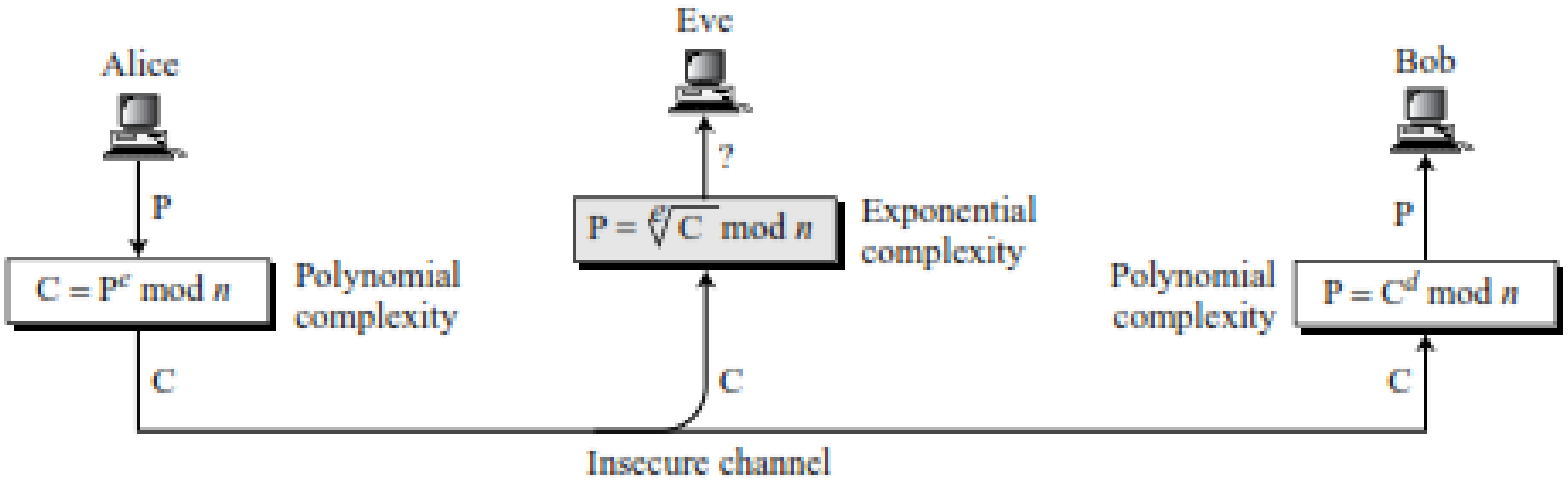
$y^d = 31^7 \equiv 4 = x \bmod 33$

Fast Exponentiation

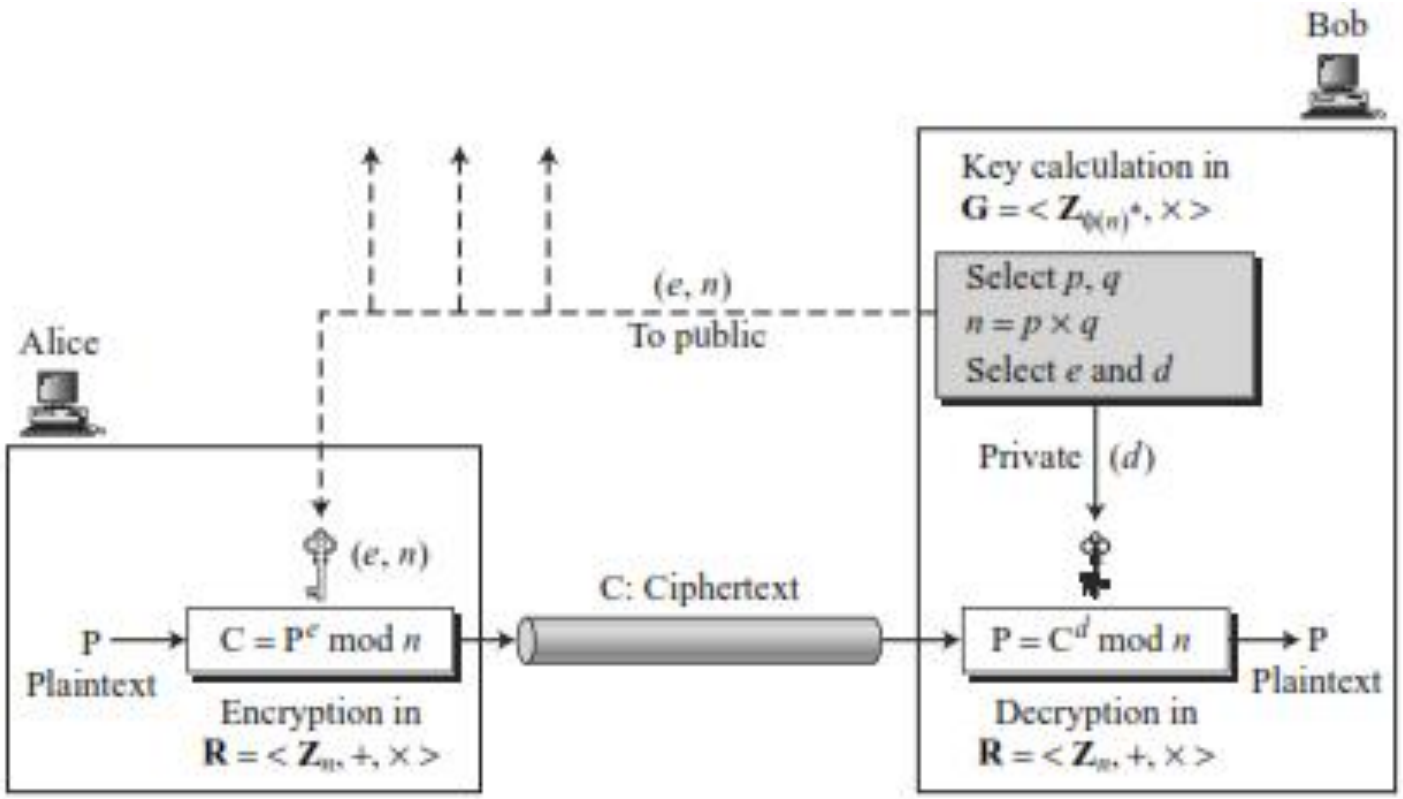
Square-and-Multiply for Modular Exponentiation
Input:
base element x
exponent $H = \sum_{i=0}^{t} h_i 2^i$ with $h_i \in 0, 1$ and $h_t = 1$
and modulus n
Output: $x^H \bmod n$
Initialization: $r = x$
Algorithm:

```
1    FOR i = t − 1 DOWNTO 0
1.1      r = r^2 mod n
         IF h_i = 1
1.2          r = r · x mod n
2    RETURN (r)
```

Complexity


Alice
Eve
Bob
P
?
P
$C = P^e \bmod n$
Polynomial complexity
$P = \sqrt[e]{C} \bmod n$
Exponential complexity
Polynomial complexity
$P = C^d \bmod n$
C
C
C
Insecure channel

RSA Encryption, Decryption, Key Generation



Example

Now assume that another person, John, wants to send a message to Bob. John can use the same public key announced by Bob (probably on his website), 13; John's plaintext is 63. John calculates the following:

Plaintext: 63	$C = 63^{13} = 28 \bmod 77$	Ciphertext: 28

Bob receives the ciphertext 28 and uses his private key 37 to decipher the ciphertext:

Ciphertext: 28	$P = 28^{37} = 63 \bmod 77$	Plaintext: 63

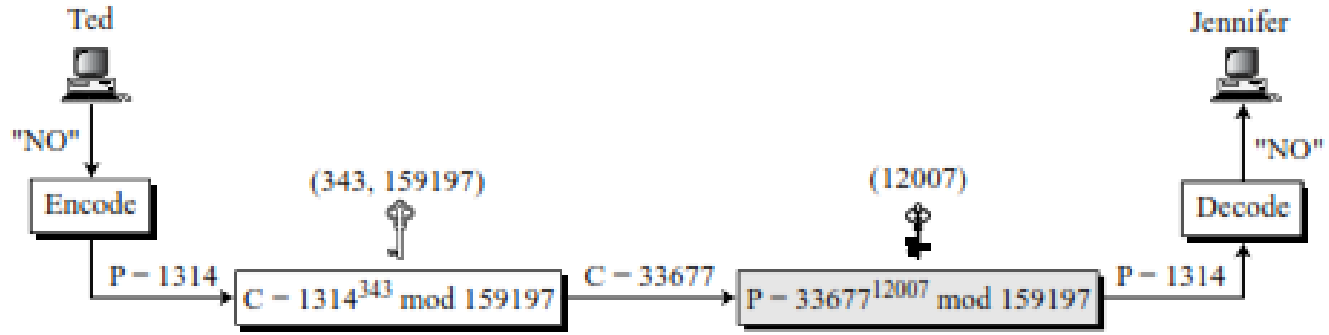
Example

Jennifer creates a pair of keys for herself. She chooses $p = 397$ and $q = 401$. She calculates $n = 397 \times 401 = 159197$. She then calculates $\phi(n) = 396 \times 400 = 158400$. She then chooses $e = 343$ and $d = 12007$. Show how Ted can send a message to Jennifer if he knows e and n.

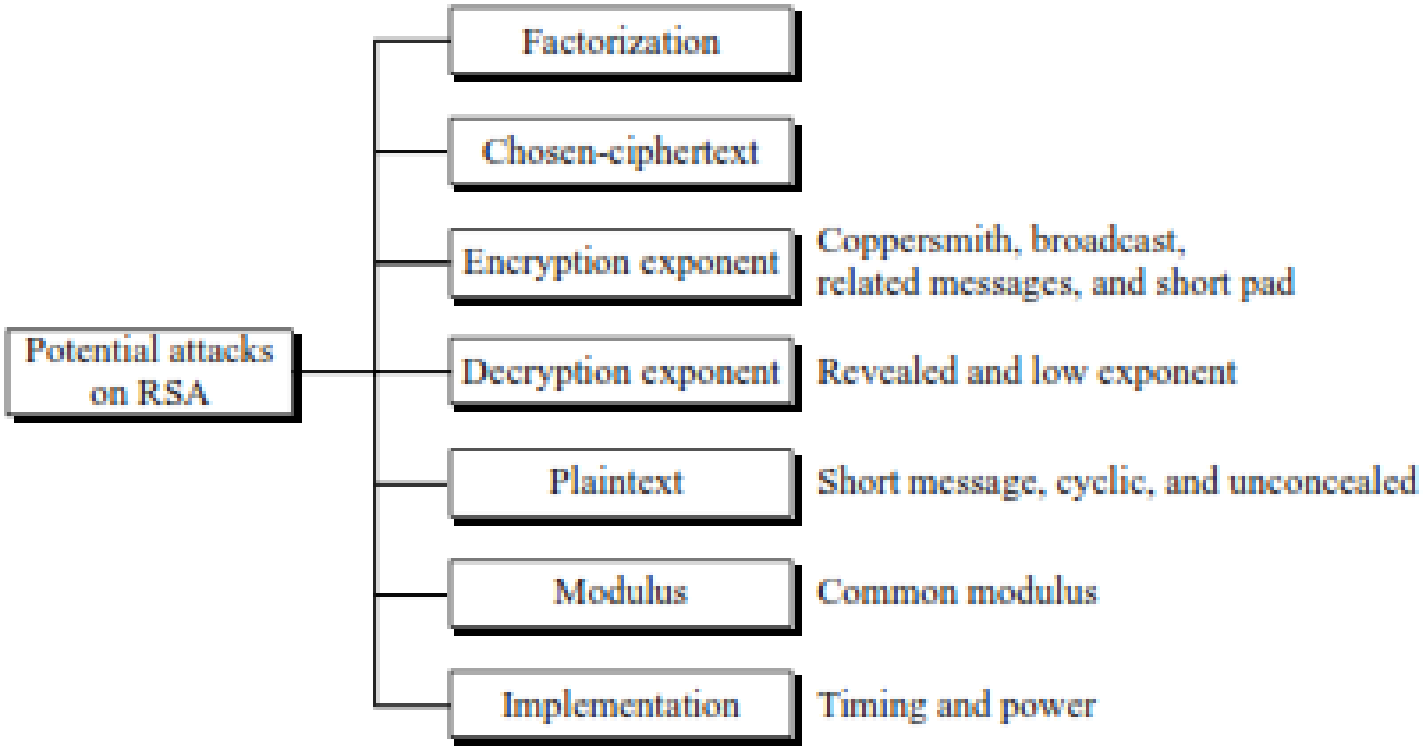
Solution

Suppose Ted wants to send the message "NO" to Jennifer. He changes each character to a number (from 00 to 25), with each character coded as two digits. He then concatenates the two coded characters and gets a four-digit number. The plaintext is 1314. Ted then uses e and n to encrypt the message. The ciphertext is $1314^{343} = 33677 \bmod 159197$. Jennifer receives the message 33677 and uses the decryption key d to decipher it as $33677^{12007} = 1314 \bmod 159197$. Jennifer then decodes 1314 as the message "NO". Figure 10.7 shows the process.

Figure 10.7 *Encryption and decryption in Example 10.7*



RSA Attacks


Potential attacks on RSA
Factorization
Chosen-ciphertext
Encryption exponent
Coppersmith, broadcast, related messages, and short pad
Decryption exponent
Revealed and low exponent
Plaintext
Short message, cyclic, and unconcealed
Modulus
Common modulus
Implementation
Timing and power

Factorization Attack

Factorization Attack

The security of RSA is based on the idea that the modulus is so large that it is infeasible to factor it in a reasonable time. Bob selects p and q and calculates $n = p \times q$. Although n is public, p and q are secret. If Eve can factor n and obtain p and q, she can calculate $\phi(n) = (p - 1)(q - 1)$. Eve then can calculate $d = e^{-1} \bmod \phi(n)$ because e is public. The private exponent d is the trapdoor that Eve can use to decrypt any encrypted message.

As we learned in Chapter 9, there are many factorization algorithms, but none of them can factor a large integer with polynomial time complexity. To be secure, RSA presently requires that n should be more than 300 decimal digits, which means that the modulus must be at least 1024 bits. Even using the largest and fastest computer available today, factoring an integer of this size would take an infeasibly long period of time. This means that RSA is secure as long as an efficient algorithm for factorization has not been found.

Chosen Ciphertext attack

Chosen-Ciphertext Attack

A potential attack on RSA is based on the multiplicative property of RSA. Assume that Alice creates the ciphertext $C = P^e \bmod n$ and sends C to Bob. Also assume that Bob will decrypt an arbitrary ciphertext for Eve, other than C. Eve intercepts C and uses the following steps to find P:

a. Eve chooses a random integer X in Z_n^*.
b. Eve calculates $Y = C \times X^e \bmod n$.
c. Eve sends Y to Bob for decryption and get $Z = Y^d \bmod n$; This step is an instance of a chosen-ciphertext attack.
d. Eve can easily find P because

$$Z = Y^d \bmod n = (C \times X^e)^d \bmod n = (C^d \times X^{ed}) \bmod n = (C^d \times X) \bmod n = (P \times X) \bmod n$$
$$Z = (P \times X) \bmod n \quad \rightarrow \quad P = Z \times X^{-1} \bmod n$$

Attacks on e (Cppersmith Theorem attack)

Attacks on the Encryption Exponent

To reduce the encryption time, it is tempting to use a small encryption exponent e. The common value for e is $e = 3$ (the second prime). However, there are some potential attacks on low encryption exponent that we briefly discuss here. These attacks do not generally result in a breakdown of the system, but they still need to be prevented. To thwart these kinds of attacks, the recommendation is to use $e = 2^{16} + 1 = 65537$ (or a prime close to this value).

Attacks on d

Revealed Decryption Exponent Attack It is obvious that if Eve can find the decryption exponent, d, she can decrypt the current encrypted message. However, the attack does not stop here. If Eve knows the value of d, she can use a probabilistic algorithm (not discussed here) to factor n and find the value of p and q. Consequently, if Bob changes only the compromised decryption exponent but keeps the same modulus, n, Eve will be able to decrypt future messages because she has the factorization of n. This means that if Bob finds out that the decryption exponent is compromised, he needs to choose new value for p and q, calculate n, and create totally new private and public keys.

In RSA, if d is comprised, then p, q, n, e, and d must be regenerated.

Small d Values

Low Decryption Exponent Attack Bob may think that using a small private-key d, would make the decryption process faster for him. Wiener showed that if $d < 1/3\ n^{1/4}$, a special type of attack based on *continuous fraction*, a topic discussed in number theory, can jeopardize the security of RSA. For this to happen, it must be the case that $q < p < 2q$. If these two conditions exist, Eve can factor n in polynomial time.

In RSA, the recommendation is to have $d \geq 1/3\ n^{1/4}$ to prevent low decryption exponent attack.

Short Message attack

Short Message Attack In the **short message attack,** if Eve knows the set of possible plaintexts, she then knows one more piece of information in addition to the fact that the ciphertext is the permutation of plaintext. Eve can encrypt all of the possible messages until the result is the same as the ciphertext intercepted. For example, if it is known that Alice is sending a four-digit number to Bob, Eve can easily try plaintext numbers from 0000 to 9999 to find the plaintext. For this reason, short messages must be padded with random bits at the front and the end to thwart this type of attack. It is strongly recommended that messages be padded with random bits before encryption using a method called OAEP, which is discussed later in this chapter.

Timing Attack

Power Attack The **Power attack** is similar to the timing attack. Kocher showed that if Eve can precisely measure the power consumed during decryption, she can launch a power attack based on the principle discussed for timing attack. An iteration involving multiplication and squaring consumes more power than an iteration that uses only squaring. The same kind of techniques used to prevent timing attacks can be used to thwart power attacks.

Fast Decryption

Example 7.6. Let the RSA parameters be given by:

$$\begin{aligned} p &= 11 & e &= 7 \\ q &= 13 & d &\equiv e^{-1} \equiv 103 \bmod 120 \\ n &= p \cdot q = 143 \end{aligned}$$

We now compute an RSA decryption for the ciphertext $y = 15$ using the CRT, i.e., the value $y^d = 15^{103} \bmod 143$. In the first step, we compute the modular representation of y:

$$\begin{aligned} y_p &\equiv 15 \equiv 4 \bmod 11 \\ y_p &\equiv 15 \equiv 2 \bmod 13 \end{aligned}$$

In the second step, we perform the exponentiation in the transform domain with the short exponents. These are:

$$\begin{aligned} d_p &\equiv 103 \equiv 3 \bmod 10 \\ d_q &\equiv 103 \equiv 7 \bmod 12 \end{aligned}$$

Here are the exponentiations:

$$\begin{aligned} x_p &\equiv y_p^{d_p} = 4^3 = 64 \equiv 9 \bmod 11 \\ x_q &\equiv y_q^{d_q} = 2^7 = 128 \equiv 11 \bmod 13 \end{aligned}$$

Fast Decryption

$$c_p = 13^{-1} \equiv 2^{-1} \equiv 6 \bmod 11 \qquad c_q = 11^{-1} \equiv 6 \bmod 13$$

The plaintext x follows now as:

$$x \equiv [qc_p]x_p + [pc_q]x_q \bmod n$$
$$x \equiv [13 \cdot 6]9 + [11 \cdot 6]11 \bmod 143$$
$$x \equiv 702 + 726 = 1428 \equiv 141 \bmod 143$$

RSA Factoring Attacks

Decimal digits	Bit length	Date
100	330	April 1991
110	364	April 1992
120	397	June 1993
129	426	April 1994
140	463	February 1999
155	512	August 1999
200	664	May 2005

RSA algorithm summary

Key Generation by Alice	
Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calcuate $\phi(n) = (p - 1)(q - 1)$	
Select integer e	$\gcd(\phi(n), e) = 1; 1 < e < \phi(n)$
Calculate d	$d \equiv e^{-1} \pmod{\phi(n)}$
Public key	$PU = \{e, n\}$
Private key	$PR = \{d, n\}$

Encryption by Bob with Alice's Public Key	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \bmod n$

Decryption by Alice with Alice's Public Key	
Ciphertext:	C
Plaintext:	$M = C^d \bmod n$

Openssl

==> openssl genrsa

```
Generating RSA private key, 2048 bit long modulus (2 primes)
.....+++++
..........+++++
e is 65537 (0x010001)
-----BEGIN RSA PRIVATE KEY-----
MIIEpgIBAAKCAQEA7Nk5LpuPA1ZiWqCHfwVilholwLTkb2hTwX4iOarC8fEfWMLk
heqfvi7Aik7PZOKtLVl4q8jebjqxBm2hnow+TcBfPCGiJGlzFn9ABGO122qtJ20e
YsjhB7kronG/DG82b/pc7ostT3ZBpjEykt8ytTPBxAt5+pFclXzj65eP4ERki25w
RMqCzFKWzc/q++fWfun053m+T26HeP4zCHMkPV9vTIJuI+dtuJ0Xal9k8kf61KBU
7wFDuDbo3rfs7dk3Nvlb68/IxSo2aLl8vMhJ5SlbM6F4ifUcZqfh9cDpqfw0Lljl
xaI9nJkbQzJ5jnqDncSAVtERHNSu4zOxsXxLkQIDAQABAoIBAQCMkLL2LU5oLal7
ndAsm5a3+JalUuZMFD+5E1HepbDDBFaVSD12GPQrW/XbX8CIYFtlEbejRDh2dRDA
/umvfg4v+N7Mgi2HrKmoDWeB82dnzFztjD4/ZHbhLYlvAFDhYVOOi6kBzcnbYhS+
PG6Gaj7e+dOitSj0g7bN5WwjUzUJZIlG/VKKWnRGZtstXX5jm0wQew6PW561tv8Y
s/J2AAab/2THcHQQ5tqsaCwyCLimSBvE7DoYxcf9LKIpTEuZOpA5Bz66PJdgnYzY
EXutIw8DU1iA6Uw5KZHaAl0ByAitpCezfThLjJkSkVP5fKRrsek9sDRIygtVa3Qa
nEEVwMjdAoGBAP+sKcVtLOqtRMcEzPqksdlgNX1Dxy99Hb+3VmVrJuFukssEDGKM
BPaaE5cCzSDRZJ4p7Bt+Xpp4gAMDnA2a4VWEMTCbVdvEgOTxSGBPvLpYqcpt3cQA
gLC2TLWlVgWDi08BJdLiwBBnBSajT72RveUeMyFyh3QLgWrvRvIHHiDXAoGBAO0m
4z8NymYt6+KYN0yBWYfBjuXlOZVjVAURcKcrnQrQ6zB/KT7IzR+NkSA/JwmCv454
pIbPLyAYtvqdi8wEIwClROvB0dra6kVuUTlKLSjy6wvKqlNEibFptJf42JtvsqHZ
y3NzH/Zw25ha7jGilwfgPR3vjl+cMlA6Ph19RSHXAoGBAPxrPiZDnmfnPxL6I2GF
vnDgbo7Intu8u+Uunaatfopsf5Ld4VheAvxwq8yYoGq5MIySuR9/yOjbHI0lQBmS
gsvKIkJp6f96ZwMhUCJ+NscHiEJp69t535QXt25S2LXC5IPQj0ZARf0rqMM30x9G
x2NwSGzKRP8F6PTpXXLQIierAoGBANtoBAy9HImmaLtuqpK8hXGFEUj+52SfzgcW
WIxBXHy6Ry3KKAWvT5+moSMdamdxMPqGAWm721StqPR6t/DbzuqDyqz318jMirwL
0VfYmalt/SoeqplSJrYeHvgPQY+lKrZlQragR20AgxoU3pTLYUHnI4RDs/j4ENCK
4hb0Y/ZnAoGBAIoAitJHM6tCypiQB+wMvBjmTeV90smglGoq20a7MiwKMHlidxxI
YcY5AVMCeahOxpIA7cNMWTMvgZru+rQblLXCWD/tZ3Gd9MQzf0sunxWKJhUV/Mk0
mPX/HkBJaXu2kN5gkFXAHCs9E8SJLWy6FG4tKox+Uw9wo0GlBr+WVAHQ
-----END RSA PRIVATE KEY-----
```

Private Key

```
==> openssl genrsa -out key.pem 2048

Generating RSA private key, 2048 bit long modulus (2 primes)
.....................................................................+++++
...+++++
e is 65537 (0x010001)
```

Internals

```
==> openssl rsa -in key.pem -inform pem -noout -text

RSA Private-Key: (2048 bit, 2 primes)
modulus:
    00:ab:af:a4:ed:1f:82:f6:99:f0:93:cb:57:bb:cd:
    85:08:6d:da:30:4c:3c:a6:25:94:4b:ea:2e:71:45:
    0c:ca:62:8e:a4:d1:fe:cc:4a:e2:c8:18:fd:ad:23:
    5f:e6:a3:f0:b4:57:a9:6f:e9:38:ca:e4:42:af:ed:
    73:b2:15:3a:04:1a:40:f6:e7:40:4c:69:53:25:95:
    ca:9d:44:6e:05:56:cf:7b:b9:d6:57:9f:58:61:01:
    00:c1:fe:fe:7e:10:96:8f:2c:d8:16:33:39:9e:37:
    4f:c3:9b:77:01:b9:70:ea:10:c7:71:81:29:19:25:
    50:25:71:12:f5:2c:31:8d:61:26:39:09:6c:98:cd:
    78:00:aa:6a:cf:0c:5c:27:07:89:0c:d1:06:5a:91:
    92:6f:6f:2e:24:46:f0:99:a3:77:29:ec:d2:38:4d:
    91:aa:fb:b6:7f:5e:3d:b4:eb:3e:7b:96:0e:3a:a9:
    8f:48:5b:0b:e4:5b:61:ba:dd:65:a3:e9:33:14:9e:
    86:94:01:e3:69:83:f0:78:9c:3e:28:32:b8:58:19:
    ba:2c:f2:9b:2d:2a:7b:e7:ec:58:1e:b4:fc:5a:39:
    24:ae:0a:8e:26:15:ce:19:f0:b0:d7:b0:c3:90:03:
    38:66:4a:5b:ba:a8:f1:d6:d3:94:98:fd:bc:4d:67:
    c2:61
publicExponent: 65537 (0x10001)
privateExponent:
    55:f8:59:1c:c8:07:bb:56:70:6a:81:8b:48:26:6c:
    b4:40:d5:de:13:7e:d7:2f:c0:27:97:77:74:0e:c0:
    8d:e3:76:4c:40:3f:57:ab:34:0e:40:bd:5e:62:75:
    56:37:c7:83:76:d6:08:8c:ff:7c:51:7a:b7:3f:af:
    0c:80:a6:91:81:58:00:8a:e1:de:a1:6b:1a:49:fc:
    b0:6d:a0:ae:19:bf:41:d4:57:e9:7e:88:31:e2:df:
    af:44:f1:c8:cc:a3:a7:c4:2b:dc:4a:00:53:22:9d:
    55:74:d6:cd:cd:3f:26:66:0a:88:e2:c5:62:ab:15:
    8b:fa:28:25:0e:e1:2d:4a:a3:71:67:ae:d5:4c:b9:
    af:ee:da:65:fd:0c:44:18:f4:b1:e6:bc:b0:c5:17:
    a9:20:a9:52:35:a0:63:c4:77:3d:7b:f7:4d:9d:c9:
    1c:dc:d0:83:7a:44:aa:f9:d5:66:79:c4:e0:b4:57:
    3a:ba:9f:9a:eb:f4:04:5e:c5:d5:c7:3c:d7:11:a2:
    74:44:22:1f:b0:69:4f:8e:43:f2:e8:8d:e1:ec:c8:
    96:3a:8d:11:60:be:48:58:17:51:ca:b5:60:ae:f3:
    4d:f5:a3:1d:85:7a:f6:4b:f2:f5:ce:0a:76:a8:a4:
    99:f0:7e:06:cc:52:e5:f0:31:9f:45:98:7a:81:00:
    85
prime1:
    00:d3:e3:2f:f5:7b:13:51:1b:bc:c1:b2:a3:77:b0:
    0a:25:41:cf:64:89:15:22:b6:cf:9f:84:27:10:df:
    f3:c7:d1:be:9c:8e:42:df:fe:f1:b0:03:2e:d3:d1:
    99:b6:52:67:d7:ee:98:01:43:0d:99:0c:e2:76:a1:
    dc:f6:88:57:a6:28:c2:89:67:ea:d1:5a:28:75:70:
    96:8e:3a:0e:0a:33:b2:8e:53:c5:06:58:c1:18:20:
    81:99:54:d0:45:7a:21:8f:b8:10:c9:ce:72:82:91:
    56:45:ed:2c:e8:cc:0d:b3:6e:30:78:5d:35:fd:79:
    bd:68:99:2f:b0:08:93:df:cf
prime2:
    00:cf:6d:e0:04:21:c7:7b:d1:1a:2b:68:d8:bd:23:
    c0:cf:66:19:65:89:8c:68:be:da:87:27:3e:40:e5:
    b3:20:03:26:59:19:4f:eb:b6:86:28:36:b0:74:32:
    e2:12:13:c4:12:38:ee:71:98:0b:91:64:af:be:3b:
    87:a2:f0:9f:d8:36:6c:7e:ce:a9:a9:29:de:42:f7:
    8c:d7:2e:c4:ef:36:88:e5:e9:ec:e4:ff:dd:22:26:
    5d:a6:79:6f:75:6c:5d:3a:52:da:54:3a:3d:5b:96:
    e7:1f:0a:0d:c7:6d:f6:3a:2d:91:3b:bf:08:e5:92:
    39:a5:4c:70:46:aa:46:16:cf
exponent1:
    0f:b0:b2:1b:76:7a:ae:b5:e4:1b:5f:d4:15:07:d7:
    28:7d:20:13:6c:c7:40:e3:d2:aa:18:4a:20:48:c5:
    2f:95:cb:8c:a2:48:37:78:14:83:99:28:bd:8c:b6:
    da:36:6d:f4:22:79:e5:16:07:0a:bf:56:81:bc:68:
    b5:64:d1:40:bf:a1:f0:34:de:c1:93:f0:8d:09:c2:
    4c:53:e6:38:41:2d:c6:b6:53:4f:ae:00:d6:7d:89:
    bb:45:f9:8a:3b:8a:02:af:79:a6:c7:ff:d8:c5:54:
    63:27:35:fd:23:27:1c:93:5b:49:7e:75:82:08:a2:
    ca:fd:14:f7:ef:1a:ac:27
exponent2:
    00:93:39:f5:6a:79:5f:51:6e:95:18:82:8e:73:90:
    d0:e5:64:1e:5a:87:4a:75:7e:21:35:14:91:87:16:
    82:11:12:ab:41:4a:4a:03:8f:c5:a0:fd:50:38:e9:
    74:b4:47:fb:3e:c3:d1:da:26:84:ef:69:7b:a3:96:
    35:2b:5d:86:d6:bb:aa:3e:47:08:fc:dc:8e:b9:11:
    63:91:c7:cc:57:cd:69:55:66:b7:91:c2:59:7e:47:
    a4:e8:e8:00:48:63:e5:b7:e3:de:bb:31:ab:23:3a:
    f4:48:7f:a6:50:0a:a8:5d:9a:c2:1e:99:f5:02:9c:
    ca:f5:9c:4f:84:98:8e:ae:d9
coefficient:
    39:04:e4:6f:11:0e:b9:51:62:b1:d4:48:2c:93:56:
    67:fb:c6:78:24:dd:e4:e7:8c:32:15:7c:30:53:a5:
    82:7c:66:00:81:b4:1e:e6:a0:b1:33:45:a8:63:63:
    1c:b6:e9:cf:55:ec:b2:4e:7c:31:a0:3f:4e:cf:3c:
    57:df:9e:3e:31:a4:c3:e1:41:a4:00:f1:87:34:c1:
    17:84:71:56:c7:13:d1:e0:e2:c2:c9:da:ae:61:62:
    36:46:0c:c3:37:62:2b:58:7c:79:8b:b6:ad:d7:54:
    4e:8c:78:b5:fb:d8:71:38:83:65:5a:e4:d7:60:da:
    cf:a3:5a:fb:71:1c:46:aa
```

Public Key

```
==> openssl rsa -in key.pem -inform pem -pubout > key.pub

writing RSA key

==> cat key.pub

-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAq6+k7R+C9pnwk8tXu82F
CG3aMEw8piWUS+oucUUMymKOpNH+zEriyBj9rSNf5qPwtFepb+k4yuRCr+1zshU6
BBpA9udATGlTJZXKnURuBVbPe7nWV59YYQEAwf7+fhCWjyzYFjM5njdPw5t3Ablw
6hDHcYEpGSVQJXES9SwxjWEmOQ1smMl4AKpqzwxcJweJDNEGWpGSb28uJEbwmaN3
KezSOE2Rqvu2f149tOs+e5YOOqmPSFsL5Fthutllo+kzFJ6GlAHjaYPweJw+KDK4
WBm6LPKbLSp75+xYHrT8WjkkrgqOJhXOGfCwl7DDkAM4zkpbuqjx1tOUmP28TWfC
YQIDAQAB
-----END PUBLIC KEY-----
```

Digital Signature

Presentation by:
V. Balasubramanian
SSN College of Engineering

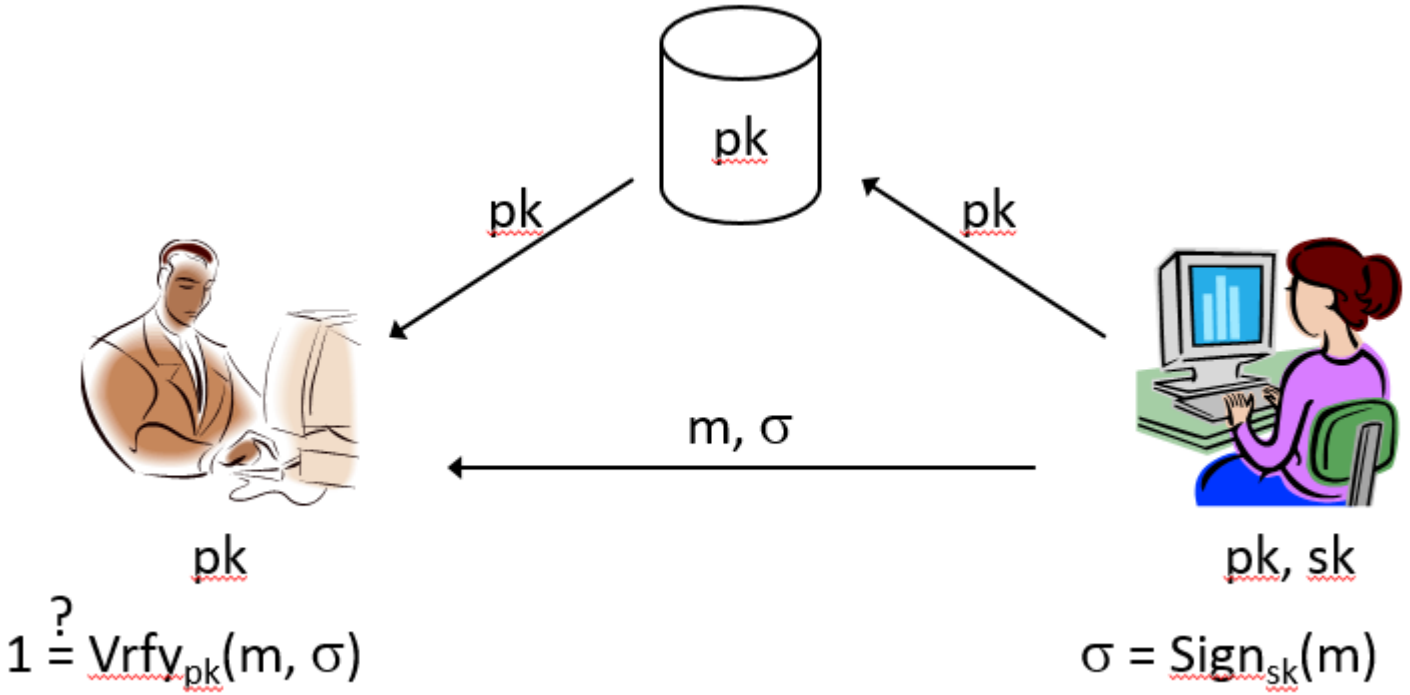
Objectives

- **To define a digital signature**
- **To define security services provided by a digital signature**
- **To define attacks on digital signatures**
- **To discuss some digital signature schemes, including RSA, ElGamal,**
- **Schnorr, DSS, and elliptic curve**
- **To describe some applications of digital signatures**

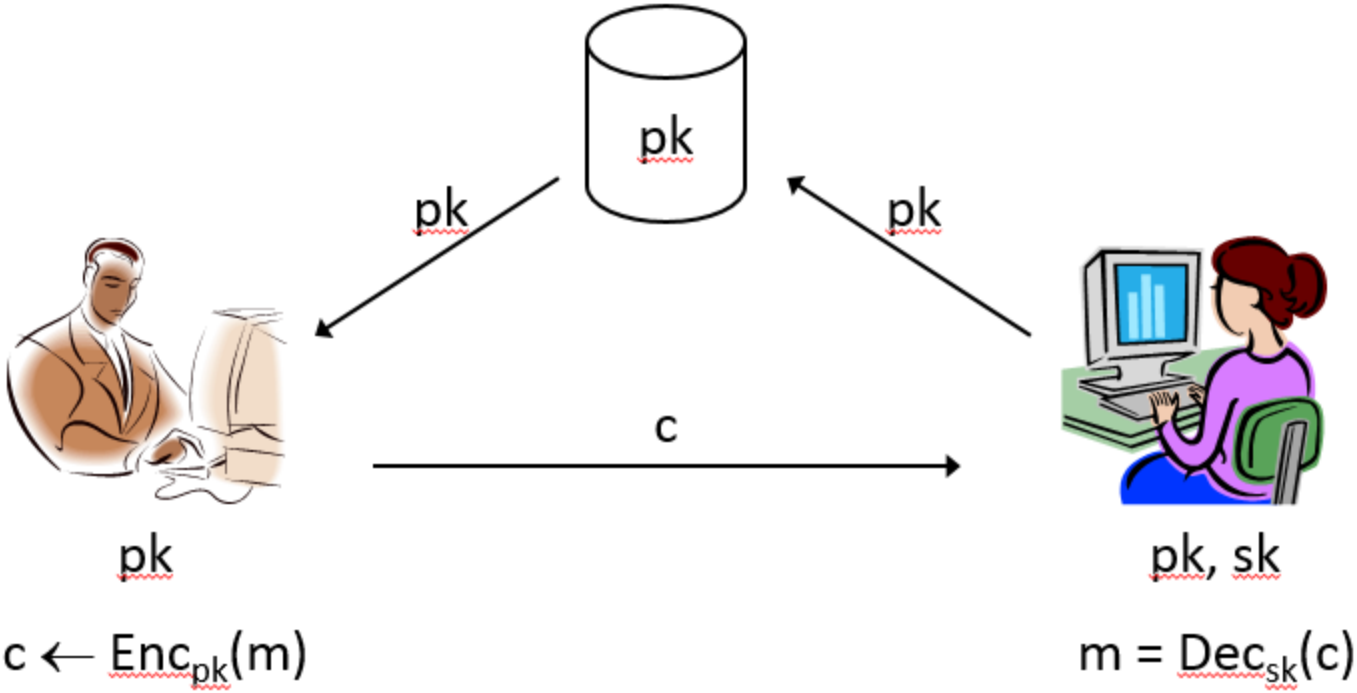
Digital signature

- Provide *integrity* in the public-key setting
- Analogous to message authentication codes, but some key differences…

Digital Signature



Public key encryption



Digital Signature

- When Alice sends a message to Bob, Bob needs to check the authenticity of the sender; he needs to be sure that the message comes from Alice and not Eve. Bob can ask Alice to sign the message electronically.
- In other words, an electronic signature can prove the authenticity of Alice as the sender of the message. We refer to this type of signature as a digital signature.

Differences between Conventional and Digital signature

- **Inclusion**
- **Verification Method**
- **Relationship**
- **Duplicity**

Inclusion

A conventional signature is included in the document; it is part of the document. But when we sign a document digitally, we send the signature as a separate document.

Verification Method

For a conventional signature, when the recipient receives a document, she compares the signature on the document with the signature on file. For a digital signature, the recipient receives the message and the signature. The recipient needs to apply a verification technique to the combination of the message and the signature to verify the authenticity.

Relationship

For a conventional signature, there is normally a one-to-many relationship between a signature and documents. For a digital signature, there is a one-to-one relationship between a signature and a message.

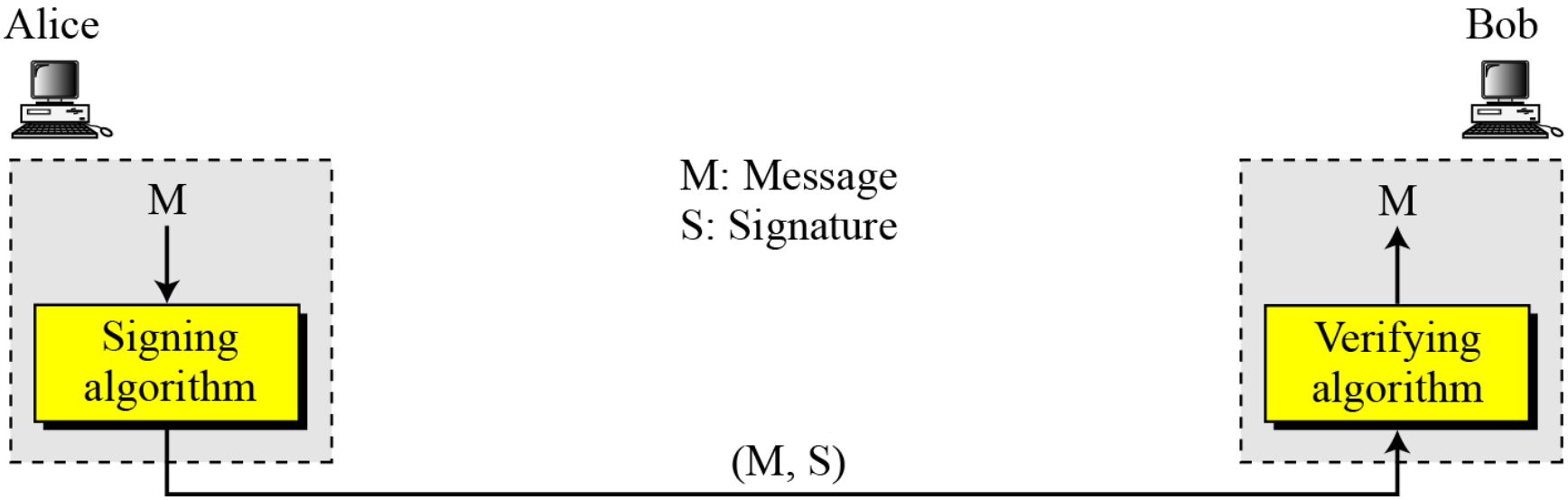
Duplicity

In conventional signature, a copy of the signed document can be distinguished from the original one on file. In digital signature, there is no such distinction unless there is a factor of time on the document.

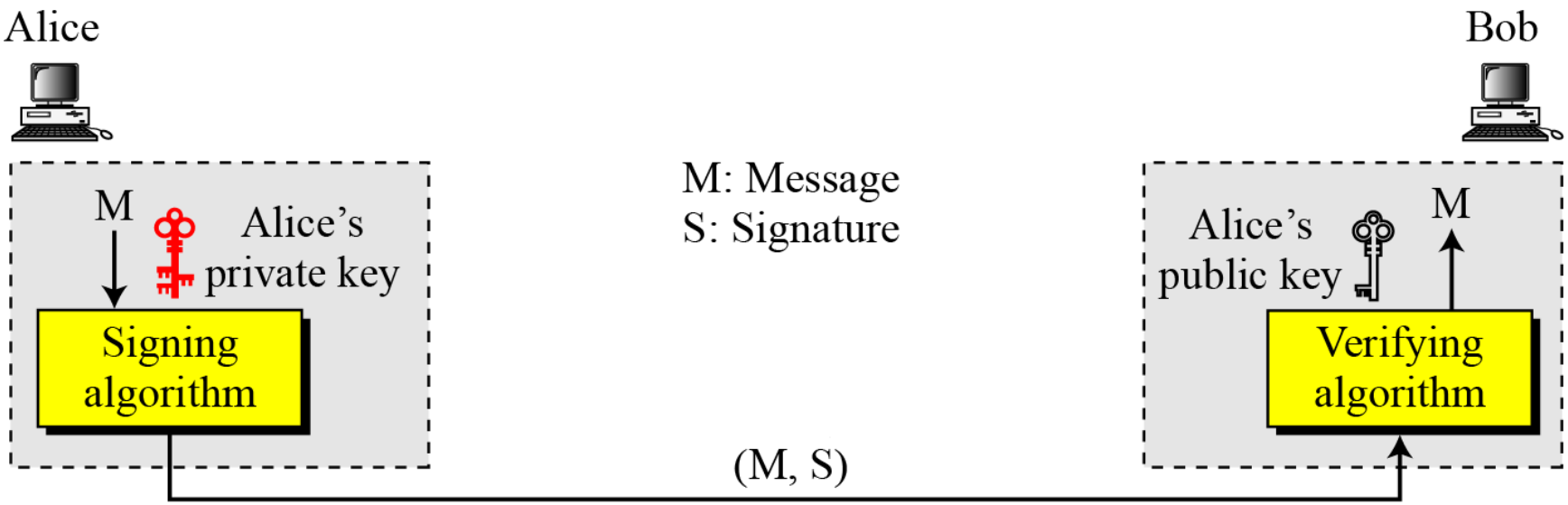
Process

- ***The sender uses a signing algorithm to sign the message.***
- ***The message and the signature are sent to the receiver.***
- ***The receiver receives the message and the signature and applies the verifying algorithm to the combination.***
- ***If the result is true, the message is accepted; otherwise, it is rejected.***

Digital Signature Process



Adding keys

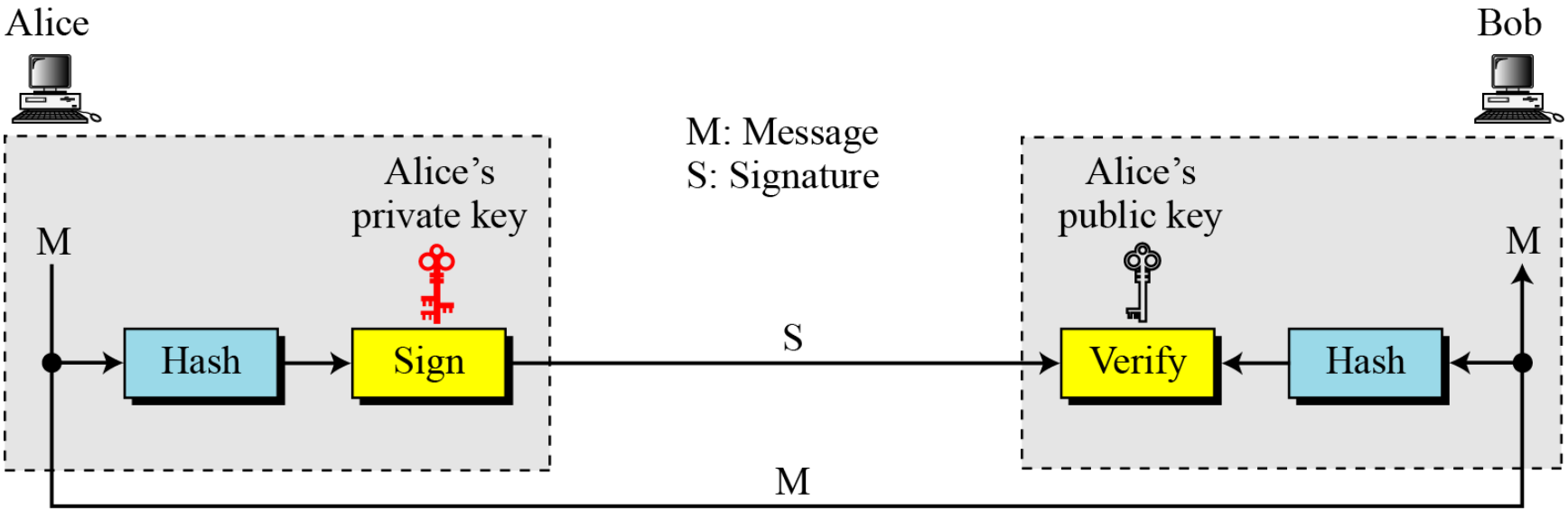


A digital signature needs a public-key system. The signer signs with her private key; the verifier verifies with the signer's public key.

Digital Signature

A cryptosystem uses the private and public keys of the receiver: a digital signature uses the private and public keys of the sender.

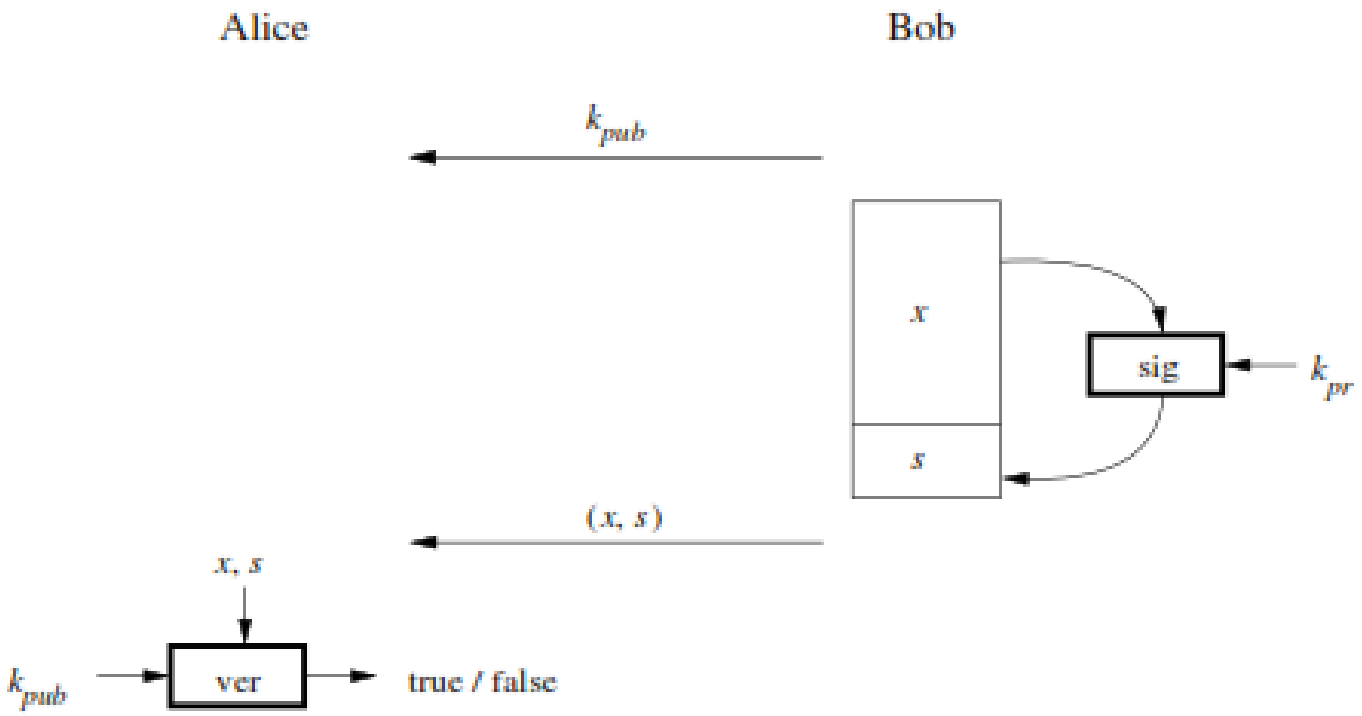
Signing the digest



DSS

- Can we use a secret (symmetric) key to both sign and verify a signature?
- The answer is negative for several reasons. First, a secret key is known by only two entities (Alice and Bob, for example). So if Alice needs to sign another document and send it to Ted, she needs to use another secret key. Second, as we will see, creating a secret key for a session involves authentication, which uses a digital signature.

Digital Signature



Protocol

Basic Digital Signature Protocol

Alice		Bob
		generate $k_{pr,B}$, $k_{pub,B}$
	$\xleftarrow{k_{pub,B}}$	publish public key
		sign message: $s = \text{sig}_{k_{pr}}(x)$
	$\xleftarrow{(x,s)}$	send message + signature
verify signature: $\text{ver}_{k_{pr,B}}(x, s) = \text{true/false}$		

Digital Signature Services

- Message Authentication
- Message Integrity
- Nonrepudiation
- Confidentiality

Message Authentication

Message Authentication

A secure digital signature scheme, like a secure conventional signature (one that cannot be easily copied) can provide message authentication (also referred to as data-origin authentication). Bob can verify that the message is sent by Alice because Alice's public key is used in verification. Alice's public key cannot verify the signature signed by Eve's private key.

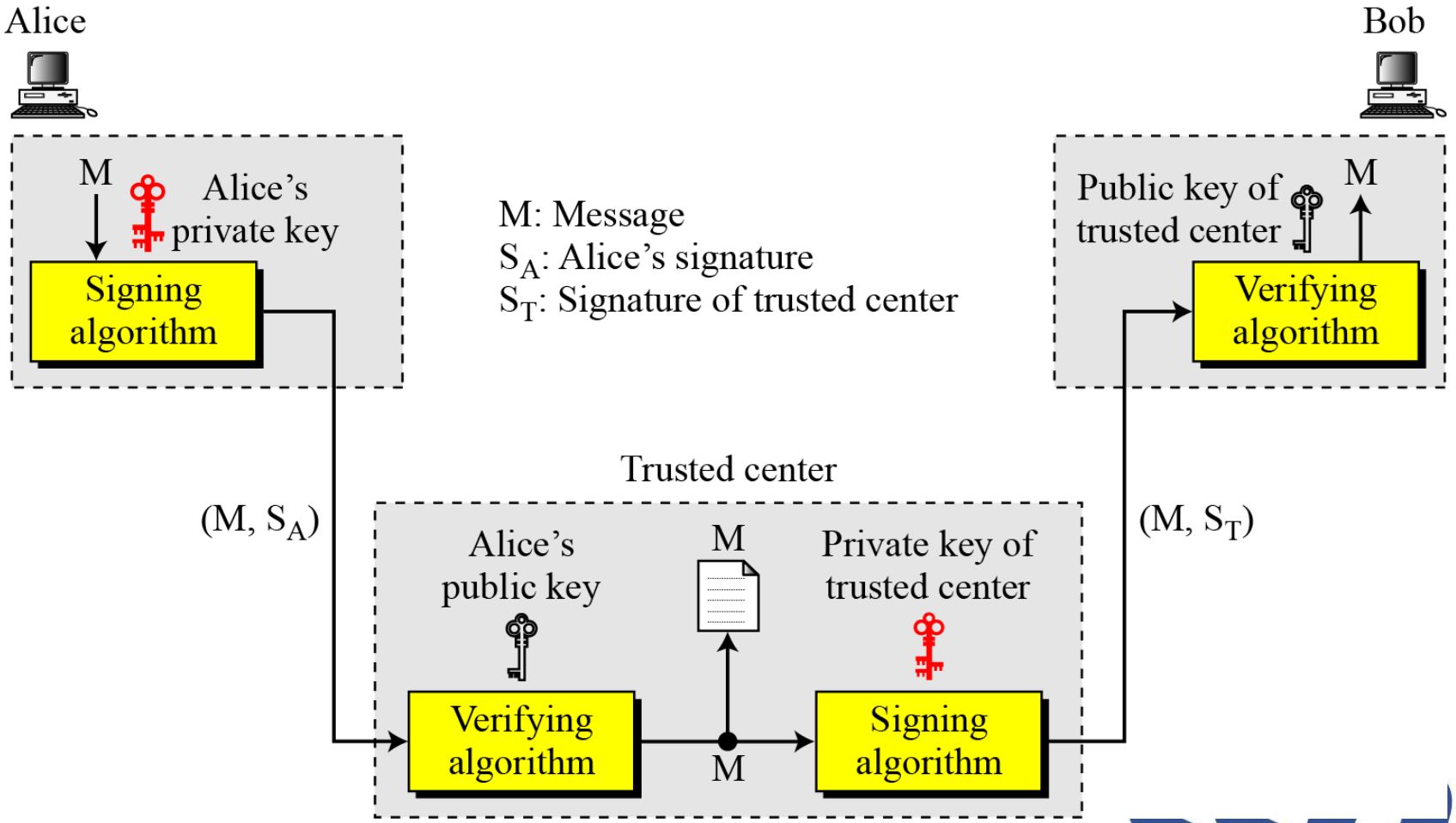
A digital signature provides message authentication.

Message Integrity

- *The integrity of the message is preserved even if we sign the whole message because we cannot get the same signature if the message is changed.*

A digital signature provides message integrity.

Non-Repudiation



Non Repudiation

- *Public verifiability*
  - "Anyone" can verify a signature
  - (Only a holder of the key can verify a MAC tag)

$\Rightarrow$ *Transferability*

- Can forward a signature to someone else…

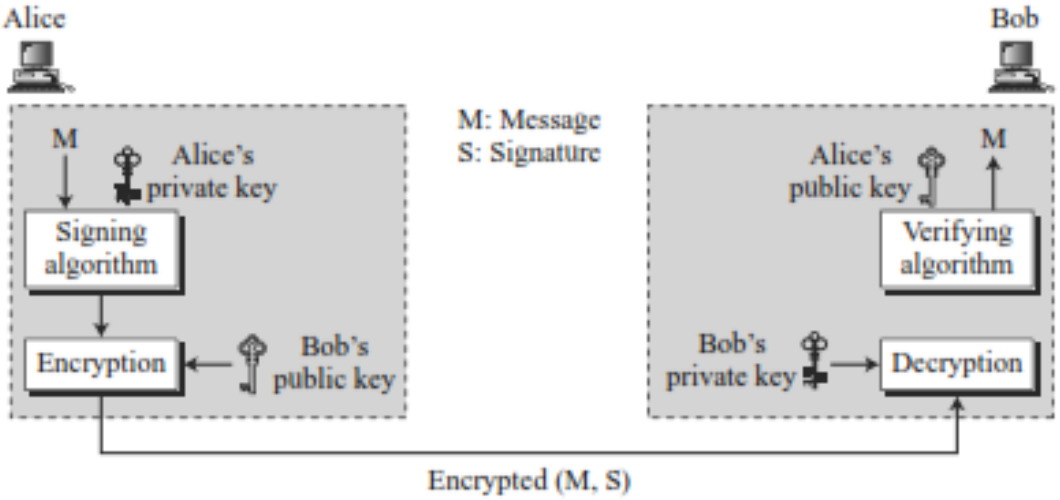
$\Rightarrow$ *Non-repudiation*

Confidentiality

Confidentiality

A digital signature does not provide confidential communication. If confidentiality is required, the message and the signature must be encrypted using either a secret-key or public-key cryptosystem. Figure 13.5 shows how this extra level can be added to a simple digital signature scheme.

Figure 13.5 *Adding confidentiality to a digital signature scheme*



Schemes

RSA Digital Signature Scheme

ElGamal Digital Signature Scheme

Schnorr Digital Signature Scheme

Digital Signature Standard (DSS)

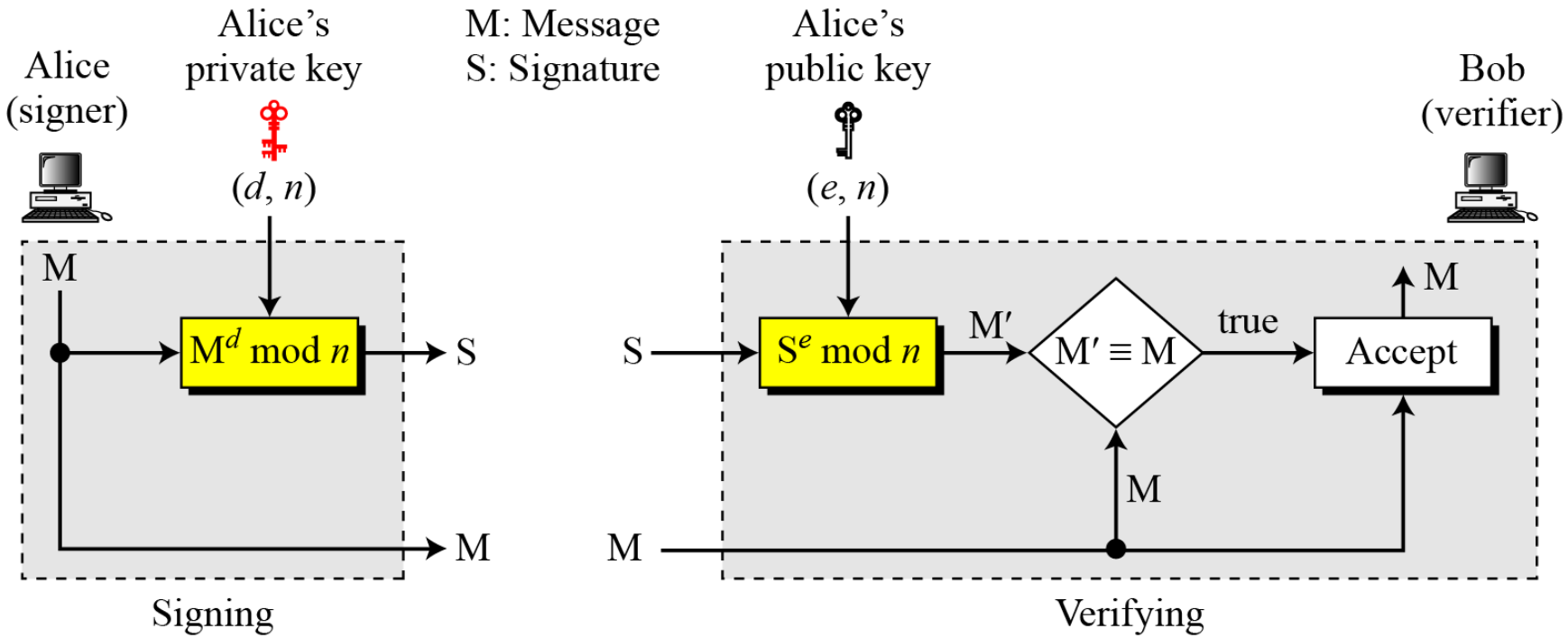
Elliptic Curve Digital Signature Scheme

RSA

- *Key Generation*
- *Key generation in the RSA digital signature scheme is exactly the same as key generation in the RSA*

In the RSA digital signature scheme, *d* is private; *e* and *n* are public.

RSA Digital Signature



Example

For the security of the signature, the value of p and q must be very large. As a trivial example, suppose that Alice chooses $p = 823$ and $q = 953$, and calculates $n = 784319$. The value of $\phi(n)$ is 782544. Now she chooses $e = 313$ and calculates $d = 160009$. At this point key generation is complete. Now imagine that Alice wants to send a message with the value of M = 19070 to Bob. She uses her private exponent, 160009, to sign the message:

$$\text{M: } 19070 \quad \rightarrow \quad S = (19070^{160009}) \bmod 784319 = 210625 \bmod 784319$$

Alice sends the message and the signature to Bob. Bob receives the message and the signature. He calculates

$$M' = 210625^{313} \bmod 784319 = 19070 \bmod 784319 \quad \rightarrow \quad M \equiv M' \bmod n$$

Bob accepts the message because he has verified Alice's signature.

Protocol

Basic RSA Digital Signature Protocol

Alice		Bob
		$k_{pr} = d, k_{pub} = (n, e)$
	$\xleftarrow{(n,e)}$	
		compute signature: $s = \text{sig}_{k_{pr}}(x) \equiv x^d \bmod n$
	$\xleftarrow{(x,s)}$	

verify: $\text{ver}_{k_{pub}}(x, s)$

$x' \equiv s^e \bmod n$

$$x' \begin{cases} \equiv x \bmod n & \Longrightarrow \text{valid signature} \\ \not\equiv x \bmod n & \Longrightarrow \text{invalid signature} \end{cases}$$

Illustration

Alice

Bob

1. choose $p = 3$ and $q = 11$
2. $n = p \cdot q = 33$
3. $\Phi(n) = (3-1)(11-1) = 20$
4. choose $e = 3$
5. $d \equiv e^{-1} \equiv 7 \bmod 20$

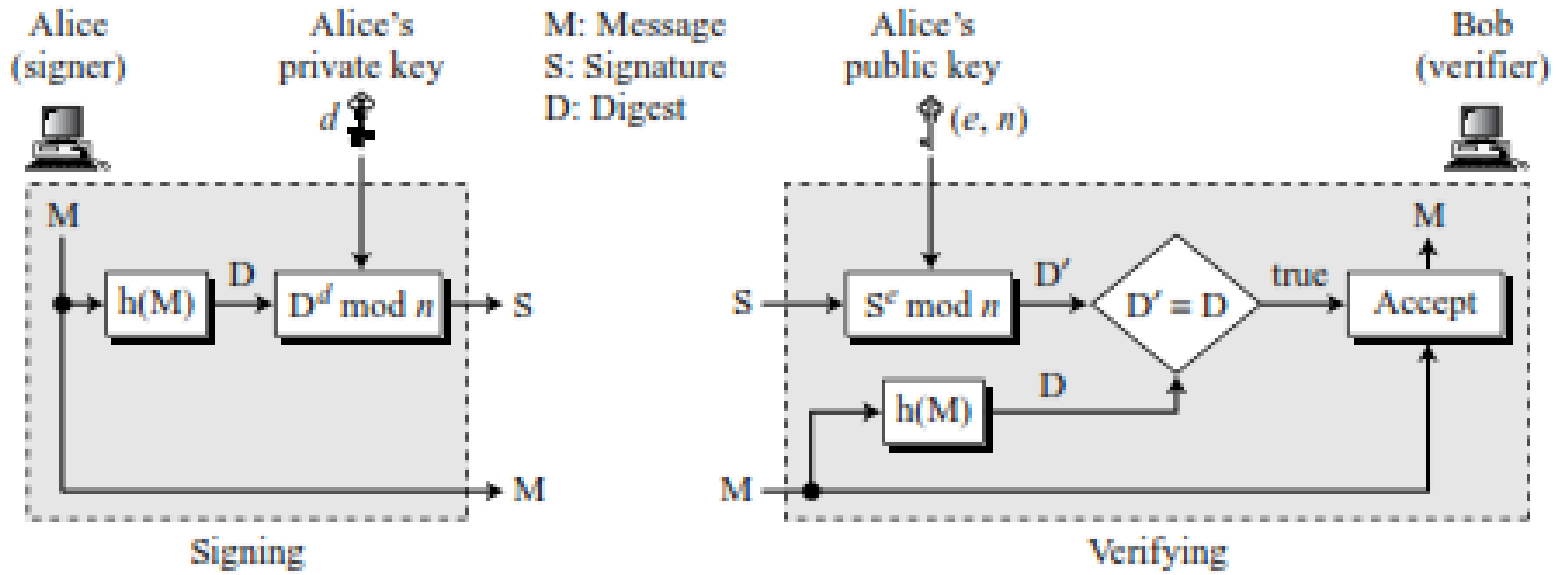
$(n,e)=(33,3)$ ←

compute signature for message $x = 4$:
$s = x^d \equiv 4^7 \equiv 16 \bmod 33$

$(x,s)=(4,16)$ ←

verify:
$x' = s^e \equiv 16^3 \equiv 4 \bmod 33$
$x' \equiv x \bmod 33 \Longrightarrow$ valid signature

RSA Signature on the Message Digest



ElGamal Digital Signature Scheme

Key Generation for Elgamal Digital Signature

1. Choose a large prime p.
2. Choose a primitive element α of $\mathbb{Z}_p^*$ or a subgroup of $\mathbb{Z}_p^*$.
3. Choose a random integer $d \in \{2, 3, \ldots, p-2\}$.
4. Compute $\beta = \alpha^d \bmod p$.

Signature

Elgamal Signature Generation

1. Choose a random ephemeral key $k_E \in \{0, 1, 2, \ldots, p-2\}$ such that $\gcd(k_E, p-1) = 1$.
2. Compute the signature parameters:

$$r \equiv \alpha^{k_E} \bmod p,$$
$$s \equiv (x - d \cdot r) k_E^{-1} \bmod p - 1.$$

Verification

Elgamal Signature Verification

1. Compute the value

$$t \equiv \beta^r \cdot r^s \bmod p$$

2. The verification follows from:

$$t \begin{cases} \equiv \alpha^x \bmod p & \Longrightarrow \text{valid signature} \\ \not\equiv \alpha^x \bmod p & \Longrightarrow \text{invalid signature} \end{cases}$$

Illustration

Alice

Bob

1. choose $p = 29$
2. choose $\alpha = 2$
3. choose $d = 12$
4. $\beta = \alpha^d \equiv 7 \bmod 29$

$(p,\alpha,\beta)=(29,2,7)$ ←

compute signature for message $x = 26$:
choose $k_E = 5$, note that $\gcd(5, 28) = 1$
$r = \alpha^{k_E} \equiv 2^5 \equiv 3 \bmod 29$
$s = (x - d\,r)\,k_E^{-1} \equiv (-10) \cdot 17 \equiv 26 \bmod 28$

$(x,(r,s))=(26,(3,26))$ ←

verify:
$t = \beta^r \cdot r^s \equiv 7^3 \cdot 3^{26} \equiv 22 \bmod 29$
$\alpha^x \equiv 2^{26} \equiv 22 \bmod 29$
$t \equiv \alpha^x \bmod 29 \Longrightarrow$ valid signature

Digital Signature Algorithm

- Federal US Government standard for digital signatures (DSS)
- Proposed by the National Institute of Standards and Technology (NIST)
- DSA is based on the Elgamal signature scheme
- Signature is only 320 bits long
- Signature verification is slower compared to RSA

Algrbraic Structure

- Cyclic Groups
- P=1024 bits
- Q=160 bits

p	q	Signature
1024	160	320
2048	224	448
3072	256	512

DSA

Key generation of DSA:

1. Generate a prime p with $2^{1023} < p < 2^{1024}$
2. Find a prime divisor q of $p-1$ with $2^{159} < q < 2^{160}$
3. Find an integer α with $\text{ord}(\alpha)=q$
4. Choose a random integer d with $0<d<q$
5. Compute $\beta \equiv \alpha^d \bmod p$

The keys are:

$k_{pub} = (p,q,\alpha,\beta)$

$k_{pr} = (d)$

Signature

DSA signature generation :

Given: message x, signature s, private key d and public key (p,q,α,β)

1. Choose an integer as random ephemeral key k_E with $0<k_E<q$
2. Compute $r \equiv (\alpha^{kE} \bmod p) \bmod q$
3. Computes $s \equiv (\mathrm{SHA}(x)+d \cdot r)\, k_E^{-1} \bmod q$

The signature consists of (r,s)

SHA denotes the hashfunction SHA-1 which computes a 160-bit fingerprint of message x.

Verification

DSA Signature Verification

1. Compute auxiliary value $w \equiv s^{-1} \bmod q$.
2. Compute auxiliary value $u_1 \equiv w \cdot SHA(x) \bmod q$.
3. Compute auxiliary value $u_2 \equiv w \cdot r \bmod q$.
4. Compute $v \equiv (\alpha^{u_1} \cdot \beta^{u_2} \bmod p) \bmod q$.
5. The verification $ver_{k_{pub}}(x,(r,s))$ follows from:

$$v \begin{cases} \equiv r \bmod q \Longrightarrow \text{valid signature} \\ \not\equiv r \bmod q \Longrightarrow \text{invalid signature} \end{cases}$$

Proof

We show need to show that the signature (*r,s*) in fact satisfied the condition *r* ≡ *v* mod *q:*

$s \equiv (\text{SHA}(x)) + d \cdot r) \cdot k_E^{-1} \bmod q$

$\Leftrightarrow \quad k_E \equiv s^{-1}\,\text{SHA}(x) + d \cdot s^{-1} r \bmod q$

$\Leftrightarrow \quad k_E \equiv u_1 + d \cdot u_2 \bmod q$

$because\ \ w = s^{-1} mod\ q$

$u_1 = w.sha(x) mod\ q$

We can raise α to either side of the equation if we reduce modulo *p:*

$\Leftrightarrow \quad \alpha^{kE} \bmod p \equiv \alpha^{u1+d \cdot u2} \bmod p$

Since β ≡ α^d mod *p* we can write:

$\Leftrightarrow \quad \alpha^{kE} \bmod p \equiv \alpha^{u1}\,\beta^{u2} \bmod p$

We now reduce both sides of the equation modulo q:

$\Leftrightarrow \quad (\alpha^{kE} \bmod p) \bmod q \equiv (\alpha^{u1}\,\beta^{u2} \bmod p) \bmod q$

Since *r* ≡ α^{kE} mod *p* mod *q* and *v* ≡ ($\alpha^{u1}\,\beta^{u2}$ mod *p*) mod *q*, this expression is identical to:

$\Leftrightarrow \quad r \equiv v$

proof

$$s \equiv (SHA(x) + d\,r)\,k_E^{-1} \bmod q$$

which is equivalent to:

$$k_E \equiv s^{-1}\,SHA(x) + d\,s^{-1}\,r \bmod q.$$

The right-hand side can be expressed in terms of the auxiliary values u_1 and u_2:

$$k_E \equiv u_1 + d\,u_2 \bmod q.$$

We can raise α to either side of the equation if we reduce modulo p:

$$\alpha^{k_E} \bmod p \equiv \alpha^{u_1 + d\,u_2} \bmod p.$$

Since the public key value β was computed as $\beta \equiv \alpha^d \bmod p$, we can write:

$$\alpha^{k_E} \bmod p \equiv \alpha^{u_1}\beta^{u_2} \bmod p.$$

We now reduce both sides of the equation modulo q:

$$(\alpha^{k_E} \bmod p) \bmod q \equiv (\alpha^{u_1}\beta^{u_2} \bmod p) \bmod q.$$

Since r was constructed as $r \equiv (\alpha^{k_E} \bmod p) \bmod q$ and $v \equiv (\alpha^{u_1}\beta^{u_2} \bmod p) \bmod q$, this expression is identical to the condition for verifying a signature as valid:

$$r \equiv v \bmod q.$$

Illustration

Alice

Bob

1. choose $p = 59$
2. choose $q = 29$
3. choose $\alpha = 3$
4. choose private key $d = 7$
5. $\beta = \alpha^d \equiv 4 \bmod 59$

$(p,q,\alpha,\beta)=(59,29,3,4)$ ←

sign:
compute hash of message $h(x) = 26$
1. choose ephemeral key $k_E = 10$
2. $r = (3^{10} \bmod 59) \equiv 20 \bmod 29$
3. $s = (26 + 7 \cdot 20) \cdot 3 \equiv 5 \bmod 29$

$(x,(r,s))=(x,(20,5))$ ←

verify:
1. $w = 5^{-1} \equiv 6 \bmod 29$
2. $u_1 = 6 \cdot 26 \equiv 11 \bmod 29$
3. $u_2 = 6 \cdot 20 \equiv 4 \bmod 29$
4. $v = (3^{11} \cdot 4^4 \bmod 59) \bmod 29 = 20$
5. $v \equiv r \bmod 29 \Longrightarrow$ valid signature

Elliptic Curve Digital Signature Algorithm

- Based on Elliptic Curve Cryptography (ECC)
- Bit lengths in the range of 160-256 bits can be chosen to provide security equivalent to 1024-3072 bit RSA (80-128 bit symmetric security level)
- One signature consists of two points, hence the signature is twice the used bit length (i.e., 320-512 bits for 80-128 bit security level).
- The shorter bit length of ECDSA often result in shorter processing time

Key Generation

Key Generation for ECDSA

1. Use an elliptic curve E with
   - modulus p
   - coefficients a and b
   - a point A which generates a cyclic group of prime order q
2. Choose a random integer d with $0 < d < q$.
3. Compute $B = dA$.

The keys are now:

$k_{pub} = (p, a, b, q, A, B)$

$k_{pr} = (d)$

Signature Algorithm

ECDSA Signature Generation

1. Choose an integer as random ephemeral key k_E with $0 < k_E < q$.
2. Compute $R = k_E A$.
3. Let $r = x_R$.
4. Compute $s \equiv (h(x) + d \cdot r)\, {k_E}^{-1} \bmod q$.

Verification

ECDSA Signature Verification

1. Compute auxiliary value $w \equiv s^{-1} \bmod q$.
2. Compute auxiliary value $u_1 \equiv w \cdot h(x) \bmod q$.
3. Compute auxiliary value $u_2 \equiv w \cdot r \bmod q$.
4. Compute $P = u_1 A + u_2 B$.
5. The verification $ver_{k_{pub}}(x,(r,s))$ follows from:

$$x_P \begin{cases} \equiv r \bmod q \Longrightarrow \text{valid signature} \\ \not\equiv r \bmod q \Longrightarrow \text{invalid signature} \end{cases}$$

Proof

Proof. We show that a signature (r,s) satisfies the verification condition $r \equiv x_P \bmod q$. We'll start with the signature parameter s:

$$s \equiv (h(x) + d\,r)\,{k_E}^{-1} \bmod q$$

which is equivalent to:

$$k_E \equiv s^{-1}\,h(x) + d\,s^{-1}\,r \bmod q.$$

The right-hand side can be expressed in terms of the auxiliary values u_1 and u_2:

$$k_E \equiv u_1 + d\,u_2 \bmod q.$$

Since the point A generates a cyclic group of order q, we can multiply both sides of the equation with A:

$$k_E\,A = (u_1 + d\,u_2)\,A.$$

Since the group operation is associative, we can write

$$k_E\,A = u_1\,A + d\,u_2\,A$$

and

$$k_E\,A = u_1\,A + u_2\,B.$$

Illustration

Alice

Bob

choose E with $p = 17$, $a = 2$, $b = 2$, and
$A = (5,1)$ with $q = 19$
choose $d = 7$
compute $B = dA = 7 \cdot (5,1) = (0,6)$

$\xleftarrow{(p,a,b,q,A,B)=(17,2,2,19,(5,1),(0,6))}$

sign:
compute hash of message $h(x) = 26$
choose ephemeral key $k_E = 10$
$R = 10 \cdot (5,1) = (7,11)$
$r = x_R = 7$
$s = (26 + 7 \cdot 7) \cdot 2 \equiv 17 \bmod 19$

$\xleftarrow{(x,(r,s))=(x,(7,17))}$

verify:
$w = 17^{-1} \equiv 9 \bmod 19$
$u_1 = 9 \cdot 26 \equiv 6 \bmod 19$
$u_2 = 9 \cdot 7 \equiv 6 \bmod 19$
$P = 6 \cdot (5,1) + 6 \cdot (0,6) = (7,11)$
$x_P \equiv r \bmod 19 \Longrightarrow$ valid signature