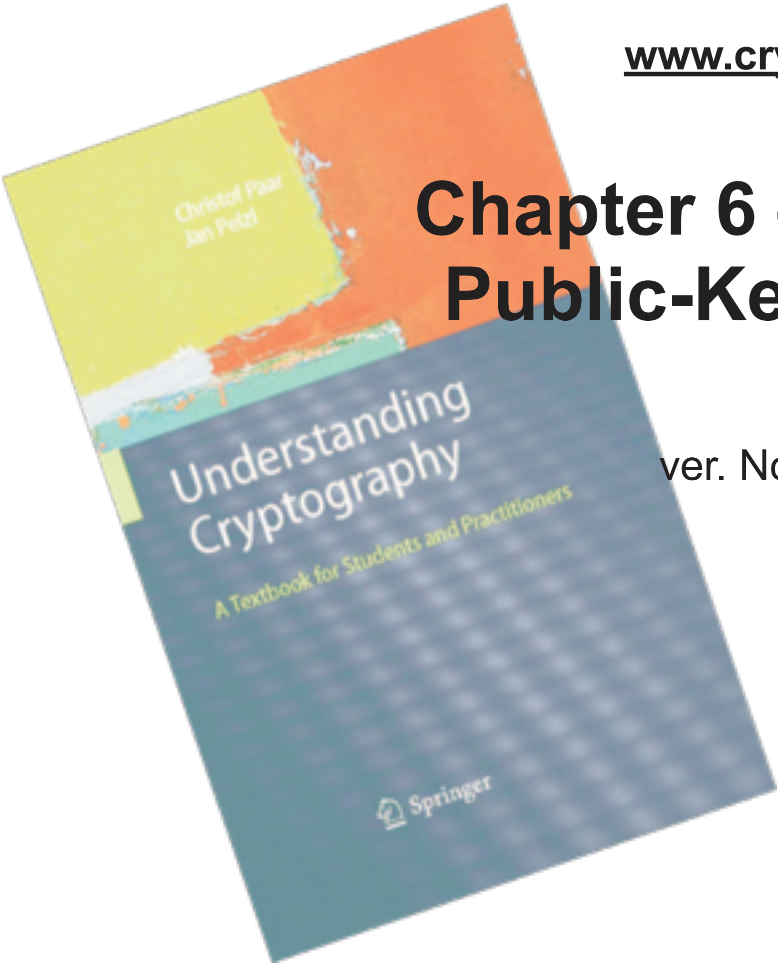
Understanding Cryptography – A Textbook for Students and Practitioners

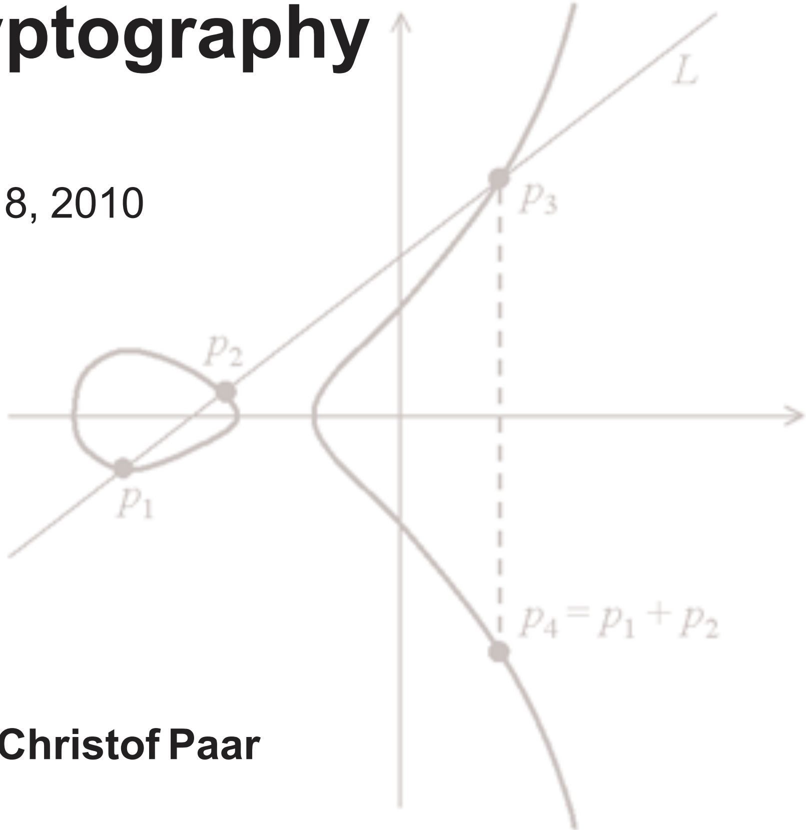
by Christof Paar and Jan Pelzl

www.crypto-textbook.com



Chapter 6 – Introduction to Public-Key Cryptography

ver. November 18, 2010



These slides were prepared by Timo Kasper and Christof Paar and modified by Sam Bowne -- revised 10-16-17

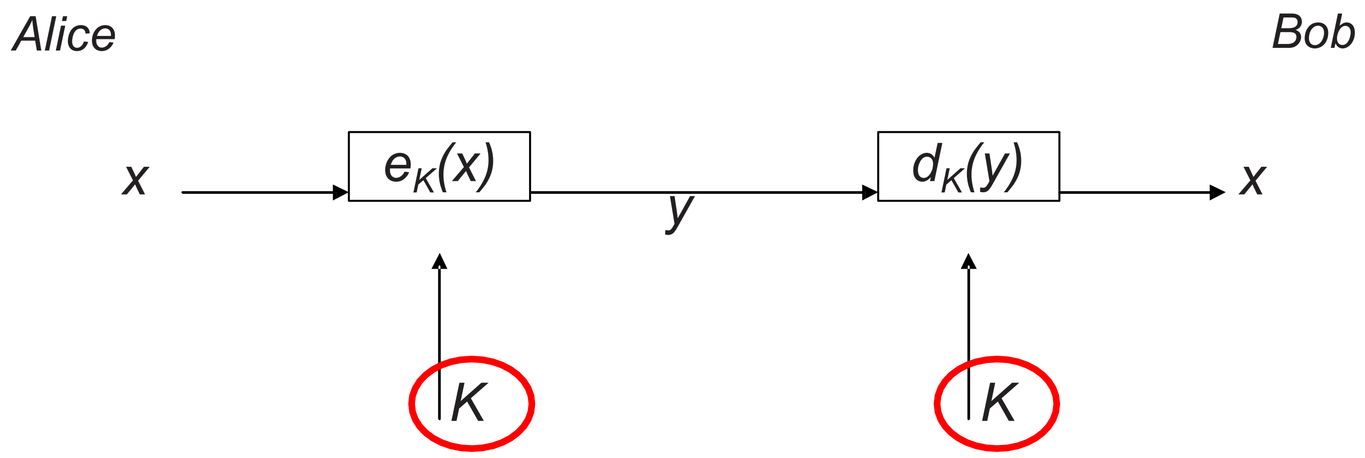
Some legal stuff (sorry): Terms of Use

Topics

- Symmetric Cryptography Revisited
- Principles of Asymmetric Cryptography
- Practical Aspects of Public-Key Cryptography
- Important Public-Key Algorithms
- *Essential Number Theory for Public-Key Algorithms (SKIP)*

Symmetric Cryptography Revisited

Symmetric Cryptography Revisited



- The **same secret key *K*** is used for encryption and decryption
- Encryption and Decryption are very similar (or even identical) functions

Symmetric Cryptography: Analogy

Safe with a lock, only Alice and Bob have a copy of the key

- Alice encrypts -- locks message in the safe with her key
- Bob decrypts -- uses his copy of the key to open the safe

Symmetric Cryptography: Shortcomings

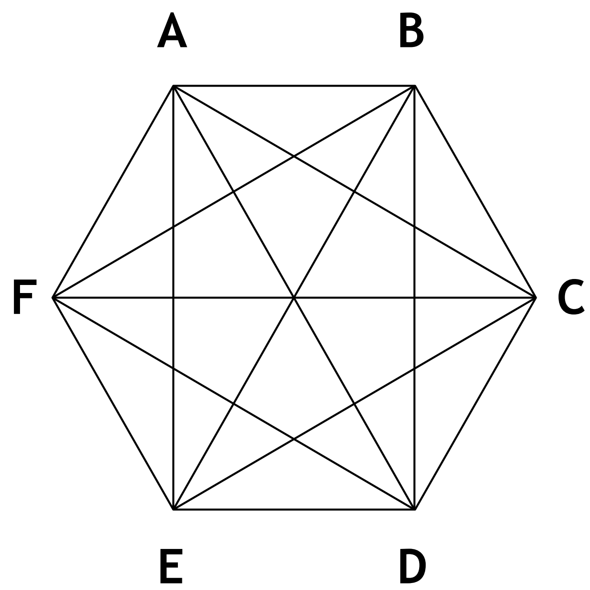
- Symmetric algorithms, e.g., AES or 3DES, are very secure, fast & widespread **but**:
  - **Key distribution problem**: The secret key must be transported securely
  - **Number of keys**: In a network, each pair of users requires an individual key

n users in the network require $\frac{n \cdot (n-1)}{2}$ keys, each user stores *(n-1)* keys

Example:

6 users (nodes)

$$\frac{6 \cdot 5}{2} = 15$$ keys (edges)



Symmetric Cryptography: Shortcomings

- Alice or Bob can **cheat each other**, because they have identical keys.
  - Alice can sign a contract, and later deny it
  - Bob could have faked the signature
  - Doesn't provide **non-repudiation**

Principles of Asymmetric Cryptography

Idea Behind Asymmetric Cryptography

New Idea:

Like a mailbox:

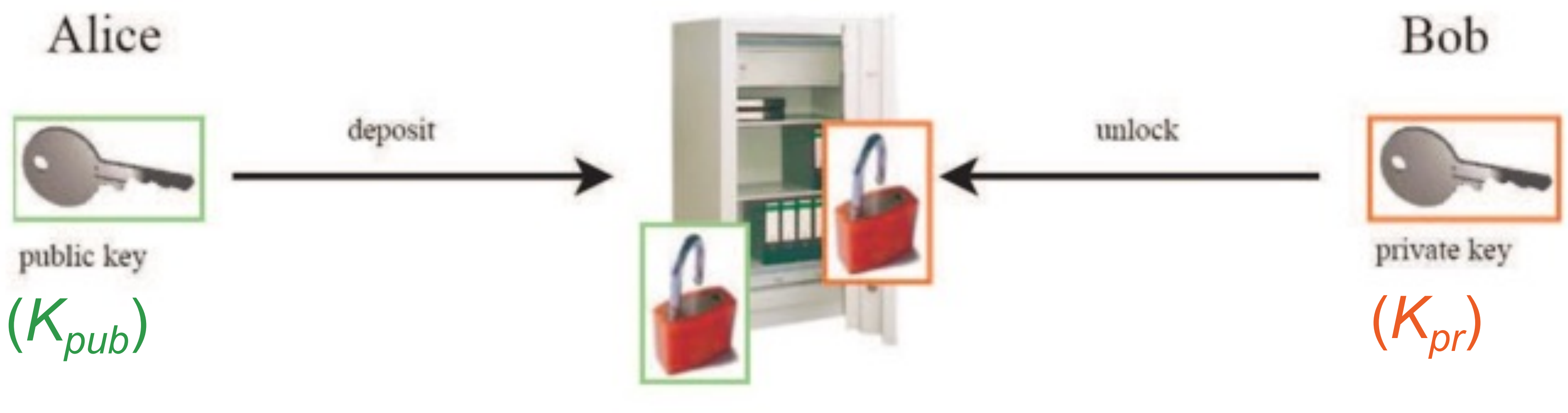
Everyone can drop a letter

But: Only the owner has the correct key to open the box

1976: first publication of such an algorithm by Whitfield Diffie and Martin Hellman,and also by Ralph Merkle.

Asymmetric Cryptography: Analogy

Safe with public lock and private lock:



- Alice deposits (encrypts) a message with Bob's - *not secret* - public key K_{pub}
- Only Bob has the - *secret* - private key K_{pr} to retrieve (decrypt) the message

Key Generation

- A message encrypted with a **public key** can be decrypted with the corresponding **private key**
  - The keys are related
- Each user must generate an individual **key pair**
  - Publish **public key** where everyone can find it
  - Protect **private key** so no one else gets it

One-Way Functions

- It must be easy to calculate the public key from the private key
  - So keys can be generated
- But difficult to calculate the private key from the public key
  - So attacker's can't get the private key

Definition 6.1.1 One-way function
A function $f()$ is a one-way function if:

1. $y = f(x)$ *is computationally easy, and*
2. $x = f^{-1}(y)$ *is computationally infeasible.*

Commonly Used One-Way Functions

- **Factorization**
  - Finding prime factors of a large number
  - **n** is known; find **p** and **q**

$$n = pq$$

- **Discrete logarithm**
  - Find an integer **x** satisfying this equation
  - **a**, **b**, and **p** are known

$$a^x = b \bmod p$$

Logarithms

- **Logarithms to base 10**
  - log(100) = 2, because $10^2 = 100$
  - log(1000) = 3, because $10^3 = 100$
  - log(2) = 0.301
- **Discrete logarithm**
  - Same thing, except on a ring and using only integers
  - Find an integer **x** satisfying this equation
  - **a**, **b** and **p** are known

$$a^x = b \bmod p$$

Kahoot!

Practical Aspects of Public-Key Cryptography

Basic Protocol for Public-Key Encryption

Alice | Bob

$(K_{pubB}, K_{prB}) = K$

K_{pubB} ←

x

$y = e_{K_{pubB}}(x)$

y →

$x = d_{K_{prB}}(y)$

Key Distribution Problem solved *

*at least for now; public keys need to be authenticated

Uses of Public-Key Cryptography

- **Key Distribution**
  - Diffie-Hellman key exchange (DHKE), RSA
  - Without a pre-shared secret (key)
- **Nonrepudiation**
  - RSA, DSA or ECDSA (Elliptic Curve Digital Signature Algorithm)
- **Identification**
  - Digital signatures
- **Encryption**
  - RSA, ECC (Elliptic Curve Cryptography), or Elgamal

Disadvantage of Public-Key Cryptography

- Computationally very intensive
- 1000 times slower than symmetric algorithms!

AES Encryption in Python

```
>>> from Crypto.Cipher import AES
>>> key = "aaaabbbbccccdddd"
>>> plain = "qqqqrrrrssssttttt"
>>> cipher = AES.new(key)
>>> cipher.encrypt(plain).encode("hex")
'f6be769eedef9d9ac10ff9c9103f4f67'
```

RSA Encryption in Python

```
>>> from Crypto.PublicKey import RSA
>>> key = RSA.generate(2048)
>>> publickey = key.publickey()
>>> plain = 'encrypt this message'
>>> ciphertext = publickey.encrypt(plain, 0)[0]
>>> print ciphertext.encode("hex")
536eda071ab9e526442f2b56e71fa5abfc603c88c2eac03d91f22bab6d0ea14bab2e8c8247df477c
5f15ce3ccc551227799d1f4f8943fa8bd278639bd90292c5799d11f9f6601c94d88f10fc314317fb
1d75f55e20d1c5dd4e7448ff39018dab44091b6664610657516bfaf95a3f0e63e9194f1e08343421
f7cf8c35550ed951b240e4c42f94b8bfc73ec3ccd519f7c489c28aaf799c78d6a695707423f72c05
4edfd8f4c2ac0f5c25a996647b8958f160983db8bdf2214fe131b0f3d558aeb7560e67f0621f0224
fd21f18034eebb9c8773e6310f80975539765d7235235a446f037179e94e504b21f9ffac6679570a
95848f238cdd3243723ed4722e549498
```

RSA Decryption in Python

```
>>> decrypted = key.decrypt(ciphertext)
>>> print decrypted
encrypt this message
```

Testing Speed in Python

```
import time
from Crypto.Cipher import AES
from Crypto.PublicKey import RSA

key = "aaaabbbbccccdddd"
plain = "ppppqqqqrrrrssss"
cipher = AES.new(key)

start = time.time()
for i in range(1000000):
  ciphertext = cipher.encrypt(plain)
print time.time() - start, " sec. for 1 million AES encryptions"

start = time.time()
for i in range(1000000):
  plaintext = cipher.decrypt(ciphertext)
print time.time() - start, " sec. for 1 million AES decryptions"
```

Testing Speed in Python

```
start = time.time()
key = RSA.generate(2048)
print time.time() - start, " sec. for one RSA-2048 key generation"

publickey = key.publickey()
plain = 'encrypt this message'

start = time.time()
for i in range(4000):
  ciphertext = publickey.encrypt(plain, 0)[0]
print time.time() - start, " sec. for 4000 RSA-2048 encryptions"

start = time.time()
for i in range(50):
  decrypted = key.decrypt(ciphertext)
print time.time() - start, " sec. for 50 RSA-2048 decryptions"
```

Testing Speed in Python

```
Sams-MacBook-Pro-3:proj sambowne$ python timeTEST
0.57718205452  sec. for 1 million AES encryptions
0.56579208374  sec. for 1 million AES decryptions
0.908615112305  sec. for one RSA-2048 key generation
0.633729934692  sec. for 4000 RSA-2048 encryptions
0.455893993378  sec. for 50 RSA-2048 decryptions
```

Basic Key Transport Protocol 1/2

In practice: **Hybrid systems**, incorporating asymmetric and symmetric algorithms

1. **Key exchange** (for symmetric schemes) and **digital signatures** are performed with (slow) **asymmetric** algorithms

2. **Encryption** of data is done using (fast) symmetric ciphers, e.g., **block ciphers or stream ciphers**

Basic Key Transport Protocol 2/2

Example: Hybrid protocol with AES as the symmetric cipher

Alice		Bob
	$\xleftarrow{K_{pubB}}$	$(K_{pubB}, K_{prB}) = K$
Choose random symmetric key K		
$y_1 = e_{K_{pubB}}(K)$	$\xrightarrow{y_1}$	
		$K = d_{K_{prB}}(y_1)$
message x		
$y_2 = AES_K(x)$	$\xrightarrow{y_2}$	$x = AES^{-1}_K(y_2)$

Key Exchange (asymmetric)

Data Encryption (symmetric)

Remaining Problem: Key Authenticity

- Alice wants to send a message to Bob
- Attacker can publish a fake public key for Bob
- Alice uses the fake key, so the attacker can read the message

- The current solution is **digital certificates** and **Certificate Authorities**

Important Public-Key Algorithms

Public-Key Algorithm Families of Practical Relevance

Integer-Factorization Schemes Several public-key schemes are based on the fact that it is difficult to factor large integers. The most prominent representative of this algorithm family is RSA.

Discrete Logarithm Schemes There are several algorithms which are based on what is known as the discrete logarithm problem in finite fields. The most prominent examples include the Diffie–Hellman key exchange, Elgamal encryption or the Digital Signature Algorithm (DSA).

Elliptic Curve (EC) Schemes A generalization of the discrete logarithm algorithm are elliptic curve public-key schemes. The most popular examples include Elliptic Curve Diffie–Hellman key exchange (ECDH) and the Elliptic Curve Digital Signature Algorithm (ECDSA).

Key Lengths and Security Levels

Symmetric	*ECC*	*RSA, DL*	*Remark*
64 Bit	128 Bit	≈ 700 Bit	Only short term security (a few hours or days)
80 Bit	160 Bit	≈ 1024 Bit	Medium security (except attacks from big governmental institutions etc.)
128 Bit	**256 Bit**	≈ **3072 Bit**	**Long term security (without quantum computers)**

Quantum Computers

- The existence of quantum computers would probably be the end for ECC, RSA & DL
- TEXTBOOK SAYS:
  - *At least 2-3 decades away, and some people doubt that QC will ever exist*

NIST Recommendations from 2016

SP 800-57 Part 1 Rev. 4

Recommendation for Key Management, Part 1: General

Date Published: January 2016

Supersedes: SP 800-57 Part 1 Rev. 3 (July 2012);

NIST Recommendations from 2016

Table 2: Comparable strengths

Security Strength	Symmetric key algorithms	FFC (e.g., DSA, D-H)	IFC (e.g., RSA)	ECC (e.g., ECDSA)
≤ 80	2TDEA[21]	$L = 1024$ $N = 160$	$k = 1024$	$f = 160\text{-}223$
112	3TDEA	$L = 2048$ $N = 224$	$k = 2048$	$f = 224\text{-}255$
128	AES-128	$L = 3072$ $N = 256$	$k = 3072$	$f = 256\text{-}383$
192	AES-192	$L = 7680$ $N = 384$	$k = 7680$	$f = 384\text{-}511$
256	AES-256	$L = 15360$ $N = 512$	$k = 15360$	$f = 512+$

NIST Recommendations from 2016

Table 4: Security-strength time frames

<table>
<tr><th colspan="2">Security Strength</th><th>Through 2030</th><th>2031 and Beyond</th></tr>
<tr><td rowspan="2">< 112</td><td rowspan="7"></td><td colspan="2">Disallowed</td></tr>
<tr><td colspan="2">Legacy-use</td></tr>
<tr><td rowspan="2">112</td><td rowspan="2">Acceptable</td><td>Disallowed</td></tr>
<tr><td>Legacy use</td></tr>
<tr><td>128</td><td>Acceptable</td><td>Acceptable</td></tr>
<tr><td>192</td><td>Acceptable</td><td>Acceptable</td></tr>
<tr><td>256</td><td>Acceptable</td><td>Acceptable</td></tr>
</table>

NIST Recommendations from 2016

```
LENGTH:  1024
0.150621891022  sec. for one RSA key generation
0.026349067688  sec. for 400 RSA encryptions
0.0133030414581  sec. for 5 RSA decryptions

LENGTH:  2048
1.00856208801  sec. for one RSA key generation
0.0688228607178  sec. for 400 RSA encryptions
0.0520279407501  sec. for 5 RSA decryptions

LENGTH:  3072
6.82681417465  sec. for one RSA key generation
0.123769044876  sec. for 400 RSA encryptions
0.134315013885  sec. for 5 RSA decryptions

LENGTH:  7680
46.0725779533  sec. for one RSA key generation
0.527646064758  sec. for 400 RSA encryptions
1.50095796585  sec. for 5 RSA decryptions

LENGTH:  15360
170.346763134  sec. for one RSA key generation
1.78894495964  sec. for 400 RSA encryptions
10.7478890419  sec. for 5 RSA decryptions
```

Lessons Learned

- Public-key algorithms have **capabilities that symmetric ciphers don't have**, in particular digital signature and key establishment functions.

- Public-key algorithms are **computationally intensive** (a nice way of saying that they are *slow*), and hence are poorly suited for bulk data encryption.

- Only **three families of public-key schemes** are widely used. This is considerably fewer than in the case of symmetric algorithms.

- The **extended Euclidean algorithm** allows us to compute **modular inverses** quickly, which is important for almost all public-key schemes.

- **Euler's phi function** gives us the number of elements smaller than an integer n that are relatively prime to n. This is important for the RSA crypto scheme.

Kahoot!