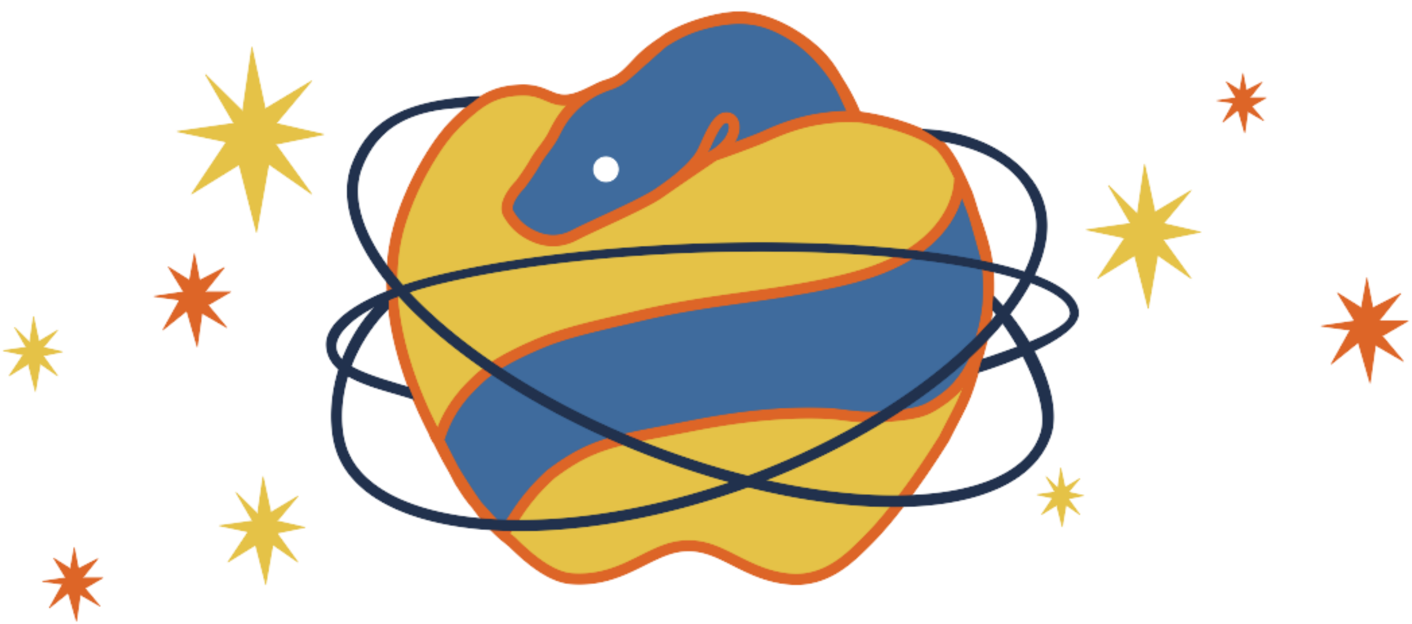
Cryptography in Python

Amirali Sanatinia
amirali@ccs.neu.edu

Northeastern University

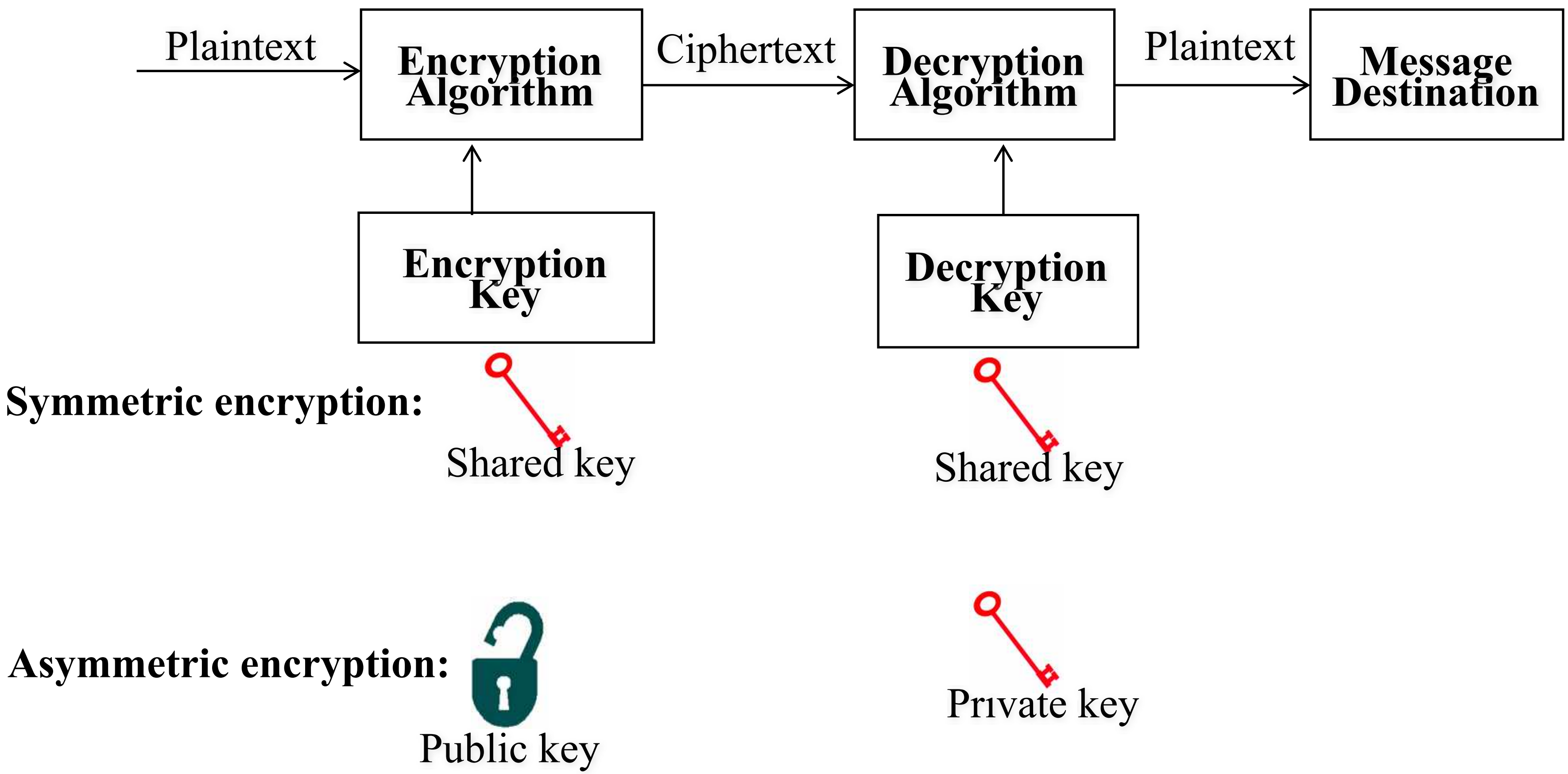


Cryptography

- Cryptography is ubiquitous today
- From mobile phones to wireless connections
- Supported in almost every programming language
- It is even embedded in the CPUs
- It is not hard to do crypto right but …

Crypto Failures

Encryption Models



Symmetric vs. Asymmetric Encryption

- Symmetric algorithms are much faster
  - In the order of a 1000 times faster

- Symmetric algorithms require a shared secret
  - Impractical if the communicating entities don't have another secure channel

- Both algorithms are combined to provide practical and efficient secure communication
  - E.g., establish a secret session key using asymmetric crypto and use symmetric crypto for encrypting the traffic

Advanced Encryption Standard (AES)

- Also known as Rijndael
- Part of NIST competition
- Requirements
  - Fast in software and hardware
  - Block size: 128; Key size: 128, 192 and 256
- Joan Daemen and Vincent Rijmen
- First published in 1998
- FIPS 197 on November 26, 2001
- Other candidates: Mars, RC6, Serpent, Twofish

Block Cipher Mode of Operation

- AES works on a block of data (128 bits)
- To encrypt a large message, each block needs to be encrypted
- Different modes of encrypting the blocks
  - ~~Electronic Codebook (ECB)~~
  - Cipher Block Chaining (CBC)
  - Counter (CTR)

ECB vs. CBC

Original

ECB

CBC

RSA

- One of the first practical public crypto systems
- Designed by Ron Rivest, Adi Shamir, and Leonard Adleman
- First published in 1977
- Was patented until September 2000
- Based on the hardness of factoring problem and modular arithmetic

Textbook RSA

- $E(M) = M^e \bmod n = C$ **(Encryption)**
- $D(C) = C^d \bmod n = M$ **(Decryption)**

- RSA parameters and basic (not secure) operations:
  - ***p*, *q*, two big prime numbers** **(private, chosen)**
  - $n = pq$, $\phi(n) = (p-1)(q-1)$ **(public, calculated)**
  - e, with $\gcd(\phi(n), e) = 1$, $1<e<\phi(n)$ **(public, chosen)**
  - $d = e^{-1} \bmod \phi(n)$ **(private, calculated)**

- $D(E(M)) = M^{ed} \bmod n = M^{k\phi(n)+1} = M$ **(Euler's theorem)**

Example of RSA

- Keys generation:
  - $p = 5$; $q = 11 \Rightarrow n = 55$
  - $e = 3 \Rightarrow d = 27$
    - Because $ed = 1 \bmod (p-1)(q-1)$
  - Public key: (e, n); Private Key: (d, n)
- Encryption
  - $M = 2$
  - Encryption(M) = $M^e \bmod n = 8$
  - Decryption(8) = $8^d \bmod n = 2$

Hashing Functions

- Input: long message
- Output: short block (called *hash* or *message digest*)
- Desired properties:
  - Pre-image: Given a hash h it is computationally infeasible to find a message m that produces h
  - Second preimage: Given message m, it is computationally infeasible to find a message m', ($m \neq m'$) such that, $h(m) = h(m')$
  - Collisions: It is computationally difficult to find any two messages m, m' ($m \neq m'$) such that, $h(m) = h(m')$

- Examples
  - Recommended Hash Algorithm (SHA-2, SHA-3) by NIST
  - SHA-1: output 160 bits being phased out
  - MD2, MD4, and MD5 by Ron Rivest [RFC1319, 1320, 1321]

Python Crypto Libraries

- PyCrypto
  - Oldest and most widely used
- M2Crypto
  - SWIG binding
- Cryptography*
  - PY2, PY3, PyPy
  - OpenSSL CFFI binding
- PyNaCl , python-nss, etc.

Cryptography In Action (SHA2)

```
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import hashes

digest = hashes.Hash(hashes.SHA256(), backend=default_backend())
digest.update(b"PyGotham 2016")
msg_digest = digest.finalize()

# Base64 encoded
'R/4s/kl2DD6keNedhSZlr/cg5cCDFaAknwQprCYAR38='
```

Cryptography In Action
(AES Encryption/Decryption)

```
import os
from cryptography.hazmat.primitives.ciphers import Cipher
from cryptography.hazmat.primitives.ciphers import algorithms
from cryptography.hazmat.primitives.ciphers import modes
from cryptography.hazmat.backends import default_backend
key = os.urandom(16) # use urandom to generate random number
iv = os.urandom(16)  # in bytes, 128 bits

# CBC Mode, we need an IV
cipher = Cipher(algorithms.AES(key), modes.CBC(iv),
    backend=default_backend())
encryptor = cipher.encryptor()
# We don't need padding here, since len("PyGotham16Crypto") = 16
cipher_text = encryptor.update("PyGotham16Crypto") + encryptor.finalize()

# create the decryptor object
decryptor = cipher.decryptor()
decryptor.update(cipher_text) + decryptor.finalize()
```

Cryptography In Action (RSA Key Generation)

```python
# Generate a 2048 bit private key
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.asymmetric import rsa

private_key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048,
    backend=default_backend())

# to get the public key
public_key = private_key.public_key()
```

Cryptography In Action
(RSA Encryption/Decryption)

```python
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import padding

message = b"The SECRET KEY"
ciphertext = public_key.encrypt(
    message,
    padding.OAEP(
        mgf=padding.MGF1(algorithm=hashes.SHA1()),
        algorithm=hashes.SHA1(),
        label=None))

# The same as encryption, but using the private key instead
plaintext = private_key.decrypt(
    ciphertext, ...)
```

Cryptography In Action (Fernet)

- Provides authenticated encryption
  - AES in CBC mode, 128 bit key, PKCS7 padding
  - SHA256 HMAC for authentication

```python
from cryptography.fernet import Fernet
key = Fernet.generate_key()
fernet = Fernet(key)
token = fernet.encrypt(b"PyGotham16Crypto")
fernet.decrypt(token)
```

Takeaways

- Don’t invent your own crypto algorithm
- Don’t implement your own crypto library
- Doing crypto in a right way is not difficult
- Use SSL for data in transit
- Use PGP for data at rest

Thank You!

Questions?