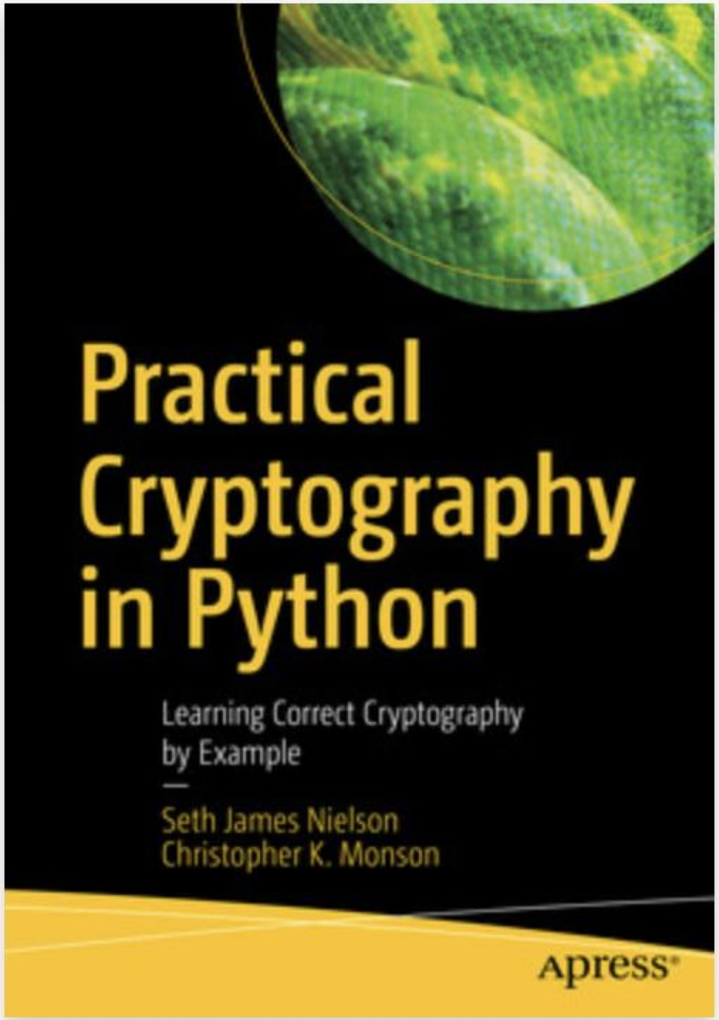
Systems Security Group

Lancaster University

Week 14 Asymmetric encryption

Recommended reading



The book is available to you via the library

Technology stack

- Python 3
  Link to a Python Cheat Sheet
- cryptography.io
  Link to the library

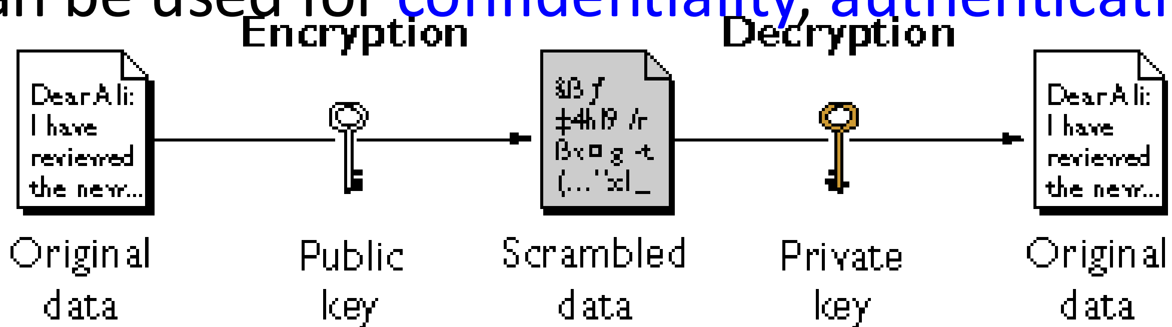
Topics

- Asymmetric encryption
- RSA
- Diffie-Hellman

Recommended reading: Chapter 4 from the book of "Practical Cryptography in Python"

Asymmetric cryptography

- A cryptographic scheme is called asymmetric if *d* <> *e and* it is computationally infeasible in practice to compute *d* out of *e*.
- In asymmetric cryptography *e* goes public and *d* is kept as a secret.
  - Anybody can use *e* to encrypt a plaintext and only the one that has *d* can decrypt it.
  - Public key cryptographic schemes.
  - Can be used for confidentiality, authentication or both



RSA#0

- Ron **R**ivest, Adi **S**hamir, Len **A**dleman (RSA)
- General-purpose approach to public-key encryption

RSA#1

- Plaintext is encrypted in blocks
  - Each block has a binary value less than *n*
  - Block size is *i* bits $2^i < n \leq 2^{i+1}$
- M is the plaintext and C the ciphertext

$$C = M^e mod\, n$$

$$M = C^d mod\, n = (M^e)^d mod\, n = M^{ed} mod\, n$$

RSA#3

- In reality e = 2^16+1 = 65537

- Order of generating keys
  - First, create a private key object.
  - Then, get a public key object corresponding to the values of the private key.

https://cryptography.io/en/latest/hazmat/primitives/asymmetric/rsa/

Serialization of keys

- Encode keys using PEM: Privacy Enhanced Mail; it simply indicates a base64 encoding with header and footer lines
- Set format for private key to PKCS8: A more modern format for serializing keys which allows for better encryption. Choose this unless you have explicit legacy compatibility requirements.
- Set format for public key to SubjectPublicKeyInfo: This is the typical public key format. It consists of an algorithm identifier and the public key as a bit string. Choose this unless you have specific needs.
- Use or not an encryption algorithm

Converting key objects to/from PEM

Key object to PEM

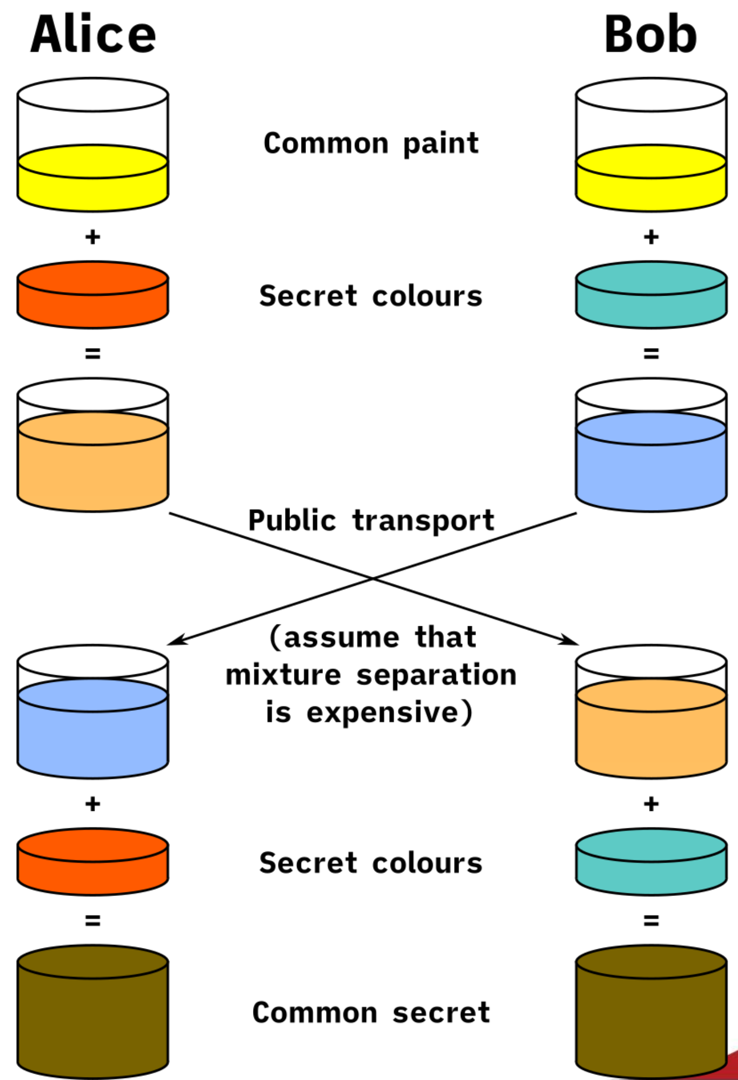
```
.private_bytes(select encoding, etc)
.public_bytes(select encoding, etc)
```

PEM to key object

```
.serialization.load_pem_private_key
.serialization.load_pem_public_key
```

Diffie-Hellman key exchange

- Used to securely exchange a key
- Based on discrete logarithms



[source: 3]

Diffie-Hellman

- Alice selects:
  - $X_A < q$ and computes $Y_A = \alpha^{X_A}\ mod\ q$
- Bob selects:
  - $X_B < q$ and computes $Y_B = \alpha^{X_B}\ mod\ q$
- X values are kept private and Y are sent away.
- Alice computes the key:
  - $K = (Y_B)^{X_A}\ mod\ q$
- Bod computes the key:
  - $K = (Y_A)^{X_B}\ mod\ q$

Systems Security Group | Lancaster University

Diffie-Hellman

- The calculated key is the same:

$$K = (Y_B)^{X_A} mod\ q$$

$$= (\alpha^{X_B} mod\ q)^{X_A} mod\ q$$

$$= (\alpha^{X_B})^{X_A} mod\ q$$

$$= (\alpha^{X_A})^{X_B} mod\ q$$

$$= (\alpha^{X_A}\ mod\ q)^{X_B} mod\ q$$

$$= (Y_A)^{X_B} mod\ q$$

Structure of your code…

Modules you want to import

```
import XYZ
```

List of functions you implement

```
def  myFunction():
     # TODO

     return # TODO
```

Have a main section to call your functions

```
if __name__ == "__main__":
     x = myFunction()
```