Chapter 1

General knowledge

1.1 Converting data

Binary to hexadecimal

Use python built-in functions.

```
bin_data = b"data"
hex_data = bin_data.hex()
bin_data = bytes.fromhex(hex_data)
```

Binary to base 64

Use python module **base64**.

```
import base64

bin_data = b"data"
b64_data = base64.b64encode(bin_data)
bin_data = base64.b64decode(b64_data)
```

1.2 PKCS7 Padding

Some block cyphers might require padding. The order of operation must always be:

1. Pad data
2. Encrypt padded data

3. Exchange data
4. Decrypt encrypted data
5. Unpad plaintext

```
from cryptography.hazmat.primitives import padding

padder = padding.PKCS7(block_size).padder()
padded_data = padder.update(data) + padder.finalize()

unpadder = padding.PKCS7(block_size).unpadder()
data = unpadder.update(padded_data) + unpadder.finalize()
```

1.3 Randomization

Random keystream

You can generate a random string of bytes by using an OS function.

```
import os

rand = os.urandom(num_bytes)
```

LFSR keystream

This is included in the **pylfsr** module.

```
from pylfsr import LFSR

seed = [0, 0, 0, 1, 0]
fpoly = [3, 2, 1]  # c3=1, c2=1, c1=1
L = LFSR(fpoly = fpoly, initstate = seed, verbose = True)

seq = L.runKCycle(num_bits)
```

Chapter 2

Symmetric encryption

2.1 AES256

AES256 is a **block cypher algorithm** with a 32B key, and it can use different modes of operation. The general syntax is:

```
from cryptography.hazmat.primitives.ciphers \
    import Cipher, algorithms, modes

block_size = 16  # e.g.
key = os.urandom(32)
iv = os.urandom(block_size)  # if needed, depends on mode
nonce = os.urandom(block_size)  # if needed, depends on mode

message = b"A secret message"  # must be binary

cypher = Cypher(algorithms.AES(key), <mode>)

encryptor = cypher.encryptor()
ct = encryptor.update(message) + encryptor.finalize

decryptor = cypher.decryptor()
pt = decryptor.update(ct) + decryptor.finalize
```

CBC mode

Needs an Initialization Vector.

```
cypher = Cypher(algorithms.AES(key), modes.CBC(iv))
```

ECB mode

Padding is required.

```
cypher = Cypher(algorithms.AES(key), modes.ECB())
```

CTR mode

Requires a nonce (unique and never reused). This mode is not reccomended for block cyphers with a block size of less than 128b.

```
cypher = Cypher(algorithms.AES(key), modes.CTR(nonce))
```

Effects of modifying ciphertexts in different modes

- **CBC and ECB modes:** The entire block of the altered byte is corrupted.
- **CTR mode:** Only the affected byte is corrupted.

2.2 ChaCha20

ChaCha20 is a **stream cypher algorithm**. It requires a 32B key and a 16B nonce.

```
nonce = os.urandom(16)

cipher = Cipher(algorithms.ChaCha20(key, nonce), mode=None)

encryptor = cipher.encryptor()
ct = encryptor.update(message)

decryptor = cipher.decryptor()
pt = decryptor.update(ct)
```