ICT2213—Applied Cryptography

Lab 2: Symmetric-key Cryptography

1 Python's `cryptography` package

The `cryptography` package provides the most common cryptographic primitives, such as symmetric ciphers, public-key algorithms, secure hash functions, and many more. It depends on the OpenSSL C library for all cryptographic operations. OpenSSL is the de facto standard for implementing cryptographic protocols in software. You can find an extensive documentation on how to properly use `cryptography` at `https://cryptography.io`. You can install the package with `pip` as follows:

```
$ pip install cryptography
```

The `cryptography` package is divided into two levels: one with safe cryptographic *recipes* and the other with *low-level* cryptographic primitives. The recipes require little to no configuration choices, and are safe and easy to use. On the other hand, the low-level cryptographic primitives are dangerous, if used incorrectly. They require making decisions and having an in-depth knowledge of the cryptographic concepts at work. Because of the potential danger in working at this level, it is referred to as the "hazardous materials" or "hazmat" layer, and the underlying primitives are located in the `cryptography.hazmat` package.

2 Fernet (symmetric encryption)

Fernet is a recipe that performs authenticated encryption, i.e., it guarantees that a message cannot be modified or read without knowledge of the secret key. Being a recipe, it can be applied without any configurations. Specifically, it applies AES encryption with a 128-bit key, and HMAC authentication (using SHA-256) with a different 128-bit key. The following script encrypts and then decrypts a message:

```
from cryptography.fernet import Fernet

key = Fernet.generate_key()
f = Fernet(key)

token = f.encrypt(b"My secret message")
print(token)

pt = f.decrypt(token)
print(pt)
```

The output of the `encrypt()` function, known as a "Fernet token", is a URL-safe base64-encoded `bytes` object. It encodes the concatenation of the following fields: Version, Timestamp, IV, Ciphertext, and HMAC. The input to the function must also be a `bytes` object.

3 Symmetric encryption

The low-level symmetric encryption primitives are located in the `cryptography.hazmat` package. There, all the parameters have to set manually by the developer. First, you have to create a `Cipher` object that combines an algorithm with a mode, and then create an encrypting or decrypting `CipherContext` instance, using the `encryptor()` or `decryptor()` functions. To encrypt or decrypt, you need to invoke both the `update()` and `finalize()` functions. Note that, these low-level functions do **not** automatically pad messages whose size is not a multiple of the cipher's block size. As such, you need to manually pad the message using a standard padding, such as PKCS7. However, padding is not required for stream ciphers, such as CTR or OFB modes. The following Python script summarizes all the aforementioned operations:

```python
import os
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives import padding

# 128 is the cipher's block size in bits (16 bytes for AES)
padder = padding.PKCS7(128).padder()
padded_data = padder.update(b"This is a secret message") + padder.finalize()
print(padded_data)

# Always use the OS's urandom function to generate random values
# such as keys and IVs
key = os.urandom(32)
iv = os.urandom(16)
cipher = Cipher(algorithms.AES(key), modes.CBC(iv))
encryptor = cipher.encryptor()
ct = encryptor.update(padded_data) + encryptor.finalize()

decryptor = cipher.decryptor()
data = decryptor.update(ct) + decryptor.finalize()

unpadder = padding.PKCS7(128).unpadder()
pt = unpadder.update(data) + unpadder.finalize()
print(pt)
```

4 Experiments

Once you have familiarized yourselves with the low-level symmetric encryption primitives, try to perform the attacks described in the lecture notes. For ECB mode, you can try to add, remove or interchange blocks of ciphertext. For CBC, OFB or CTR modes, try to perform the bit flipping attacks.