Cryptography Lab - List 2 Report

Gabriel Tański

26.04.2020

1 Basic information

1.1 AES - background

AES stands for Advanced Encryption Standard. It is a symmetric block cipher standardized by NIST in 2001. It superseded DES and 3DES as the go-to encryption algorithm. DES is based on Feistel Networks, was standardized in 1970s, had a key length of 56 bits, block size of 32 bits. It was deemed insecure from its conception, and was ultimately 'easily' broken in 1998. 3DES is a three-fold application of DES, which uses a 168-bit key. AES uses a 128-bit block size and a key size of 128, 192, or 256 bits.

1.2 IV vs nonce

IV stands for Initialization Vector and nonce is a short for a number used once. Both are used in various modes of encryption to provide some variance of ciphertexts when identical blocks are encrypted with the same key. Sometimes a nonce can be called a counter, because the only requirement for nonce is to be used once, and so under certain circumstances using consecutive integers is enough to provide secure encryption. IV is also a nonce, but there is one more requirement for IVs - they need to be unpredictable. Knowing one or more previous IVs cannot give the attacker any advantage in predicting the next one.

1.3 AES Modes

Five AES modes that were implemented in the solution and one 'standard' mode are shortly described below:

- ECB (Electronic Code Book) - The most rudimentary of AES modes. It does not use any kind of IV or nonce, so identical plaintexts will yield identical ciphertexts when encrypted using ECB.
- CBC (Cipher Block Chaining) - This mode uses a 16-byte IV and requires the plaintext to be padded to a multiple of 16 bytes. This mode will be explained in detail later on.
- CTR (Counter) - Uses block counters mixed with a nonce. Such a value is then encrypted and the ciphertext is XOR-ed with message block to produce a ciphertext block. Each block of the message is encrypted independently and can be decrypted independently.
- OFB (Output Feedback) - Has some parallels with CBC and CTR. An IV is encrypted, then it is XOR-ed with a block of plaintext to produce a block of ciphertext, but the encrypted IV is also passed as an IV for the next block in the chain.
- CFB (Cipher Feedback) - Very similar to OFB, but the resulting block of ciphertext is passed as an IV for the next block.

- EAX (Encrypt-then-Authenticate-then-Translate) - this mode is a bit more advanced and uses two passes to achieve privacy and authenticity of each block. Uses a nonce.

1.4 CBC mode

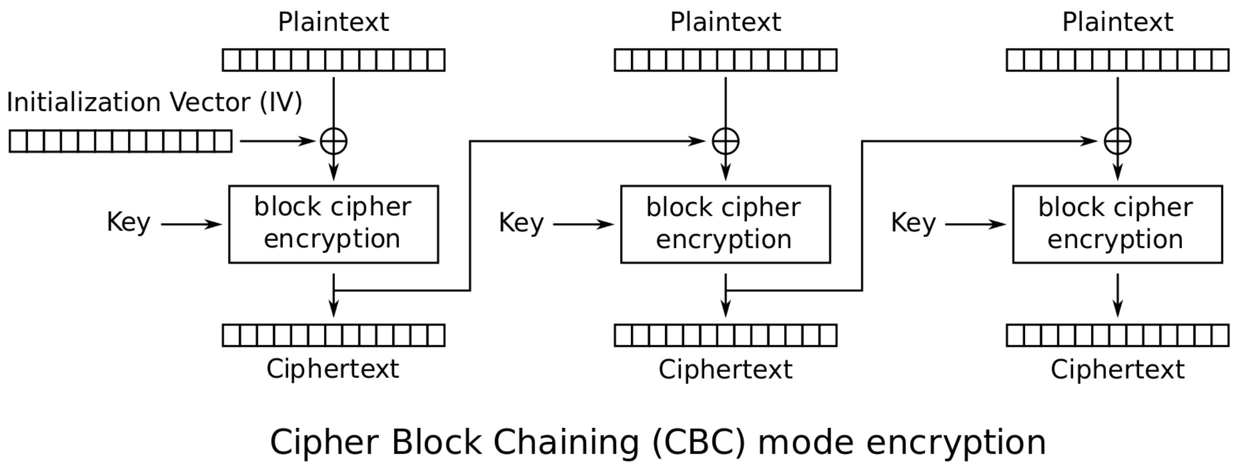

Figure 1: source

As we can see, the scheme is relatively simple. Every block of plaintext gets XOR-ed with some value - for the first block it is the IV passed to the algorithm, and for the following blocks their 'IV' is the ciphertext of the previous block. The blocks are chained, which puts some limitations on the speed of encrypting and decrypting large amounts of data, as well as makes it impossible to update encrypted data without decrypting everything.

2 The CPA experiment

2.1 The attack

In order to make this attack successful, one (legal) assumption will be made - the adversary $\mathcal{A}$ is able to predict the values of the IV that will be used. All messages referred to in the experiment have the length equal to the block size of AES (16 bytes).

1. A secret key sk is generated on the side of the Oracle.
2. $\mathcal{A}$ picks a random message m.
3. $\mathcal{A}$ asks the Oracle for the value of $Enc_{sk}(m)$. henceforth denoted by c, and also obtains iv used for that encryption.
4. Using the assumption, $\mathcal{A}$ predicts the next IV to be iv_1.
5. $\mathcal{A}$ calculates $m_{i_0} = (m \oplus iv) \oplus iv_i$ and takes a random $m_{i_1} \neq m_{i_0}$.
6. $\mathcal{A}$ sends a tuple (m_{i_0}, m_{i_1}) as a challenge for the oracle. If the resulting ciphertext is c, $\mathcal{A}$ outputs $b' = 0$. Otherwise $\mathcal{A}$ outputs $b' = 1$.
7. Steps $5 - 6$ can be repeated any number of times, where i will be the index of the challenge sent.
8. After n challenges, $\mathcal{A}$ succeeds with a probability of $1 \geq \frac{1}{2} + negl(n)$.

2.2 Explanation

The attack heavily relies on the assumption that $\mathcal{A}$ is able to predict the consecutive IVs used by the oracle. That assumption is allowed, as the Oracle is flawed in a way that gives this advantage to $\mathcal{A}$.

In the third step, $\mathcal{A}$ asks for a ciphertext that will be used for reference and gets the IV that was used to obtain it. In step 5, $\mathcal{A}$ calculates a value of $m_{i_0} = (m \oplus iv) \oplus iv_i$. Remembering that the plaintext is XOR-ed with the IV before encryption, we know that the value that was encrypted in step 3, was actually $m \oplus iv$, and that it yielded c. Knowing the next IV, iv_i, enables $\mathcal{A}$ to choose m_{i_0} so that $Enc_{sk}(m_{i_0} \oplus iv_i) = Enc_{sk}(m \oplus iv) = c$. Knowing both IVs and the original message, $\mathcal{A}$ can easily calculate the message m_{i_0}. Therefore $\mathcal{A}$ always picks a m_{i_0}, such that XOR-ing it with iv_i will give a plaintext that gets encrypted to c. Therefore, when oracle returns the ciphertext, $\mathcal{A}$ <u>always</u> knows whether the ciphertext was created using the carefully 'selected' m_{i_0} or a random m_{i_1}.

3 Further information

If the program sent with to this report malfunctions, please inform me. The CPA experiment can be started by running

```
python task2.py
```

Assuming that `python` points to a Python interpreter with version ≥ 3.8. There is at least one syntactic feature incompatible with older versions of Python. The encryption oracle can also be queried manually, by running `task1.py` script with correct parameters. The information about the parameters can be obtained using:

```
python task1.py --help
```

Additionally, the code is shipped with `keystore.jks` file, which can be used for testing the software. Unfortunately the library used for handling keystores supports only `jks` and `jceks` formats. The keystore can be created with one of these formats using the keytool flag `-storetype jceks` or `-storetype jks` when creating the keystore.