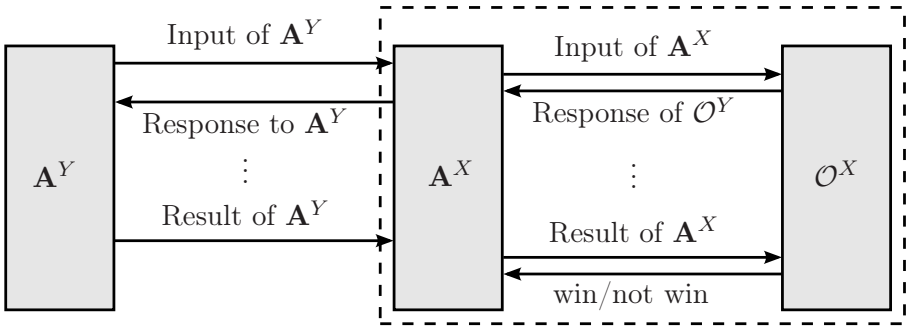
Problem Set 2
Course **Secure Channels**
(Summer Term 2018)

Bauhaus-Universität Weimar, Chair of Media Security
Prof. Dr. Stefan Lucks, Eik List
URL: `http://www.uni-weimar.de/de/medien/professuren/mediensicherheit/teaching/`

Due Date: 03 May 2018, 11:00 AM, via email to `eik.list(at)uni-weimar.de`.

Proof Hints: *In this problem set, you will deepen your understanding in security notions and their relations. In general, to disprove a statement, it suffices to describe a counter example. To prove a relation $X \implies Y$, show the opposite: If there would exist a successful $\mathbf{A}^Y$ adversary, one could use it also for constructing a successful $\mathbf{A}^X$ adversary. So, assume there exists a successful Y-adversary $\mathbf{A}^Y$. Then, describe an X-adversary $\mathbf{A}^X$ (a simulator) that interacts with an X-oracle $\mathcal{O}^X$. $\mathbf{A}^X$ "simulates" a Y oracle in the view of $\mathbf{A}^Y$. The actual goal of $\mathbf{A}^X$ is to use the output of $\mathbf{A}^Y$ to win the X-game with $\mathcal{O}^X$.*



Task 1 – Relations among Notions (4+4 Credits)
Prove (or disprove) two of the following relations between notions from the lecture. $X \implies Y$ means that every X-secure scheme is also Y-secure.

a) RoR-CPA security $\implies$ LoR-CPA security

b) Sem-CPA security $\implies$ FtG-CPA security

c) LoR-CPA security $\implies$ FtG-CPA security

Bonus (+2): Prove all three relations. **Bonus (+1):** Provide the advantages of $\mathbf{A}^Y$ in relation of the advantage of $\mathbf{A}^X$.

Task 2 – Parity Security (4+2 Credits)
Let $(\text{Encr}_K, \text{Decr}_K)$ denote an encryption scheme. Let $\text{Parity}(X) = x_1 \oplus x_2 \oplus \cdots \oplus x_n$ be the parity function. We define Par-CPA security as follows

Definition 1 (PAR-CPA Experiment)**.** The oracle initializes a random key $K \xleftarrow{\$} \{0,1\}^k$.

1. For $1 \leq i \leq q' < q$:
   - Eve chooses $M_i \in \{0,1\}^n$ and asks the oracle for $C_i \leftarrow \text{ENCR}_K(M_i)$.
2. Eve chooses a distribution $\mathcal{M}$ of n-bit plaintexts and sends it to the oracle. The oracle chooses uniformly at random a message $M \xleftarrow{\$}_{\mathcal{M}} \{0,1\}^n$ according to $\mathcal{M}$ and responds with $C \leftarrow \text{ENCR}_K(M)$.
3. For $q' + 1 \leq i \leq q$:
   - Eve chooses a new $M_i \in \{0,1\}^n$ and asks the oracle for $C_i \leftarrow \text{ENCR}_K(M_i)$
4. Eve outputs a bit $\beta \in \{0,1\}$. She wins iff $\text{PARITY}(M) = \beta$.

a) Prove (or disprove) that SEM-CPA security $\implies$ PAR-CPA security

b) Prove (or disprove) that PAR-CPA security $\implies$ SEM-CPA security

Task 3 – CBC Padding Oracle (8 Credits)

Consider the PKCS#7 Padding scheme $MMM \ldots MN \ldots N$ where each M denotes a message byte of the last message block, and N denotes the number of bytes of the padding, where the padding is always applied, i.e., $1 \leq N \leq 16$.

a) Implement a python3 class `CBCPaddingOracle` that applies the $MMM \ldots MNN \ldots N$ padding scheme above to a message and encrypts the padded message with CBC under a user-specified IV and key. You can use the module `pycrypto` for AES-CBC.

b) Implement a function `verify_padding` in the oracle which, given a ciphertext, returns `True` if the padding after decryption was correct and `False` otherwise.

c) Implement a class `CBCPaddingOracleAttack` against the scheme above which uses the padding oracle's `verify_padding` function to successfully recover a message to a given ciphertext.

d) You can find a few test cases that your attack should pass on the website of the problem session.

e) Achieve close to optimal score for your code with `pylint`.

```python
from Crypto.Cipher import AES

class CBCPaddingOracle:
    """
    Implements a CBC padding oracle for AES-128-CBC.
    """

    def __init__(self, key: bytes):
        """
        :param key:
        """
        self.key = key

    # ----------------------------------------------------------

    def encrypt(self, initial_value: bytes, message: bytes) -> bytes:
        """
        Pads the given message to a multiple of the block length,
        computes, and returns its encryption with AES-128-CBC-PKCS#7 padding.

        :param initial_value: 16-byte initial value.
        :param message: Plaintext of arbitrary length
        :return: Ciphertext whose length is a multiple of 16 bytes, but
        always at least the length of the message.
        """
        raise Exception("to be implemented")

    # ----------------------------------------------------------

    def verify_padding(self, initial_value: bytes, ciphertext: bytes) -> bool:
        """
        Given a ciphertext, evaluates if the padding is correct.
        :param initial_value: 16-byte initial value.
        :param ciphertext: Ciphertext. Length must be multiple of 16 bytes.
        :return: True if padding is correct, and False otherwise.
        """
        raise Exception("to be implemented")
```

```python
from cbc_padding_oracle import CBCPaddingOracle

class CBCPaddingOracleAttack:

    def recover_message(self,
                        oracle: CBCPaddingOracle,
                        initial_value: bytes,
                        ciphertext: bytes) -> bytes:
        """
        Implements the CBC padding-oracle attack.
        :param initial_value: 16-byte initial value.
        :param ciphertext: Ciphertext. Length must be multiple of 16 bytes.
        :return: Returns the plaintext that corresponded to the given
                 ciphertext.
        """
        raise Exception("to be implemented")
```