CryptoGame

Jenish, Hojae, Emerson

Dec 12, 2023

CONTENTS:

CHAPTER

ONE

PLAYCRYPT

CHAPTER

TWO

GAME PACKAGE

2.1 Submodules

2.2 game.cr module

class game.cr.**CR**(*hash_function*, *key_len: int*)

Bases: object

The CR class represents a Collision Resistance (CR) game for hash functions.

Attributes:

hash_function: User defined hash function key_len (int): The length of the random key used in the collision resistance game. messages: A list to store committed messages.

Methods:

__init__(self, hash_function, key_len: int) -> None:

Initializes a new Collision Resistance game instance with the specified hash function and key length.

_initialize(self):

Private method to generate and set a random key of the specified length.

finalize(self, x1, x2):

Finalizes the collision resistance game by comparing the hash values of two provided values with the generated key.

Usage:

 # Example usage of the Collision Resistance game

def hash_function():

...

cr_instance = CR(hash_function=hash_function, key_len=16) cr_instance._initialize()

def adv(cr_instance):

... return x1, x2

result = cr_instance.finalize(x1, x2)

finalize(*x1*, *x2*)

Finalizes the collision resistance game by computing the function on the inputs and the key.

Parameters:

x1: The first value to be compared. x2: The second value to be compared.

Returns:
bool: True if the hash values are equal, False otherwise.

2.3 game.indcpa module

class game.indcpa.**INDCPA**(*enc*, *dec*, *key_len: int*, *msg_len: int*, *max_queries: int*)

Bases: object

The INDCPA class represents an Indistinguishability under Chosen-Plaintext Attack (IND-CPA) game.

Attributes:
enc: The encryption function used in the game. dec: The decryption function used in the game. key_len (int): The length of the encryption key. msg_len (int): The length of the plaintext messages. max_queries (int): The maximum number of queries allowed in the game. query_set: A list to store pairs of plaintext messages used in queries.

Methods:

__init__(self, enc, dec, key_len: int, msg_len: int, max_queries: int) -> None:
Initializes a new IND-CPA game instance with the specified encryption and decryption functions, key length, message length, and maximum allowed queries.

_initialize(self):
Private method to initialize the game by generating a random key and a random bit.

LR(self, m_0, m_1):
Simulates a game query by encrypting one of the provided plaintext messages based on a random bit.

finalize(self, guess):
Finalizes the game by checking if the guess matches the randomly chosen bit.

Usage:

 # Example usage of the IND-CPA game
b = BlockCipher(10, 5) def enc(key, m, block_cipher = b):

q = len(m) // block_cipher.block_len C_t = [] for i in range(q):

ct = block_cipher.evaluate(key, m[i*block_cipher.block_len:(i+1)*block_cipher.block_len])
C_t.append(ct)

return C_t

def dec():
pass

game = INDCPA(ecb_enc, ecb_dec, key_len=10,msg_len=5, max_queries=2) game._initialize()

def adv(game):
q1 = game.LR('00000', '00000') q2 = game.LR('00000', '10000')

if q1 == q2:
return 1

else:
return 0

w

guess = adv(game)

game.finalize(guess)

LR(*m_0*, *m_1*)

Simulates the LR oracle by encrypting one of the provided plaintext messages based on a random bit.

Parameters:
m_0: The first plaintext message. m_1: The second plaintext message.

Returns:
str: The ciphertext corresponding to the chosen plaintext message.

finalize(*guess*)

Finalizes the game by checking if the guess matches the randomly chosen bit.

Parameters:
guess: The guessed bit.

Returns:
bool: True if the guess matches the chosen bit, False otherwise.

2.4 game.int_ctxt module

class game.int_ctxt.**INTCTXT**(*enc*, *dec*, *key_len: int*, *max_queries: int*)

Bases: object

The INTCTXT class represents an Integrity under Chosen-Ciphertext Attack (INT-CTXT) game.

Attributes:
enc: The encryption function used in the game. dec: The decryption function used in the game. key_len (int): The length of the encryption key. max_queries (int): The maximum number of queries allowed in the game.

Methods:

__init__(self, enc, dec, key_len: int, max_queries: int) -> None:
Initializes a new Integrity under Chosen-Ciphertext Attack (INT-CTXT) game instance with the specified encryption and decryption functions, key length, and maximum allowed queries.

_initialize(self):
Private method to initialize the game by generating a random key and a random bit.

Enc(self, m):
Simulates a game query by encrypting the provided plaintext message and updating the query set.

finalize(self, C):
Finalizes the game by checking if the provided ciphertext is not in the query set and can be decrypted.

Usage:
For usage check intctxt.ipynb

Enc(*m*)

Simulates the Enc oracle by encrypting the provided plaintext message and updating the query set.

Parameters:
m: The plaintext message to be encrypted.

Returns:
str: The ciphertext corresponding to the provided plaintext message.

`finalize`(*C*)

Finalizes the game by checking if the provided ciphertext is not in the query set and can be decrypted.

Parameters:
C: The ciphertext to be finalized.

Returns:
bool: True if the ciphertext is not in the query set and can be decrypted, False otherwise.

2.5 game.prf module

`class` `game.prf.PRF`(*prf*, *key_len: int*, *msg_len: int*, *max_queries: int*)

Bases: `object`

The PRF class represents a Pseudorandom Function (PRF) game.

Attributes:
prf: The pseudorandom function used in the game. _random: ideal block cipher key_len (int): The length of the key used in the pseudorandom function and block cipher. msg_len (int): The length of the messages processed by the pseudorandom function and block cipher. max_queries (int): The maximum number of queries allowed in the game. query_set: A list to store messages used in queries.

Methods:

__init__(self, prf, key_len: int, msg_len: int, max_queries: int) -> None:
Initializes a new Pseudorandom Function (PRF) game instance with the specified PRF, key length, message length, and maximum allowed queries.

_initialize(self):
Private method to initialize the game by generating a random key and a random bit.

Fn(self, m):
Simulates a Fn oracle by evaluating the pseudorandom function on the provided message.

finalize(self, guess):
Finalizes the game by checking if the guess matches the randomly chosen bit.

Usage:
For usage check prf.ipynb

`Fn`(*m*)

Simulates the Fn oracle by evaluating the pseudorandom function on the provided message.

Parameters:
m: The message on which the pseudorandom function is applied.

Returns:
str: The result of applying the pseudorandom function on the message.

`finalize`(*guess*)

Finalizes the game by checking if the guess matches the randomly chosen bit.

Parameters:
guess: The guessed bit.

Returns:
bool: True if the guess matches the chosen bit, False otherwise.

2.6 game.ufcma module

class game.ufcma.**UFCMA**(*mac*, *key_len: int*, *max_queries: int*)

Bases: object

The UFCMA class represents a UFCMA game.

Attributes:

mac: The message authentication code (MAC) used in the game. key_len (int): The length of the key used in the MAC. max_queries (int): The maximum number of queries allowed in the game.

Methods:

__init__(self, mac, key_len: int, max_queries: int) -> None:

Initializes a new Universal Forgery under Chosen-Message Attack (UF-CMA) game instance with the specified MAC, key length, and maximum allowed queries.

_initialize(self):

Private method to initialize the game by generating a random key.

Tag(self, m):

Simulates a tag generation oracle by computing the MAC of the provided message.

finalize(self, message, tag):

Finalizes the game by checking if the message was not queried before and if the provided tag is valid.

Usage:

For usage check mac.ipynb

Tag(*m*)

Simulates a teg gen oracle by computing the MAC of the provided message.

Parameters:

m: The message for which the MAC is computed.

Returns:

str: The MAC of the provided message.

finalize(*message*, *tag*)

Finalizes the game by checking if the message was not queried before and if the provided tag is valid.

Parameters:

message: The message to be finalized. tag: The MAC tag to be validated.

Returns:

bool: True if the message was not queried before and the tag is valid, False otherwise.

2.7 Module contents

CHAPTER

THREE

SIMULATE MODULE

class simulate.**Simulate**(*Game: object*, *adversary*, *n_iteration=1000*)

Bases: object

The Simulate class provides methods for simulating different cryptographic games and calculating the advantage of an adversary against various security notions.

Attributes:

Game (object): The cryptographic game instance. adversary: The adversary function that interacts with the cryptographic game. n_iteration (int): The number of iterations for simulation.

Methods:

__init__(self, Game: object, adversary, n_iteration=1000) -> None:
Initializes a new Simulate instance.

simulate_INDCPA(self, verbose=False, n_iteration=1000):
Simulates the INDCPA (Indistinguishability under Chosen-Plaintext Attack) game and calculates the advantage of the adversary.

simulate_PRF(self, verbose=False, n_iteration=1000):
Simulates the PRF (Pseudorandom Function) game and calculates the advantage of the adversary.

simulate_cr(self, verbose=False, n_iteration=1000):
Simulates the CR (Collision Resistance) game and calculates the advantage of the adversary.

simulate_ufcma(self, verbose=False, n_iteration=1000):
Simulates the UFCMA game and calculates the advantage of the adversary.

simulate_intctxt(self, verbose=False, n_iteration=1000):
Simulates the INTCTXT (Integrity under Chosen-Ciphertext Attack) game and calculates the advantage of the adversary.

simulate_INDCPA(*verbose=False*, *n_iteration=1000*)

Simulates the INDCPA (Indistinguishability under Chosen-Plaintext Attack) game and calculates the advantage of the adversary.

Parameters:

verbose (bool): If True, print detailed probabilities. n_iteration (int): The number of iterations for simulation.

simulate_PRF(*verbose=False*, *n_iteration=1000*)

Simulates the PRF (Pseudorandom Function) game and calculates the advantage of the adversary.

Parameters:

verbose (bool): If True, print detailed probabilities. n_iteration (int): The number of iterations for simulation.

simulate_cr(*verbose=False*, *n_iteration=1000*)

Simulates the CR (Collision Resistance) game and calculates the advantage of the adversary.

Parameters:
verbose (bool): If True, print detailed probabilities. n_iteration (int): The number of iterations for simulation.

simulate_intctxt(*verbose=False*, *n_iteration=1000*)

Simulates the INTCTXT (Integrity under Chosen-Ciphertext Attack) game and calculates the advantage of the adversary.

Parameters:
verbose (bool): If True, print detailed probabilities. n_iteration (int): The number of iterations for simulation.

simulate_ufcma(*verbose=False*, *n_iteration=1000*)

Simulates the UFCMA game and calculates the advantage of the adversary.

Parameters:
verbose (bool): If True, print detailed probabilities. n_iteration (int): The number of iterations for simulation.

CHAPTER

FOUR

TOOLS PACKAGE

4.1 Submodules

4.2 tools.AES module

class tools.AES.**AES**(*key_len*, *block_len*)

Bases: Function

evaluate(*key*, *plaintext*)

Encrypts m with AES in ECB mode.

Parameters

- **k** – should be a binary string of length 128, 192, or 256
- **m** – should be a binary string of length multiple of 128

Returns
cipher text as binary string

inverse(*key*, *ciphertext*)

Decrypts c with AES in ECB mode.

Parameters

- **k** – should be a binary string of length 128, 192, or 256
- **c** – should be a binary string of length multiple of 128

Returns
plaintext as binary string

4.3 tools.block_cipher module

class tools.block_cipher.**BlockCipher**(*key_len*, *block_len*)

Bases: Function

evaluate(*key*, *plaintext*)

Evaluate the block cipher. Implements an ideal block cipher

Parameters:
key (str): The key used for encryption. plaintext (str): The plaintext to encrypt.

Returns:
str: The ciphertext resulting from the encryption.

Raises:
ValueError: If the key length does not match the specified key length. ValueError: If the plaintext length does not match the specified block length.

inverse(*key*, *ciphertext*)

Computing the blockcipher inverse BlockCipher.

Parameters:
key (str): The key used for decryption. ciphertext (str): The ciphertext to decrypt.

Returns:
str: The plaintext resulting from the decryption.

Raises:
ValueError: If the ciphertext length does not match the block length.

4.4 tools.functions module

class tools.functions.**Function**

Bases: object

evaluate(*K*, *m*)

inverse(*K*, *c*)

4.5 tools.modes_of_operation module

class tools.modes_of_operation.**CBC**(*function: Function*)

Bases: object

decrypt(*key*, *C*)

encrypt(*key*, *M*)

class tools.modes_of_operation.**CBC_MAC**(*function: Function*)

Bases: object

Tag_gen(*key*, *message*)

class tools.modes_of_operation.**CTR**(*function: Function*)

Bases: object

decrypt(*key*, *C*)

encrypt(*key*, *M*)

class tools.modes_of_operation.**ECB**(*function: Function*)

Bases: object

decrypt(*key*, *C*)

encrypt(*key*, *M*)

class tools.modes_of_operation.**ECBC_MAC**(*function*)

Bases: object

Tag_gen(*key*, *message*)

4.6 tools.utils module

tools.utils.**ctr_add**(*X*, *i*, *n*)

tools.utils.**generate_binary_strings**(*n*)

tools.utils.**print_table**(*data_dict*)

tools.utils.**random_bits**(*length: int*)

Generate a random binary string of the specified length.

Parameters:
length (int): The length of the binary string to generate.

Returns:
str: A random binary string of the specified length.

Raises:
ValueError: If the length is less than or equal to 0.

tools.utils.**xor**(*x1*, *x2*)

4.7 Module contents

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

R

S

T

U

X