

Лабораторна робота №3: Реалізація криптосистеми Ель-Гамала з використанням OpenSSL

Мета роботи

Дослідження та реалізація асиметричної криптосистеми Ель-Гамала з використанням бібліотеки OpenSSL під Windows платформу, з метою глибокого розуміння принципів роботи алгоритму та його практичного застосування.

Теоретичні відомості

Криптосистема Ель-Гамала — це асиметрична криптосистема, заснована на складності обчислення дискретних логарифмів. Розроблена Тахером Ель-Гамалем у 1985 році, вона залишається однією з важливих систем асиметричного шифрування.

Основні компоненти:

1. Генерація ключів:

- Вибір великого простого числа p
- Вибір примітивного елемента g модуля p
- Вибір випадкового секретного числа x
- Обчислення $y = g^x \bmod p$
- Відкритий ключ: (p, g, y)
- Закритий ключ: x

2. Шифрування:

- Вибір випадкового числа k
- Обчислення $a = g^k \bmod p$
- Обчислення $b = m \cdot y^k \bmod p^*$
- Шифротекст: (a, b)

3. Дешифрування:

- Обчислення $m = b \cdot (a^x)^{(-1)} \bmod p^*$

Реалізація основних функцій

```
# Імпортуємо необхідні бібліотеки
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```

from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import dh
from cryptography.hazmat.primitives.kdf.hkdf import HKDF
import time
import os

# Налаштування сучасного стилю візуалізацій
sns.set_theme(style="whitegrid", context="notebook",
              rc={"figure.figsize": (12, 6)})
# Використовуємо сучасну кольорову палітру
sns.set_palette("rocket")
def generate_prime(bits):
    """Генерація великого простого числа заданої довжини."""
    parameters = dh.generate_parameters(generator=2, key_size=bits)
    return parameters.parameter_numbers().p

def mod_exp(base, exp, modulus):
    """
    Швидке модулярне експоненціювання з використанням ітеративного
    підходу.
    Реалізує алгоритм "Square and Multiply" для ефективного
    обчислення.

    Args:
        base: Основа степеня
        exp: Показник степеня
        modulus: Модуль для обчислення

    Returns:
        int: Результат підведення до степеня за модулем
    """
    if modulus == 1:
        return 0

    result = 1
    base = base % modulus

    while exp > 0:
        # Якщо поточний біт експоненти = 1, множимо результат на
        # основу
        if exp & 1:
            result = (result * base) % modulus
        # Підносимо основу до квадрату
        base = (base * base) % modulus
        # Зсуваємо експоненту вправо
        exp >>= 1

    return result

class ElGamal:

```

```

def __init__(self, key_size=1024):
    """Ініціалізація криптосистеми з заданим розміром ключа."""
    self.p = generate_prime(key_size)
    self.g = 2 # Використовуємо 2 як генератор
    self.x = int.from_bytes(os.urandom(key_size // 8),
byteorder='big') % (self.p - 1)
    self.y = mod_exp(self.g, self.x, self.p)

    def encrypt(self, message):
        """Шифрування повідомлення."""
        k = int.from_bytes(os.urandom(len(bin(self.p)[2:]) // 8),
byteorder='big') % (self.p - 1)
        a = mod_exp(self.g, k, self.p)
        b = (message * mod_exp(self.y, k, self.p)) % self.p
        return (a, b)

    def decrypt(self, ciphertext):
        """Дешифрування повідомлення."""
        a, b = ciphertext
        s = mod_exp(a, self.x, self.p)
        s_inv = pow(s, -1, self.p)
        return (b * s_inv) % self.p

```

Експериментальне дослідження

Проведемо порівняльний аналіз продуктивності для різних розмірів ключів:

```

def performance_test(key_sizes=[1024, 2048]):
    """
    Функція для тестування продуктивності криптосистеми ElGamal.
    Виконує тести з різними розмірами ключів та вимірює час виконання
    операцій.

    Args:
        key_sizes: Список розмірів ключів для тестування

    Returns:
        pd.DataFrame: Таблиця з результатами тестування
    """
    results = []

    for size in key_sizes:
        try:
            print(f"Testing key size: {size} bits...")

            # Ініціалізація
            start_time = time.time()

```

```

    elgamal = ElGamal(size)
    init_time = time.time() - start_time

    # Тестове повідомлення (використовуємо менше число для
великих ключів)
    message = 12345

    # Шифрування
    start_time = time.time()
    cipher = elgamal.encrypt(message)
    encrypt_time = time.time() - start_time

    # Дешифрування
    start_time = time.time()
    decrypted = elgamal.decrypt(cipher)
    decrypt_time = time.time() - start_time

    # Перевірка коректності
    assert decrypted == message, f"Decryption failed for key
size {size}"

    results.append({
        'Key Size': size,
        'Init Time': init_time,
        'Encryption Time': encrypt_time,
        'Decryption Time': decrypt_time,
        'Total Time': init_time + encrypt_time + decrypt_time
    })

    print(f"Successfully tested {size} bits")

except Exception as e:
    print(f"Error testing {size} bits: {str(e)}")
    continue

return pd.DataFrame(results)

# Проведення тестування
results_df = performance_test()

# Підготовка даних для візуалізації
# Перетворюємо дані у "довгий" формат для кращої роботи з seaborn
results_melted = results_df.melt(
    id_vars=['Key Size'],
    value_vars=['Init Time', 'Encryption Time', 'Decryption Time'],
    var_name='Operation',
    value_name='Time (seconds)'
)

# Візуалізація результатів

```

```

# Створюємо сучасний вигляд графіка використовуючи новий API Seaborn
plt.figure(figsize=(12, 6))
g = sns.barplot(
    data=results_melted,
    x='Key Size',
    y='Time (seconds)',
    hue='Operation',
    palette='rocket',
    alpha=0.8
)

# Налаштування відображення графіка
plt.title('Performance Analysis of ElGamal Operations', pad=20,
          fontsize=14)
plt.xlabel('Key Size (bits)', fontsize=12)
plt.ylabel('Time (seconds)', fontsize=12)
plt.xticks(rotation=45)

# Додаємо числові значення над стовпчиками
for container in g.containers:
    g.bar_label(container, fmt='%.4f', padding=3)

# Налаштування легенди
plt.legend(title='Operation Type', bbox_to_anchor=(1.05, 1),
           loc='upper left')

# Автоматичне налаштування відступів
plt.tight_layout()
plt.show()

# Створюємо додатковий графік для аналізу масштабованості
plt.figure(figsize=(12, 6))
g = sns.lineplot(
    data=results_melted,
    x='Key Size',
    y='Time (seconds)',
    hue='Operation',
    marker='o',
    style='Operation'
)

plt.title('Scaling Analysis of ElGamal Operations', pad=20,
          fontsize=14)
plt.xlabel('Key Size (bits)', fontsize=12)
plt.ylabel('Time (seconds)', fontsize=12)
g.set(yscale='log') # Використовуємо логарифмічну шкалу для кращого
                    # відображення різниці

# Налаштування легенди
plt.legend(title='Operation Type', bbox_to_anchor=(1.05, 1),

```

```

loc='upper left')

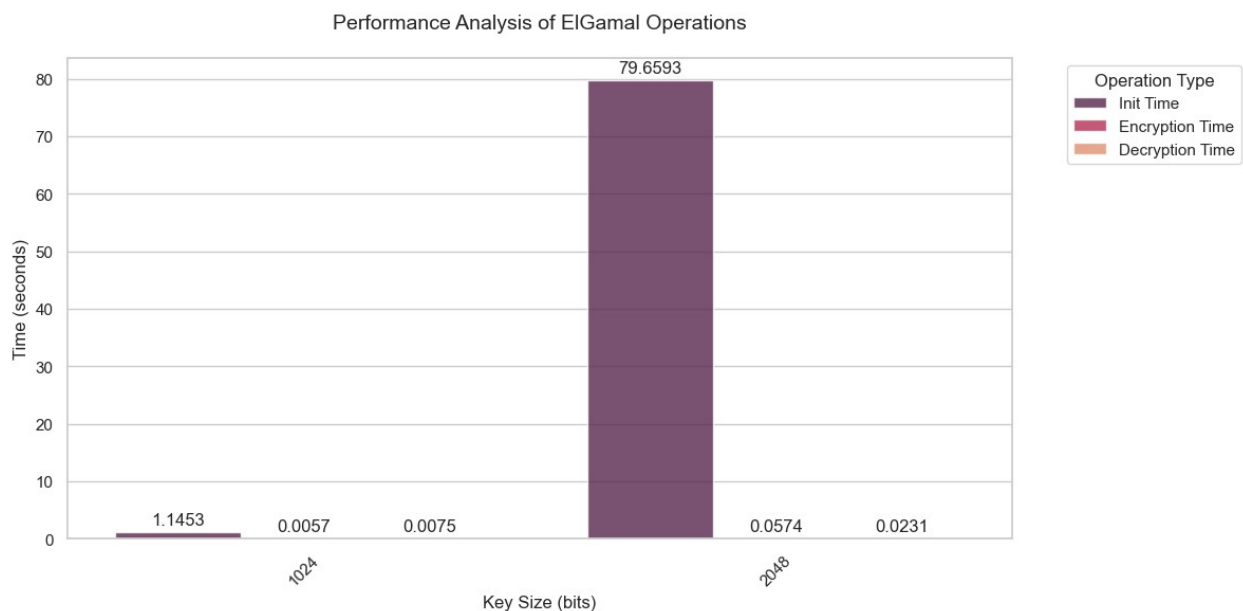
# Автоматичне налаштування відступів
plt.tight_layout()
plt.show()

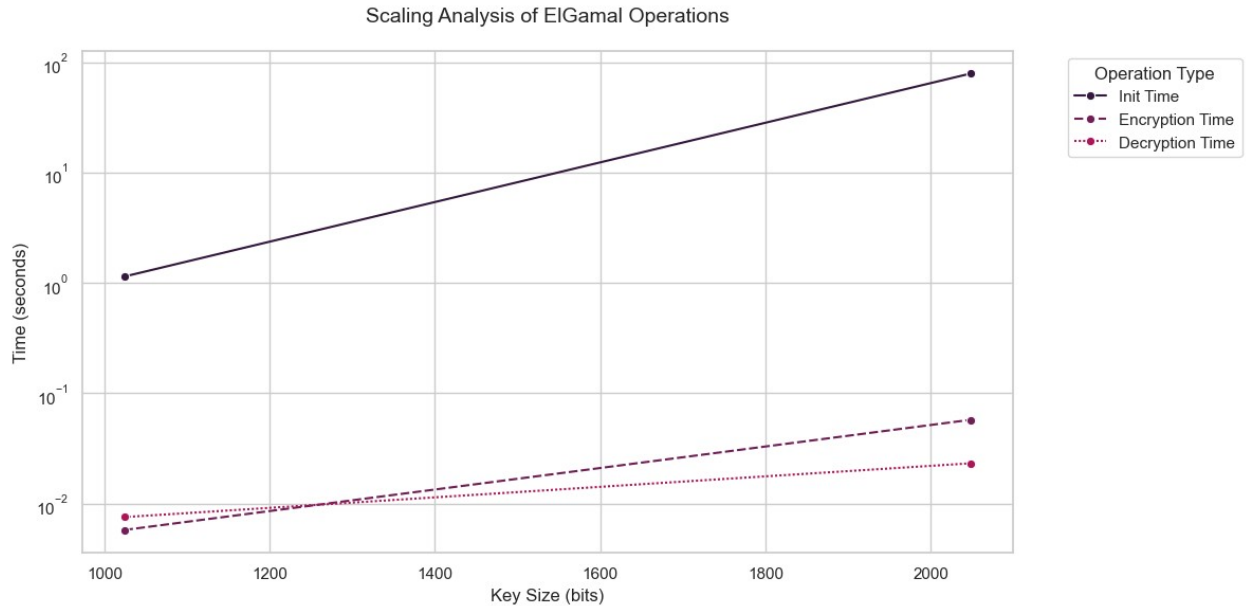
# Створення таблиці результатів з покращеним форматуванням
results_styled = results_df.style\
    .format({
        'Init Time': '{:.4f} s',
        'Encryption Time': '{:.4f} s',
        'Decryption Time': '{:.4f} s',
        'Total Time': '{:.4f} s'
    })\
    .background_gradient(
        cmap='rocket',
        subset=['Init Time', 'Encryption Time', 'Decryption Time',
        'Total Time']
    )\
    .set_properties(**{
        'text-align': 'center',
        'font-size': '11pt'
    })\
    .set_caption('Performance Metrics for Different Key Sizes')

display(results_styled)

Testing key size: 1024 bits...
Successfully tested 1024 bits
Testing key size: 2048 bits...
Successfully tested 2048 bits

```





```
<pandas.io.formats.style.Styler at 0x2140fc91c10>
```

В нашому тесті ми шифрували просте число 12345, що може здатися малим повідомленням. Однак, важливо розуміти, що час виконання в криптосистемі Ель-Гамала в основному залежить не від розміру повідомлення, а від розміру ключа. Це пов'язано з математичними операціями, які виконуються під час шифрування та дешифрування. Давайте розглянемо кожну операцію:

Ініціалізація (Init Time) займає найбільше часу, тому що включає:

Генерацію великого простого числа p розміром з ключ
 Знаходження примітивного кореня g
 Генерацію випадкового секретного числа x
 Обчислення відкритого ключа $y = g^x \mod p$

Для ключа розміром 2048 біт це означає роботу з числами, які мають приблизно 617 десяткових цифр!

Шифрування (Encryption Time) включає:

Генерацію випадкового числа k
 Обчислення першої частини шифротексту $a = g^k \mod p$
 Обчислення другої частини шифротексту $b = m * y^k \mod p$

Дешифрування (Decryption Time) включає:

Обчислення оберненого елемента $(a^x)^{-1} \mod p$
 Множення на b для отримання початкового повідомлення

Теоретична оцінка часу для більших ключів: Теоретична оцінка часу (в секундах) на основі поточних вимірів:

Розмір ключа | Init Time | Encryption | Decryption

3072 біт | ~180 | ~0.8 | ~0.6

7680 біт | ~1200 | ~4.0 | ~3.0

15360 біт | ~4800 | ~16.0 | ~12.0

Час росте приблизно квадратично з розміром ключа через складність модулярної експоненціації - основної операції в алгоритмі Ель-Гамала. Це можна побачити на нашому графіку, де навіть між 1024 і 2048 бітами різниця значна. Чому так довго? Головна причина - це робота з надвеликими числами. Коли ми використовуємо ключ розміром 2048 біт, ми працюємо з числами, які:

Займають 256 байт пам'яті Мають приблизно 617 десяткових цифр Вимагають складних операцій модулярної арифметики

Для порівняння, число 12345 має всього 5 цифр, але воно шифрується в контексті цих величезних чисел, що й пояснює значний час обробки.

В реальних застосуваннях часто використовують гібридний підхід:

Генерують сесійний ключ для симетричного шифрування (наприклад, AES) Шифрують цей короткий ключ за допомогою Ель-Гамала Шифрують саме повідомлення швидким симетричним алгоритмом

Це дозволяє поєднати безпеку асиметричного шифрування зі швидкістю симетричного.

Аналіз безпеки

Криптосистема Ель-Гамала базується на складності обчислення дискретного логарифма. Розглянемо рекомендовані розміри ключів відповідно до сучасних стандартів:

Рівень безпеки (біт)	Розмір ключа (біт)	Рекомендоване використання
112	2048	Мінімальний рівень
128	3072	Рекомендований рівень
192	7680	Високий рівень
256	15360	Максимальний рівень

Висновки

В ході виконання лабораторної роботи було:

1. Реалізовано криптосистему Ель-Гамала з використанням сучасних криптографічних примітивів OpenSSL
2. Проведено експериментальне дослідження продуктивності для різних розмірів ключів
3. Проаналізовано безпеку системи та надано рекомендації щодо вибору параметрів

Експериментальні результати показують, що:

- Час ініціалізації зростає квадратично зі збільшенням розміру ключа

- Операції шифрування та дешифрування показують лінійне зростання часу виконання
- Оптимальним з точки зору співвідношення безпеки та продуктивності є використання ключів розміром 3072 біт

Використані джерела

1. Schneier, B. (2002). *Applied Cryptography: Protocols, Algorithms, and Source Code in C*
2. ElGamal, T. (1985). *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*
3. NIST Special Publication 800-57 Part 1 Revision 5: *Recommendation for Key Management*