НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
Навчально-науковий фізико-технічний інститут
Кафедра математичних методів захисту інформації

Звіт до лабораторної №3

за темою:

Реалізація основних асиметричних криптосистем з застосуванням OpenSSL C++ на Windows платформі

Оформлення звіту:
Юрчук Олексій, ФІ-52мн

20 жовтня 2025 р.
м. Київ

ЗМІСТ

1 Introduction

Криптографія з відкритим ключем революціонізувала, вирішивши проблему розподілу ключів, від яких потерпали симетричні криптосистеми. Серед основних асиметричних криптосистем криптосистема Ель-Ґамаля, запропонована Тахером Ель-Ґамалем у 1985 році [1]. Вона є однією з найширше досліджених схем. На відміну від RSA, яка базується на проблемі розкладу цілих чисел на множники, безпека схеми Ель-Ґамаля базується на проблемі дискретного логарифму (DLP) в скінченному полі F_p^* [2].

В цій лабораторній я намагався зробити всебічний теоретичний аналіз криптосистеми ElGamal, розглянути її математичні основи, властивості безпеки та застосування як у загальних, так і в спеціалізованих криптографічних протоколах. Більшу частину уваги я приділив реалізації, що стосуються бібліотеки OpenSSL на платформі Windows, вичерпно описавши інформацію, яка є необхідною для практичного застосування.

1.1 Мета

Основні цілі мого дослідження наступні:

- Описати математичні основи криптосистеми ElGamal
- Проаналізувати припущення щодо безпеки та можливі наслідки атак
- Дослідити застосовність ElGamal у загальних криптографічних протоколах (шифрування, цифрові підписи)
- Описати теоретичну базу для реалізації на базі OpenSSL у ОС Windows
- І, власне, зробити невелику реалізацію цієї схеми у себе на комп'ютері

1.2 Історичне підґрунтя

Криптосистема ElGamal з'явилася в період інтенсивного розвитку криптографії з відкритим ключем. Після революційного прориву у вигляді роботи Діффі та Хеллмана з обміну ключами [3] та криптосистеми RSA [4], інші дослідники почали шукали альтернативні підходи з іншими основами для параметрів безпеки. Внесок Ель-Ґамаля полягав не лише в створенні схеми шифрування, а й алгоритмі цифрового підпису, який використовував складність обчислення дискретних логарифмів [1].

2 Математичне підґрунтя

2.1 Алгебраїчні структури

Означення 2.1 (Циклічна група)**.**

Група $(G, \cdot)$ називається циклічною, якщо існує $g \in G$ такий що кожен елемент з G може бути виражений через g^k для деякого цілого k. Елемент g називається генератором групи G.

Означення 2.2 (Скінченне поле)**.**

Скінченним полем $\mathbb{F}_p$ (також позначають як $GF(p)$, для простого числа p) – це множина $\{0, 1, 2, \ldots, p-1\}$ з операціями додання і множення за модулем p. Мультиплікативна група цього поля визначається як $\mathbb{F}_p^* = \mathbb{F}_p \setminus \{0\}$ та має порядок $p - 1$.

Теорема 2.1 (про кількість генераторів мультиплікативної групи поля)**.**

Для довільного простого p, мультиплікативна група $\mathbb{F}_p^*$ є циклічною та має $\varphi(p-1)$ твірних елементів, де φ – функція Ойлера (Euler's totient function) [5].

2.2 The Discrete Logarithm Problem

Означення 2.3 (Discrete Logarithm Problem (DLP))**.**

Нехай G – циклічна група порядку n з генератором g. Для заданого $h \in G$, задача DPL полягає в знаходженні цілого числа $x \in \{0, 1, \ldots, n-1\}$, такого що: $g^x = h$, або інакше кажучи: $x = \log_g h$ [2].

Обчислювальна складність розв'язання задачі DLP у правильно обраних мультиплікативних групах є основою безпеки ElGamal. Найвідоміші алгоритми для розв'язання DLP:

- **Baby-step Giant-step**: Time complexity $O(\sqrt{n})$, space complexity $O(\sqrt{n})$ [6]
- **Pollard's Rho**: Time complexity $O(\sqrt{n})$, space complexity $O(1)$ [7]
- **Index Calculus**: Subexponential time for prime fields, $L_p[1/3, c]$ where $L_p[\alpha, c] = \exp((c + o(1))(\ln p)^{\alpha}(\ln \ln p)^{1-\alpha})$ [8]
- **Number Field Sieve**: Найкращий з відомих алгоритмів для великих простих полів [9]

2.3 Обчислювальні припущення

Означення 2.4 (Обчислювальне припущення Computational Diffie-Hellman (CDH))**.**

Нехай задано g, g^a та g^b в циклічній групі G, що має порядок n. Обчислювально неможливо обчислити g^{ab} [10].

Означення 2.5 (Припущення про диференціальну стійкість Decisional Diffie-Hellman (DDH))**.**

Нехай задано g, g^a, g^b та g^c в циклічній групі G. З обчислювальної точки зору неможливо розрізнити, чи $c = ab \pmod n$ чи c є випадковим [10].

Безпека криптосистеми ElGamal базується на цих двох припущеннях, причому семантична безпека (semantic security) вимагає саме DDH.

3 Схема шифрування Ель-Ґамаля

3.1 Генерування ключів

Algorithm 1 ElGamal Key Generation

Require: Параметр безпеки λ

Ensure: Public key (p, g, h), private key x

1: Вибирають велике просте число p (зазвичай $|p| \geq 2048$ bits)
2: Вибирають генератор g з $\mathbb{F}_p^*$ (або з його великої підгрупи)
3: Вибирають випадковим чином privatev key $x \xleftarrow{\$} \mathbb{Z}_{p-1}$
4: Обчислюють відповідний йому public key $h \leftarrow g^x \bmod p$
5: **return** Public key: $pk = (p, g, h)$, Private key: $sk = x$

3.1.1 Вибір параметрів

Безпека схеми ElGamal критично залежить від правильного вибору параметрів:

- **Prime** p: Має бути щонаймешне 2048 bits для 112-bit security та 3072 bits for 128-bit security [11]

- **Generator** g: Може бути генератором як $\mathbb{F}_p^*$ так і підгрупи порядку q де $q|(p-1)$ та q є доволі великим (e.g., 256 bits)
- **Private key** x: Рівномірно розподілений на множині $\{1, 2, \ldots, p-2\}$ (або $\{1, 2, \ldots, q-1\}$ якщо підгрупа)

3.2 Шифрування

Algorithm 2 ElGamal Encryption

Require: Public key (p, g, h), повідомлення $m \in \mathbb{F}_p^*$

Ensure: Шифротекст (c_1, c_2)

1: Вибрати випадковим чином тимчасовий ключ $y \xleftarrow{\$} \mathbb{Z}_{p-1}$
2: Обчислити $c_1 \leftarrow g^y \bmod p$
3: Обчислити "shared secret" $s \leftarrow h^y \bmod p$
4: Обчислити $c_2 \leftarrow m \cdot s \bmod p$
5: **return** Шифротекст: $C = (c_1, c_2)$

3.2.1 Правильність

Правильність шифрування за схемою ElGamal випливає з властивості:

$$\begin{aligned} c_2 \cdot (c_1^x)^{-1} &= m \cdot h^y \cdot (g^y)^{-x} \bmod p \\ &= m \cdot (g^x)^y \cdot g^{-xy} \bmod p \\ &= m \cdot g^{xy} \cdot g^{-xy} \bmod p \\ &= m \bmod p \end{aligned}$$

3.3 Розшифрування

Algorithm 3 ElGamal Decryption

Require: Private key x, шифрування (c_1, c_2)

Ensure: Відкритий текст m

1: Обчислити "shared secret" $s \leftarrow c_1^x \bmod p$
2: Обчислити оберене за модулем $s^{-1} \leftarrow s^{-1} \bmod p$
3: Відновлене повідомлення $m \leftarrow c_2 \cdot s^{-1} \bmod p$
4: **return** Відкритий текст: m

3.3.1 Ефективність обчислень

Обчислювальні витрати:

- **Key Generation**: 1 піднесення до степеня за модулем
- **Encryption**: 2 піднесення до степеня за модулем, 1 модулярне множення
- **Decryption**: 1 піднесення до степеня за модулем, 1 знаходження оборотного за модулем, 1 модулярне множення

Завдяки використанню швидких алгоритмів піднесення до степеня таких як square-and-multiply, sliding window та Montgomery multiplication, ці операції є ефективними для відповідних розмірів вхідних параметрів.

3.4 Шифрування повідомлення

Практичним викликом для ElGamal є те, що повідомлення повинні бути елементами $\mathbb{F}_p^*$. Існує декілька підходів:

1. **Direct Encoding**: Для $m < p$, використовується m напряму (потребує великого p)
2. **Hybrid Encryption**: Вкористовується схема ElGamal для шифрування симетричного ключа, а далі використовується симетричне шифрування (e.g., AES) для фактичного повідомлення [12]
3. **Block Encoding**: Розбиття повідомлення на блоки, кожен з яких менше за p

4 Аналіз параметрів безпеки

4.1 Визначення безпеки

Означення 4.1 (IND-CPA Security)**.**

Криптосистема є нерозрізнимою при атаці з обраним відкритим текстом (IND-CPA), якщо жоден супротивник за поліноміальний час не може розрізнити шифрування двох обраних відкритих текстів з імовірністю, значно вищою за $1/2$ *[13].*

Теорема 4.1 (ElGamal IND-CPA Security)**.**

Схема ElGamal є безпечнішою за IND-CPA за умови виконання припущення DDH в базовій групі [14].

Ідея доведення. Зниження безпеки відбувається шляхом демонстрації того, що якщо супротивник може зламати безпеку IND-CPA ElGamal, то ми можемо побудувати алгоритм для вирішення проблеми DDH. Нехай задано DDH challenge (g, g^a, g^b, g^c), ми визначаємо public key як $h = g^a$ та створюємо шифротекст у вигляді $(g^b, m_i \cdot g^c)$ для випадково обраного $i \in \{0, 1\}$. Якщо $c = ab$, то це є дійсним шифруванням ElGamal; в іншому випадку – випадковим. Здатність супротивника розрізнити ці випадки безпосередньо перекладається на вирішення DDH [14]. □

4.2 Піддатливіть та IND-CCA2

Критичною слабкістю базового ElGamal є його **піддатливість**: маючи шифротексти (c_1, c_2), які шифрують повідомлення m, будь хто можне створити $(c_1, c_2 \cdot r)$ що розшифровуватиметься до $m \cdot r$ для довільного $r \in \mathbb{F}_p^*$.

Припущення:.

Базове шифрування ElGamal не є безпечним за стандартом IND-CCA2 (indistinguishable при адаптивній атаці з вибором шифрованого тексту)

Для досягнення безпеки IND-CCA2, шифрування ElGamal можна вдосконалити за допомогою:

- **Fujisaki-Okamoto transformation** [15]
- **OAEP padding** (Optimal Asymmetric Encryption Padding) [16]
- **Cramer-Shoup cryptosystem** (варіант за Ель-Ґамалем) [17]

4.3 Семантична безпека

Теорема 4.2 (Semantic Security)**.**

Шифрування ElGamal забезпечує семантичну безпеку за умови DDH. Зокрема, шифровані тексти не розкривають жодної інформації про відкриті тексти, крім їхньої довжини [13].

Рандомізація в шифруванні (за допомогою тимчасового ключа y) гарантує, що подвійне шифрування одного і того ж повідомлення на виході дає різні шифровані тексти. Це є важливою властивістю для семантичної безпеки.

5 Схема цифрового підпису Ель-Ґамаля

5.1 Генерація та перевірка підпису

Algorithm 4 ElGamal Signature Generation

Require: Private key x, message m, public parameters (p, g)

Ensure: Signature (r, s)

1: Вибирається випадкових тимчасовий ключ $k \xleftarrow{\$} \mathbb{Z}_{p-1}^*$, де $\gcd(k, p-1) = 1$
2: Обчислюється $r \leftarrow g^k \bmod p$
3: Обчислюється геш $h \leftarrow H(m)$, де H – криптографічна геш-функція
4: Обчислюється $s \leftarrow k^{-1}(h - xr) \bmod (p-1)$
5: **return** Signature: $\sigma = (r, s)$

Algorithm 5 ElGamal Signature Verification

Require: Public key (p, g, h), message m, signature (r, s)

Ensure: **valid** or **invalid**

1: Перевіряється чи $0 < r < p$ та $0 < s < p - 1$
2: Обчислюється геш $h \leftarrow H(m)$
3: Перевіряється умова: $g^h \stackrel{?}{\equiv} h^r \cdot r^s \pmod p$
4: **if** equation holds **then**
5: **return valid**
6: **else**
7: **return invalid**
8: **end if**

5.2 Правильність підпису

Коректність підпису випливає з умови:

$$\begin{aligned} h^r \cdot r^s &\equiv (g^x)^r \cdot (g^k)^s \pmod p \\ &\equiv g^{xr+ks} \pmod p \\ &\equiv g^{xr+k\cdot k^{-1}(h-xr)} \pmod p \\ &\equiv g^{xr+h-xr} \pmod p \\ &\equiv g^h \pmod p \end{aligned}$$

5.3 Питання щодо безпеки підписів

Критичні вимоги:

1. **Унікальність тимчасового ключа**: Щоразу різне k для різних повідомлень
2. **Секретність тимчасового ключа**: Якщо k розкрито, то private key x є скомпроментовним і може бути обчисленим як: $x \equiv r^{-1}(ks - h) \pmod{p-1}$
3. **Геш-функція**: Необхідно використовувати криптографічний хеш для запобігання екзистенційній підробці

Припущення: (Signature Key Recovery Attack)**.**

Якщо два підписи (r, s_1) та (r, s_2) генеруються з одним і тим же тимчасовим ключем k для повідомлень m_1 і m_2, приватний ключ може бути відновлений таким чином:

$$x \equiv r^{-1} \left(\frac{H(m_1) - H(m_2)}{s_1 - s_2} \bmod (p-1) \right) \pmod{p-1}$$

6 Розширені варіанти алгоритму

6.1 ElGamal в підгрупах

Для підвищення ефективності ElGamal може працювати в підгрупах $\mathbb{F}_p^*$ простого порядку q, де $q|(p-1)$.

Переваги:

- Коротші підписи та шифровані тексти (size $|q|$ замість $|p|$)
- Більш ефективний (з точки зору обчислень)
- Відповідає структурі параметрів DSA

Вибір параметрів:

- Вибрати "safe prime" $p = 2q + 1$, де p та q є простими числами
- Або використати NIST-style parameters: p в межах 2048-3072 bits, q в межах 224-256 bits

6.2 Використання еліптичних кривих зі схемою ElGamal

ElGamal природно поширюється на групи еліптичних кривих, пропонуючи значно менші розміри ключів за еквівалентного розміру безпеки [18, 19].

Algorithm 6 EC-ElGamal Encryption

Require: EC public key (E, G, Q), message point $M \in E$

Ensure: Шифротекст (C_1, C_2)

1: Випадково обирається $k \xleftarrow{\$} \mathbb{Z}_n$ де $n = \text{order}(G)$
2: Обчислюється $C_1 \leftarrow kG$
3: Обчислюється $C_2 \leftarrow M + kQ$
4: **return** (C_1, C_2)

Security Equivalence: Безпека EC-ElGamal зводиться до ECDLP (Elliptic Curve Discrete Logarithm Problem) та EC-DDH [20].

6.3 Signed ElGamal

Для досягнення безпеки IND-CCA2 signed ElGamal поєднує шифрування з автентифікацією:

1. Шифрування повідомлення: $(c_1, c_2) \leftarrow \text{ElGamal.Enc}(pk, m)$
2. Генерування підпису: $\sigma \leftarrow \text{Sign}(sk_{\text{sig}}, (c_1, c_2))$
3. Output: (c_1, c_2, σ)

Ця конструкція досягає рівня безпеки IND-CCA2 в the random oracle model [21].

7 Загальні криптографічні протоколи з використанням схеми ElGamal

7.1 Порогова криптографія (Threshold Cryptography)

Структура ElGamal природно підтримує схеми порогового дешифрування, де t з n сторін мають співпрацювати для вдалого дешифрування [22].

7.1.1 Порогове розшифрування

Setup:

1. Private key x передається за допомогою Shamir's secret sharing: $x = \sum_{i=1}^{t} x_i \lambda_i$ де λ_i — лагранжові коефіцієнти
2. Кожна i сторона володіє часткою x_i
3. Public key: $h = g^x$

Decryption:

1. За заданим шифротекстом (c_1, c_2), кожна сторона i обчислює часткове розшифрування: $d_i = c_1^{x_i}$
2. Зловмисник комбінує і відновлює повідомлення: $s = \prod_{i=1}^{t} d_i^{\lambda_i} = c_1^{\sum_{i=1}^{t} x_i \lambda_i} = c_1^x$
3. Recover message: $m = c_2 / s$

7.2 Повторне шифрування

Мультиплікативна гомоморфна властивість Ель-Ґамаля дозволяє перешифрувати повідомлення без знання його змісту(див. [23]).

Припущення: (ElGamal Re-encryption)**.**
Задано шифротекст (c_1, c_2). *Будь хто може створити нове шифрування того ж повідомлння:*

$$(c_1', c_2') = (c_1 \cdot g^r, c_2 \cdot h^r) \textit{ for random } r$$

Це шифрування також розшифровуватиметься у вихідне m.

Застосування – змішані мережі Сервери можуть перемішувати та перешифровувати голоси/повідомлення, не дізнаючись їхнього змісту, забезпечуючи анонімність користувачів [23].

7.3 Перевірка шифрування

Схема ElGamal підтримує докази з нульовим розкриттям інформації про властивості відкритого тексту без його розкриття [24].

Як приклад – шифрований текст (c_1, c_2) шифрує значення в діапазоні $[0, 2^n - 1]$ без розкриття значення.

8 Specialized Cryptographic Protocols

8.1 Гомоморфні властивості

ElGamal демонструє **Мультиплікативний гомоморфізм**:

Теорема 8.1 (Multiplicative Homomorphism)**.**
Дано два шифровані ElGamal тексти (c_1, c_2)*, що шифрують* m_1 *та* (c'_1, c'_2) *та* m_2*:*

$$(c_1 \cdot c'_1, c_2 \cdot c'_2) \textit{ is a valid encryption of } m_1 \cdot m_2$$

Застосування:

- Безпечне голосування: Множення зашифрованих голосів для підрахунку без розшифрування окремих голосів
- Безпечні аукціони: Обчислення статистики зашифрованих ставок
- Видобуток даних із збереженням конфіденційності (!)

8.2 Electronic Voting Protocols

ElGamal широко використовується в електронному голосуванні завдяки своїй гомоморфній властивості та легкості перевірки [25].

8.2.1 Базовий протокол голосування

1. **Setup**: Виборча комісія за допомогою ElGamal генерує public key (p, g, h)
2. **Voting**: Виборець кодує свій голос як $m \in \{g^0, g^1\}$ – ні/так, шифруючи: $(c_1, c_2) = (g^r, m \cdot h^r)$
3. **Proof**: Виборець надає доказ з нульовим розкриттям інформації, що $m \in \{1, g\}$
4. **Tallying**: Всі шифротексти між собою: $(C_1, C_2) = (\prod c_{1,i}, \prod c_{2,i})$
5. **Result**: Розшифрування (C_1, C_2) для отримання g^{total}, обчислюється дискретний логарифм, для обрахунку total value.

8.3 Secure Multi-Party Computation

ElGamal слугує ”цеглинкою” для протоколу безпеки multi-party computation (MPC) [26].

Приклад протоколу – проблема мільйонерів: Дві сторони хочуть визначити, хто з них багатший, не розкриваючи свого фактичного статку:

1. Alice має багатство w_A, Bob має багатство w_B
2. Alice генерує ElGamal keypair, та публікує pk

3. Alice шифрує: $E(w_A)$
4. Bob обчислює: $E(w_A - w_B) = E(w_A) \cdot E(-w_B)$ використовуючи гомоморфізм
5. Використовуючи додаткові криптографічні методи, вони визначають sign без дешифрування

8.4 ”Справедлива” криптографія

ElGamal можна модифікувати для support verifiable escrow, де довірена третя сторона може провести розшифрування за певних умов [27].

Перевірка шифрування для третьої сторони:

1. Користувач шифрує повідомлення ключем одержувача: (c_1, c_2)
2. Користувач також шифрує тимчасовий ключ y за допомогою escrow agent’s key: (c_1', c_2')
3. Користувач надає доказ з нульовим розкриттям інформації, що обидва шифровані тексти використовують однаковий y
4. Escrow agent може відновити y, а потім і розшифрувати (c_1, c_2), якщо має на те повноваження

9 Implementation with OpenSSL on Windows

9.1 Архітектура OpenSSL для ElGamal

OpenSSL не надає вбудованих функцій шифрування ElGamal, але пропонує певні будівельні блоки для цього:

- **BIGNUM library**: Арифметика довільної точності
- **BN_mod_exp**: Модулярне піднесення до степеня (що є критично важливим для ElGamal)
- **BN_mod_inverse**: Знаходження оберненого за модулем (для розшифрування)
- **DH (Diffie-Hellman)**: Схема Діффі-Хеллмана має спільну структуру параметрів з ElGamal
- **DSA**: Варіант для підпису ElGamal

9.2 Використання інфраструктури DSA

OpenSSL’s DSA implementation може бути адаптована для підпису Ель-Ґамаля:

```
#include <openssl/dsa.h>
#include <openssl/bn.h>

// Generate DSA parameters (compatible with ElGamal)
DSA *dsa = DSA_new();
DSA_generate_parameters_ex(dsa, 2048, NULL, 0, NULL, NULL, NULL);

// Generate key pair
DSA_generate_key(dsa);

// Access parameters
const BIGNUM *p, *q, *g, *pub_key, *priv_key;
DSA_get0_pqg(dsa, &p, &q, &g);
DSA_get0_key(dsa, &pub_key, &priv_key);
```

9.3 Implementing ElGamal Encryption

```c
#include <openssl/bn.h>
#include <openssl/rand.h>

typedef struct {
    BIGNUM *p;      // Prime modulus
    BIGNUM *g;      // Generator
    BIGNUM *h;      // Public key (g^x mod p)
    BIGNUM *x;      // Private key (only for decryption)
} ElGamal_Key;

typedef struct {
    BIGNUM *c1;     // g^y mod p
    BIGNUM *c2;     // m * h^y mod p
} ElGamal_Ciphertext;

int elgamal_encrypt(ElGamal_Key *pubkey, BIGNUM *message,
                    ElGamal_Ciphertext *ciphertext, BN_CTX *ctx) {
    BIGNUM *y = BN_new();
    BIGNUM *shared_secret = BN_new();

    // Generate random ephemeral key y
    BN_rand_range(y, pubkey->p);

    // Compute c1 = g^y mod p
    BN_mod_exp(ciphertext->c1, pubkey->g, y, pubkey->p, ctx);

    // Compute shared secret = h^y mod p
    BN_mod_exp(shared_secret, pubkey->h, y, pubkey->p, ctx);

    // Compute c2 = m * shared_secret mod p
    BN_mod_mul(ciphertext->c2, message, shared_secret,
               pubkey->p, ctx);

    // Cleanup sensitive data
    BN_clear_free(y);
    BN_clear_free(shared_secret);

    return 1;
}

int elgamal_decrypt(ElGamal_Key *privkey,
                    ElGamal_Ciphertext *ciphertext,
                    BIGNUM *message, BN_CTX *ctx) {
    BIGNUM *shared_secret = BN_new();
    BIGNUM *s_inv = BN_new();

    // Compute shared secret = c1^x mod p
    BN_mod_exp(shared_secret, ciphertext->c1, privkey->x,
               privkey->p, ctx);
```

```
    // Compute inverse of shared secret
    BN_mod_inverse(s_inv, shared_secret, privkey->p, ctx);

    // Recover message: m = c2 * s^(-1) mod p
    BN_mod_mul(message, ciphertext->c2, s_inv, privkey->p, ctx);

    // Cleanup
    BN_clear_free(shared_secret);
    BN_clear_free(s_inv);

    return 1;
}
```

9.4 Ефективне модулярне піднесення до степеня

OpenSSL надає оптимізовані реалізації цієї операції з використанням:

- **Montgomery multiplication**: Ефективна модульна арифметика
- **Windowing methods**: Зменшення кількості множень
- **Chinese Remainder Theorem**: При деяких параметрів

```
// Using BN_CTX for temporary variables (thread-safe)
BN_CTX *ctx = BN_CTX_new();
BN_CTX_start(ctx);

BIGNUM *result = BN_CTX_get(ctx);

// Efficient modular exponentiation
// Uses sliding window method automatically
BN_mod_exp(result, base, exponent, modulus, ctx);

BN_CTX_end(ctx);
BN_CTX_free(ctx);
```

9.5 Особливості Windows платформи

9.5.1 Джерело ентропії

На Windows, OpenSSL використовує:

- `BCryptGenRandom` (Windows 10+) або `CryptGenRandom` (legacy)
- Інструкції для CPU: RDRAND/RDSEED
- Лічильники продуктивності системи

9.5.2 Compilation and Linking

```
# Using MinGW-w64
gcc -o elgamal elgamal.c -I/c/OpenSSL/include ^
    -L/c/OpenSSL/lib -lcrypto -lws2_32
```

9.6 Управління пам'яттю та безпека

```
// Secure memory allocation for sensitive data
BIGNUM *private_key = BN_secure_new();

// Clear sensitive data before freeing
BN_clear_free(private_key);

// For regular BIGNUMs
BN_free(public_key);

// Securely wipe memory buffer
OPENSSL_cleanse(buffer, buffer_size);
```

9.7 Обробка помилок

```
#include <openssl/err.h>

int elgamal_operation() {
    if (!BN_mod_exp(result, base, exp, mod, ctx)) {
        // Print OpenSSL error stack
        unsigned long err = ERR_get_error();
        char err_buf[256];
        ERR_error_string_n(err, err_buf, sizeof(err_buf));
        fprintf(stderr, "OpenSSL error: %s\n", err_buf);
        return 0;
    }
    return 1;
}
```

10 Аналіз продуктивності

10.1 Обчислювальна складність

Operation	Modular Exp.	Other Operations
Key Generation	1	1 multiplication
Encryption	2	1 multiplication
Decryption	1	1 inversion, 1 multiplication
Sign Generation	1	1 inversion, 2 multiplication, 1 hash
Sign Verification	2	1 multiplication, 1 hash

Таблиця 1: Computational Costs of ElGamal

Для n-bit модуля модульне піднесення до степеня вимагає $O(n^3)$ bit бітових операцій із використанням стандартних алгоритмів або $O(n^2 \log n)$ з використанням методів на основі FFT [2].

10.2 Порівняння з RSA

Operation	ElGamal	RSA	Ratio
Key Generation	Moderate	Slow	ElGamal 5-10× faster
Encryption	Slow	Fast	RSA 10-100× faster
Decryption	Moderate	Slow	Similar
Ciphertext Size	$2\times$ size	$1\times$ size	ElGamal 2× larger

Таблиця 2: ElGamal vs RSA Performance

Висновки:

- Шифрування в RSA є швидшим (малий відкритий показник $e = 65537$)
- Генерація ключів в ElGamal є швидшою (немає вимоги до розкладу на прості множники)
- Шифротексти за допомогою ElGamal удвічі більші (оскільки використано два елементи групи)
- ElGamal забезпечує семантичну безпеку без заповнення (на відміну від класичного RSA)

10.3 Порівняння з ElGamal Elliptic Curve

Security (bits)	ElGamal $\lvert p\rvert$	EC-ElGamal	Ratio
80	1024	160	6.4:1
112	2048	224	9.1:1
128	3072	256	12:1
192	7680	384	20:1
256	15360	512	30:1

Таблиця 3: Security Levels: ElGamal vs EC-ElGamal

EC-ElGamal забезпечує значне зменшення розміру ключа, що призводить до прискорення операцій і зменшення вимог до пропускної здатності мережі [20].

11 Security Recommendations

11.1 Рекомендації щодо вибору параметрів ElGamal

1. **Modulus Size**: Мінімум 2048 bits для поточної безпеки (2025 рік), 3072 bits біт для довгострокового захисту (2030+) [11]
2. **Safe Primes**: Використання $p = 2q + 1$ де числа p та q – прості числа, що забезпечується великим порядком підгрупи
3. **Generator Selection**:
   - Для цілої групи $\mathbb{F}_p^*$: довільний генератор підходить
   - Для підгупи: перевірка, чи $g^q \equiv 1 \pmod{p}$
4. **Random Number Generation**: Використання криптографічно захищеного PRNG (OpenSSL's `RAND_bytes`)
5. **Ephemeral Key Management**:

- Генерування щоразу нових k або y для кожної операції
- Видалення одразу після використання
- Ніколи не використовуйти повторно в різних повідомленнях

11.2 Implementation Security

11.2.1 Запобігання атакам через бічні канали

- **Timing Attacks**: Використання реалізації з постійним часом

```
// OpenSSL's BN_mod_exp uses constant-time methods
// when BN_FLG_CONSTTIME is set
BN_set_flags(private_key, BN_FLG_CONSTTIME);
BN_mod_exp(result, base, private_key, modulus, ctx);
```

- **Power Analysis**: Implement blinding techniques
- **Cache-Timing**: Use scatter-gather table lookups

11.2.2 Безпека пам’яті

```
// Always check return values
if (!BN_rand_range(ephemeral_key, modulus)) {
    // Handle error - do not proceed
    return ERROR_CODE;
}

// Secure cleanup
BN_clear_free(ephemeral_key);
OPENSSL_cleanse(buffer, sizeof(buffer));
```

11.3 Безпека на рівні протоколу

1. **Key Freshness**: Регулярна зміна ключів
2. **Perfect Forward Secrecy**: Використання тимчасових ключів для кожного нового сеансу
3. **Authentication**: Поєднання шифрування з підписами або MAC
4. **Padding**: Використання structured padding для CCA2 безпеки
5. **Domain Separation**: Використання різних ключів для різних цілей

12 Практичне застосування та приклади з реального життя

12.1 PGP and GNU Privacy Guard

Хоча PGP в основному використовує RSA та DSA, деякі його реалізації підтримують і ElGamal:

- **Encryption**: ElGamal для інкапсуляції ключів
- **Signatures**: DSA (варіант ElGamal)
- **Hybrid Approach**: ElGamal шифрує симетричний ключ, а AES шифрує власне повідомлення

12.2 Застосування у криптовалютних активах

Декілька протоколів криптовалюти використовують конструкції на основі схеми Ель-Ґамаля:

12.2.1 Pedersen Commitments

ElGamal використовується в Monero та інших монетах, що забезпечують конфіденційність:

$$C(m, r) = g^m h^r$$

Обчислювальне приховування та ідеальне зв'язування за DLP [28].

12.2.2 Конфіденційні транзакції

Розширення гомоморфізму Ель-Ґамаля для приховування сум транзакцій при забезпеченні підтвердження їх дійсності [29].

12.3 Безпечні системи голосування

Helios Voting System: Електронне голосування з відкритим кодом з використанням ElGamal [30]

- Гомоморфне підрахунку голосів без розшифрування окремих голосів
- Доступна публічна перевірка за допомогою доказів з нульовим розкриттям інформації
- Використовується в університетських виборах, організаційних голосуваннях

12.4 Хмарна безпека

Searchable Encryption:ElGamal дозволяє здійснювати пошук зашифрованих даних::

- Шифрування ключових слів за допомогою ElGamal
- Використовуйте гомоморфні властивості для перевірки рівності
- Збереження конфіденційності при увімкненні хмарного пошуку [31]

13 Advanced Usage

13.1 Zero-Knowledge Proofs for ElGamal

Prove knowledge of m у шифротексті $(c_1, c_2) = (g^y, mh^y)$ без розкриття m або y:

Algorithm 7 Schnorr-like ZK Proof для знання відкритого тексту ElGamal

1: **Prover** chooses random $r \xleftarrow{\$} \mathbb{Z}_p$
2: **Prover** sends commitment: $t = g^r$
3: **Verifier** sends challenge: $c \xleftarrow{\$} \mathbb{Z}_p$
4: **Prover** computes response: $s = r + cy \bmod (p-1)$
5: **Verifier** checks: $g^s \stackrel{?}{=} t \cdot c_1^c$

13.2 Шифрування для декількох одержувачів

ElGamal може бути розширений для ефективного шифрування трансляції, де один шифрований текст розшифровується в одне і те ж повідомлення для декількох одержувачів [32].

Побудова:

1. Одержувачі мають public keys $h_1 = g^{x_1}, \ldots, h_n = g^{x_n}$
2. Відправник генерує єдиний ephemeral key y
3. Шифротекст: $(c_1, c_{2,1}, \ldots, c_{2,n}) = (g^y, mh_1^y, \ldots, mh_n^y)$
4. Кожен одержувач i розшифровує: $m = c_{2,i}/c_1^{x_i}$

13.3 Identity-Based ElGamal

Розширення шифрування на основі ідентичності з використанням білінійних пар:

- Відкритий ключ користувача, отриманий з ідентифікаційного рядка
- Немає потреби в сертифікатах відкритих ключів
- Приватний ключ, виданий довіреним органом

14 Порівняння ElGamal з іншими криптосистемами

14.1 Порівняння функцій

Feature	RSA	ElGamal	ECC	Paillier
Encryption	Yes	Yes	Yes	Yes
Signatures	Yes	Yes	Yes	No
Homomorphic	Partially	Multiplicative	No	Additive
Semantic Security	No*	Yes	Yes	Yes
Ciphertext Size	$1\times$	$2\times$	$2\times$	$1\times$
Key Gen Speed	Slow	Fast	Fast	Slow
Encryption Speed	Fast	Moderate	Moderate	Slow
Decryption Speed	Slow	Moderate	Moderate	Slow
Key Size (128-bit sec)	3072	3072	256	3072
Standardization	High	Moderate	High	Low
Patent Issues	No	No	Some	Some

* Вимагає заповнення (OAEP) для семантичної безпеки

Таблиця 4: Порівняння асиметричних криптосистем

14.2 Порівняння основ безпеки

- **RSA**: Integer factorization problem
- **ElGamal**: Discrete logarithm problem (DLP)
- **ECC**: Elliptic curve discrete logarithm problem (ECDLP)
- **Paillier**: Decisional composite residuosity assumption

Всі вони вважаються вразливими до квантових обчислень; постквантові альтернативи включають схеми на основі решіток (NTRU, Kyber) та кодів (McEliece) [33].

15 Майбутні напрямки для досліджень

15.1 Думки про Post-Quantum світ

Shor's algorithm [34] вирішує DLP за поліноміальний час на квантових комп'ютерах:

- ElGamal є вразливим до достатньо великих квантових комп'ютерів
- Приблизний час 2040-ті – для квантових комп'ютерів, що мають значення для криптографії
- Необхідні стратегії міграції до постквантових альтернатив

Квантово-стійкі альтернативи:

- **На основі решітки**: NTRU, Kyber (стандарт NIST)
- **На основі коду**: McEliece
- **Multivariate**: Rainbow
- **На основі гешу**: SPHINCS+

15.2 Розширення протоколу

Активні напрямки досліджень:

- **Функціональне шифрування**: функції дешифрування відкритих текстів
- **Повторне шифрування проксі**: делегування прав на дешифрування
- **Обчислення з перевіркою**: доведення правильності зашифрованих обчислень

16 Implementation Roadmap for OpenSSL

16.1 Етап 1: Базова реалізація

Цілі:

1. Впровадити генерацію ключів ElGamal за допомогою OpenSSL BIGNUM
2. Впровадити функції шифрування/дешифрування
3. Впровадити генерації/верифікації підпису
4. Створення комплексного набору тестів

Ключові компоненти:

```
// Data structures
typedef struct elgamal_key_st ELGAMAL_KEY;
typedef struct elgamal_ctx_st ELGAMAL_CTX;

// API functions
ELGAMAL_KEY* ElGamal_new(void);
void ElGamal_free(ELGAMAL_KEY *key);
int ElGamal_generate_key(ELGAMAL_KEY *key, int bits);
int ElGamal_encrypt(ELGAMAL_CTX *ctx, const BIGNUM *plaintext,
                    BIGNUM *c1, BIGNUM *c2);
int ElGamal_decrypt(ELGAMAL_CTX *ctx, const BIGNUM *c1,
                    const BIGNUM *c2, BIGNUM *plaintext);
```

16.2 Етап 2: Інтеграція протоколу

Цілі:

1. Впровадити гібридне шифрування (ElGamal + AES)
2. Додати підтримку zero-knowledge proof
3. Впровадити threshold cryptography
4. Створити демонстраційні приклади для протоколу

16.3 Етап 3: Оптимізація

Цілі:

1. Впровадити constant-time operations
2. Додати засоби захисту від атак через побічні канали
3. Тестування та налагодження продуктивності
4. Аудит безпеки та формальна верифікація

16.4 Додаткові settings (Development Tools)

- **IDE**: Visual Studio 2022, CLion
- **Build System**: CMake for cross-platform compatibility
- **Testing**: Google Test, OpenSSL test framework
- **Profiling**: VTune, Windows Performance Analyzer
- **Memory Checking**: Dr. Memory, Application Verifier

17 Висновки

Криптосистема ElGamal є фундаментальним внеском у криптографію з відкритим ключем, пропонуючи унікальні властивості, що відрізняють її від інших асиметричних схем. Вона базується на проблемі дискретного логарифму яка власне забезпечує гарантії безпеки, що відрізняються від систем на основі факторизації, таких як RSA, сприяючи криптографічній різноманітності.

17.1 Ключові переваги

1. **Семантична безпека:** ймовірнісне шифрування забезпечує семантичну безпеку за умови DDH
2. **Гомоморфні властивості:** мультиплікативний гомоморфізм дозволяє створювати протоколи, що зберігають конфіденційність
3. **Простота:** концептуально елегантна з простими доказами безпеки
4. **Гнучкість:** легко адаптується до різних алгебраїчних структур (підгрупи, еліптичні криві)
5. **Сумісність з різними протоколами:** природньо підходить для порогової криптографії, голосування

17.2 Практичне застосування

1. **Розширення шифрованого тексту:** $\times 2$ більший обсяг порівняно з RSA
2. **Пластичність:** для деяких застосувань потрібні додаткові заходи (підписи, CCA2-заповнення)
3. **Продуктивність:** повільніше шифрування, ніж RSA, але швидше генерування ключів
4. **Стандартизація:** менш стандартизований, ніж RSA/ECC у основних протокола

17.3 Значення для майбутніх досліджень

Важливість ElGamal's виходить за межі його безпосереднього використання:

- Основа для численних прогресивних протоколів (голосування, MPC, гомоморфне шифрування)
- Теоретичний інструмент для освоєння public-key cryptography
- Еталон для порівняння нових криптографічних схем
- Освітня цінність у демонстрації основних криптографічних концепцій

References

[1] Taher ElGamal. «A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms». In: *IEEE Transactions on Information Theory* 31.4 (1985), pp. 469–472. DOI: 10.1109/TIT.1985.1057074.

[2] Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996. ISBN: 978-0849385230. URL: http://cacr.uwaterloo.ca/hac/.

[3] Whitfield Diffie and Martin E. Hellman. «New Directions in Cryptography». In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654. DOI: 10.1109/TIT.1976.1055638.

[4] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. «A Method for Obtaining Digital Signatures and Public-Key Cryptosystems». In: *Communications of the ACM* 21.2 (1978), pp. 120–126. DOI: 10.1145/359340.359342.

[5] Kenneth H. Rosen. *Elementary Number Theory*. 6th. Pearson, 2011. ISBN: 978-0321500311.

[6] Daniel Shanks. «Class Number, a Theory of Factorization, and Genera». In: *Proceedings of Symposia in Pure Mathematics* 20 (1971), pp. 415–440.

[7] John M. Pollard. «Monte Carlo Methods for Index Computation $(\mathrm{mod}\ p)$». In: *Mathematics of Computation* 32.143 (1978), pp. 918–924. DOI: 10.2307/2006496.

[8] Daniel M. Gordon. «Discrete Logarithms in $GF(p)$ Using the Number Field Sieve». In: *SIAM Journal on Discrete Mathematics*. Vol. 6. 1. SIAM, 1993, pp. 124–138. DOI: 10.1137/0406010.

[9] Oliver Schirokauer. «Discrete Logarithms and Local Units». In: vol. 345. Royal Society, 1993, pp. 409–423. DOI: 10.1098/rsta.1993.0139.

[10] Dan Boneh. «The Decision Diffie-Hellman Problem». In: *Algorithmic Number Theory: Third International Symposium, ANTS-III*. Springer, 1998, pp. 48–63. DOI: 10.1007/BFb0054851.

[11] Elaine Barker. *Recommendation for Key Management: Part 1 – General*. Tech. rep. NIST Special Publication 800-57 Part 1 Revision 5. Gaithersburg, MD: National Institute of Standards and Technology, 2020. DOI: 10.6028/NIST.SP.800-57pt1r5.

[12] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. *The Oracle Diffie-Hellman Assumptions and an Analysis of DHIES*. Cryptology ePrint Archive, Paper 2001/108. 2001. URL: https://eprint.iacr.org/2001/108.

[13] Shafi Goldwasser and Silvio Micali. «Probabilistic Encryption». In: vol. 28. 2. Elsevier, 1984, pp. 270–299. DOI: 10.1016/0022-0000(84)90070-9.

[14] Yiannis Tsiounis and Moti Yung. «On the Security of ElGamal Based Encryption». In: *Public Key Cryptography: First International Workshop on Practice and Theory in Public Key Cryptography, PKC'98*. Springer, 1998, pp. 117–134. DOI: 10.1007/BFb0054019.

[15] Eiichiro Fujisaki and Tatsuaki Okamoto. «Secure Integration of Asymmetric and Symmetric Encryption Schemes». In: *Advances in Cryptology – CRYPTO '99*. Springer, 1999, pp. 537–554. DOI: 10.1007/3-540-48405-1_34.

[16] Mihir Bellare and Phillip Rogaway. «Optimal Asymmetric Encryption». In: *Advances in Cryptology – EUROCRYPT '94*. Springer, 1994, pp. 92–111. DOI: 10.1007/BFb0053428.

[17] Ronald Cramer and Victor Shoup. «A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack». In: *Advances in Cryptology – CRYPTO '98*. Springer, 1998, pp. 13–25. DOI: 10.1007/BFb0055717.

[18] Neal Koblitz. «Elliptic Curve Cryptosystems». In: *Mathematics of Computation* 48.177 (1987), pp. 203–209. DOI: 10.2307/2007884.

[19] Victor S. Miller. «Use of Elliptic Curves in Cryptography». In: *Advances in Cryptology – CRYPTO '85 Proceedings*. Springer, 1986, pp. 417–426. DOI: 10.1007/3-540-39799-X_31.

[20] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 2006. ISBN: 978-0387952734.

[21] Mihir Bellare and Phillip Rogaway. «Random Oracles are Practical: A Paradigm for Designing Efficient Protocols». In: *1st ACM Conference on Computer and Communications Security*. ACM, 1993, pp. 62–73. DOI: 10.1145/168588.168596.

[22] Yvo Desmedt and Yair Frankel. «Threshold Cryptosystems». In: *Advances in Cryptology – CRYPTO '89*. Springer, 1990, pp. 307–315. DOI: 10.1007/0-387-34805-0_28.

[23] David L. Chaum. «Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms». In: *Communications of the ACM* 24.2 (1981), pp. 84–90. DOI: 10.1145/358549.358563.

[24] Jan Camenisch and Victor Shoup. «Practical Verifiable Encryption and Decryption of Discrete Logarithms». In: *Advances in Cryptology – CRYPTO 2003*. Springer, 2003, pp. 126–144. DOI: 10.1007/978-3-540-45146-4_8.

[25] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. «A Secure and Optimally Efficient Multi-Authority Election Scheme». In: *Advances in Cryptology – EUROCRYPT '97*. Springer, 1997, pp. 103–118. DOI: 10.1007/3-540-69053-0_9.

[26] Oded Goldreich. *Secure Multi-Party Computation*. Cambridge University Press, 1998. URL: http://www.wisdom.weizmann.ac.il/~oded/PSBookFrag/part4N.ps.

[27] Silvio Micali. «Fair Public-Key Cryptosystems». In: *Advances in Cryptology – CRYPTO '92*. Springer, 1993, pp. 113–138. DOI: 10.1007/3-540-48071-4_9.

[28] Torben Pryds Pedersen. «Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing». In: *Advances in Cryptology – CRYPTO '91*. Springer, 1991, pp. 129–140. DOI: 10.1007/3-540-46766-1_9.

[29] Greg Maxwell. *Confidential Transactions*. Technical Report. 2016. URL: https://people.xiph.org/~greg/confidential_values.txt.

[30] Ben Adida. «Helios: Web-based Open-Audit Voting». In: *17th USENIX Security Symposium*. USENIX Association, 2008, pp. 335–348.

[31] Dan Boneh et al. «Public Key Encryption with Keyword Search». In: *Advances in Cryptology – EUROCRYPT 2004*. Springer, 2004, pp. 506–522. DOI: 10.1007/978-3-540-24676-3_30.

[32] Mihir Bellare, Alexandra Boldyreva, and Jessica Staddon. «Multi-Recipient Encryption Schemes: How to Save on Bandwidth and Computation Without Sacrificing Security». In: *IEEE Symposium on Security and Privacy*. IEEE, 2000, pp. 97–108. DOI: 10.1109/SECPRI.2000.848450.

[33] Daniel J. Bernstein. «Introduction to Post-Quantum Cryptography». In: *Post-Quantum Cryptography*. Springer, 2009, pp. 1–14. DOI: 10.1007/978-3-540-88702-7_1.

[34] Peter W. Shor. «Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer». In: vol. 26. 5. SIAM, 1997. DOI: 10.1137/S0097539795293172.

A Умовні позначення

Notation	Meaning
$\mathbb{Z}_n$	Integers modulo n: $\{0, 1, \ldots, n-1\}$
$\mathbb{Z}_n^*$	Multiplicative group modulo n (coprime to n)
$\mathbb{F}_p$	Finite field with p elements (prime p)
$\mathbb{F}_p^*$	Multiplicative group of $\mathbb{F}_p$
$g^x \bmod p$	Modular exponentiation
$a \equiv b \pmod{n}$	a is congruent to b modulo n
$\log_g h$	Discrete logarithm of h to base g
$x \xleftarrow{\$} S$	x chosen uniformly at random from set S
$\gcd(a, b)$	Greatest common divisor of a and b
$\varphi(n)$	Euler's totient function
$\lvert x \rvert$	Bit length of x
$H(\cdot)$	Cryptographic hash function
$\stackrel{?}{=}$	Equality check/verification

Table 5: Умовні позначення

B Sample OpenSSL Code: Complete ElGamal Implementation

```c
// elgamal_complete.h
#ifndef ELGAMAL_H
#define ELGAMAL_H

#include <openssl/bn.h>

typedef struct {
    BIGNUM *p;
    BIGNUM *g;
    BIGNUM *h;      // Public key
    BIGNUM *x;      // Private key (NULL for public-only)
} ElGamal_Key;

// Key management
ElGamal_Key* ElGamal_Key_new(void);
void ElGamal_Key_free(ElGamal_Key *key);
int ElGamal_generate_parameters(ElGamal_Key *key, int bits);
int ElGamal_generate_key(ElGamal_Key *key);

// Encryption/Decryption
int ElGamal_encrypt(const ElGamal_Key *pubkey,
                    const BIGNUM *message,
                    BIGNUM *c1, BIGNUM *c2);
int ElGamal_decrypt(const ElGamal_Key *privkey,
                    const BIGNUM *c1, const BIGNUM *c2,
                    BIGNUM *message);
```

```
27
28 // Utility functions
29 int ElGamal_verify_parameters(const ElGamal_Key *key);
30 void ElGamal_print_key(const ElGamal_Key *key);
31
32 #endif // ELGAMAL_H
```

C Основні статті

- ElGamal, T. (1985). ”A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms”
- Diffie, W., Hellman, M. (1976). ”New Directions in Cryptography”
- Goldwasser, S., Micali, S. (1984). ”Probabilistic Encryption”
- Menezes, van Oorschot, Vanstone: ”Handbook of Applied Cryptography”
- Katz, Lindell: ”Introduction to Modern Cryptography”
- Stinson, Paterson: ”Cryptography: Theory and Practice”

D Гайди для реалізації схеми ElGamal

- OpenSSL Documentation: https://www.openssl.org/docs/
- ”Network Security with OpenSSL” by Viega, Messier, Chandra
- ”Implementing Cryptography Using Python” by Shannon Bray