КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра математичної
інформатики

Дисципліна «Проблеми кодування та захисту інформації»

ЗВІТ

з виконання лабораторної роботи №2

на тему:

“Atomic Swap”

Виконав аспірант

Волохович Ігор

ЗМІСТ

1. Мета роботи 2
2.1. Поняття атомарного обміну 2
2.2 Хешований контракт з часовим замком (HTLC) 3
3. Реалізація 3
Реалізація протоколу атомарного обміну 8
Крок 1: Ініціатор генерує секрет і хеш 8
Крок 2: Ініціатор створює транзакції 8
Крок 3: Контрагент підписує транзакцію повернення 9
Крок 4: Ініціатор відправляє TX1 в блокчейн 9
Крок 5: Контрагент створює свої транзакції 9
Крок 6: Ініціатор підписує повернення для контрагента 9
Крок 7: Контрагент відправляє TX3 в блокчейн 10
Крок 8: Ініціатор отримує ALT-монети, розкриваючи секрет 10
Крок 9: Контрагент отримує BTC, використовуючи розкритий секрет 10
Реалізація користувацького інтерфейсу 10
Тестування 12
Функції безпеки 13
Висновок 13

1. Мета роботи

Метою даної лабораторної роботи є реалізація протоколу атомарного обміну для міжланцюгових обмінів криптовалют. Атомарні обміни дозволяють двом сторонам обмінюватись різними криптовалютами безпосередньо, безпечно та без необхідності довіряти один одному, без використання посередників чи бірж.

Атомарний обмін, скорочено від "атомарний крос-ланцюговий обмін", – це криптографічна техніка, яка дозволяє прямий обмін криптовалют між двома сторонами без необхідності довіри або сторонніх посередників. Термін "атомарний" означає, що обмін або відбувається повністю, або не відбувається взагалі – немає проміжного стану, коли одна сторона отримала кошти, а інша – ні.2. Теоретичні відомості

2.1. Поняття атомарного обміну

Атомарний обмін, скорочено від "атомарний крос-ланцюговий обмін", – це криптографічна техніка, яка дозволяє прямий обмін криптовалют між двома сторонами

без необхідності довіри або сторонніх посередників. Термін "атомарний" означає, що обмін або відбувається повністю, або не відбувається взагалі – немає проміжного стану, коли одна сторона отримала кошти, а інша – ні.

2.2 Хешований контракт з часовим замком (HTLC)

HTLC – це смарт-контракти, які встановлюють умови для переказу активів, використовуючи два ключових механізми:

- Хешові замки: Криптографічний механізм, при якому кошти блокуються за допомогою хешу і можуть бути отримані лише за умови розкриття оригінального значення, яке створює цей хеш.
- Часові замки: Обмеження на основі часу, яке гарантує, що кошти можуть бути повернуті первісному власнику через вказаний період часу, якщо обмін не завершується.

3. Реалізація

Реалізація складається з кількох ключових компонентів:

1. **Симуляція блокчейну**: Представляє блокчейни BTC та ALT
2. **Система транзакцій**: Керує переказами з умовним виконанням
3. **Управління гаманцями**: Обробляє ключі та баланси користувачів
4. **Криптографічні сервіси**: Генерує секрети та перевіряє хеші
5. **Протокол атомарного обміну**: Координує весь процес обміну
6. **Тестова структура**: Перевіряє, що реалізація працює коректно

```
     _       _                          _             ____
    / \    | |_     ___     _ __ ___   (_)   ___    / ___|  __      __   __ _    _ __
   / _ \   | __|  / _ \  | '_ ` _ \  | |  / __|   \___ \  \ \ /\ / /  / _` | | '_ \
  / ___ \  | |_  | (_) | | | | | | | | | | (__     ___) |  \ V  V /  | (_| | | |_) |
 /_/   \_\  \__|  \___/  |_| |_| |_| |_|  \___|   |____/    \_/\_/    \__,_| | .__/
                                                                             |_|
```

Atomic swap simulation for trustless cross-chain cryptocurrency exchange

Initial Balances:

Wallet	BTC	ALT
Alice	10	0
Bob	0	500

How much BTC should Alice trade? (1.0): 10
How much ALT should Bob trade? (100.0): 499
Starting atomic swap between Alice and Bob
Alice wants to trade 10 BTC for 499 ALT coins from Bob

Step 1: Alice generates secret x and its hash H(x)
Secret x (only known to Alice): y5/cXhy482...
Hash H(x) (shared with Bob): 7ab3cbe075...

Step 2: Alice creates transactions TX1 and TX2 (refund)
TX1 created: 10 BTC from Alice to Bob with hash lock
TX2 created: Refund to Alice after 48 hours if swap fails

Step 3: Alice sends TX2 to Bob for signing
Bob has signed TX2 (refund transaction)

Step 4: Alice submits TX1 to the Bitcoin blockchain

```
Step 4: Alice submits TX1 to the Bitcoin blockchain
TX1 confirmed on Bitcoin with ID: 5d60ffb5
Alice's BTC balance is now 0
Hash H(x) is now publicly visible on the blockchain: 7ab3cbe075...

Step 5: Bob creates transactions TX3 and TX4 (refund)
TX3 created: 499 ALT from Bob to Alice with same hash lock
TX4 created: Refund to Bob after 24 hours if swap fails

Step 6: Bob sends TX4 to Alice for signing
Alice has signed TX4 (refund transaction)

Step 7: Bob submits TX3 to the Altcoin blockchain
TX3 confirmed on Altcoin with ID: ffdefeb6
Bob's ALT balance is now 1

Step 8: Alice spends TX3 by revealing secret x
Alice has claimed 499 ALT by revealing secret x
Alice's ALT balance is now 499
Secret x is now publicly visible on the Altcoin: y5/cXhy482...

Step 9: Bob uses revealed x to spend TX1
Bob has claimed 10 BTC using the revealed secret x
Bob's BTC balance is now 10

Atomic swap completed successfully!
Alice traded 10 BTC for 499 ALT from Bob

Final Balances:
┌─────────┬─────┬─────┐
│ Wallet  │ BTC │ ALT │
├─────────┼─────┼─────┤
│ Alice   │   0 │ 499 │
│ Bob     │  10 │   1 │
```

Рис. 1. Результат виконання програми

Клас Blockchain

Цей клас симулює блокчейн і керує транзакціями:

```
public class Blockchain
{
      public string Name { get; private set; }
      public Dictionary<string, Transaction> Transactions { get; private set; } = new
Dictionary<string, Transaction>();

      public Blockchain(string name)
```

```
        {
        Name = name;
        }

        public string AddTransaction(Transaction transaction)
        {
        // Генерувати простий ID транзакції
        string txId = Guid.NewGuid().ToString().Substring(0, 8);
        transaction.Id = txId;
        Transactions[txId] = transaction;

        return txId;
        }

        public override string ToString()
        {
        return Name;
        }
}
```

Клас Transaction

Представляє транзакцію з умовами та можливостями часового замка:

```
public class Transaction
{
        public string Id { get; set; }
        public string From { get; set; }
        public string To { get; set; }
        public decimal Amount { get; set; }
        public Dictionary<string, object> Conditions { get; set; } = new
Dictionary<string, object>();
        public bool IsSpent { get; set; } = false;
        public DateTime Timestamp { get; set; } = DateTime.Now;
        public DateTime? TimeLock { get; set; }

        public Transaction(string from, string to, decimal amount)
        {
        From = from;
        To = to;
        Amount = amount;
        }

        // Помічник для клонування транзакції
        public Transaction Clone()
        {
        return new Transaction(From, To, Amount)
        {
                Id = Id,
                Conditions = new Dictionary<string, object>(Conditions),
                IsSpent = IsSpent,
                Timestamp = Timestamp,
                TimeLock = TimeLock
        };
        }
}
```

Клас Wallet

Керує криптографічними ключами та балансами:

```
public class Wallet
{
	public string Name { get; private set; }
	public string PublicKey { get; private set; }
	private string PrivateKey { get; set; }
	public decimal BTCBalance { get; set; }
	public decimal ALTBalance { get; set; }

	public Wallet(string name, decimal btcBalance = 0, decimal altBalance = 0)
	{
	Name = name;
	// Генерувати фіктивну пару ключів для симуляції
	PublicKey = $"{name}-public-key";
	PrivateKey = $"{name}-private-key";
	BTCBalance = btcBalance;
	ALTBalance = altBalance;
	}

	// Симулювати підпис повідомлення приватним ключем
	public string SignMessage(string message)
	{
	return $"Signed({message})-by-{Name}";
	}

	// Перевірити підпис (спрощено для симуляції)
	public bool VerifySignature(string originalMessage, string signature, string
publicKey)
	{
	return signature == $"Signed({originalMessage})-by-{publicKey.Split('-')[0]}";
	}
}
```

HashingService

Обробляє криптографічні операції:

```
public static class HashingService
{
	public static string GenerateRandomSecret()
	{
	byte[] randomBytes = new byte[32];
	using (var rng = RandomNumberGenerator.Create())
	{
		rng.GetBytes(randomBytes);
	}
	return Convert.ToBase64String(randomBytes);
	}

	public static string ComputeHash(string input)
	{
	using (SHA256 sha256Hash = SHA256.Create())
	{
		byte[] bytes = sha256Hash.ComputeHash(Encoding.UTF8.GetBytes(input));
```

```
            StringBuilder builder = new StringBuilder();
            for (int i = 0; i < bytes.Length; i++)
            {
                  builder.Append(bytes[i].ToString("x2"));
            }
            return builder.ToString();
      }
      }
}
```

Реалізація протоколу атомарного обміну

Ядром реалізації є клас AtomicSwap, який покроково реалізує протокол:

```
public class AtomicSwap
{
      private Blockchain BTCBlockchain { get; set; }
      private Blockchain ALTBlockchain { get; set; }

      public AtomicSwap(Blockchain btcBlockchain, Blockchain altBlockchain)
      {
      BTCBlockchain = btcBlockchain;
      ALTBlockchain = altBlockchain;
      }
```

```
      public async Task<SwapResult> PerformSwap(
      Wallet initiator,
      Wallet counterParty,
      decimal btcAmount,
      decimal altAmount,
      int btcTimelock = 48,
      int altTimelock = 24)
      {
      // Реалізація протоколу обміну...
      }
}
```

Крок 1: Ініціатор генерує секрет і хеш

Перший крок включає ініціатора (Сторону A), який генерує випадкове секретне значення та його хеш:

```
// Крок 1: Ініціатор (A) генерує секрет x та його хеш H(x)
AnsiConsole.MarkupLine($"\n[bold blue]Крок 1:[/] {initiator.Name} генерує секрет x та
його хеш H(x)");
string secretX = HashingService.GenerateRandomSecret();
string secretXHash = HashingService.ComputeHash(secretX);
```

Цей секрет (x) спочатку буде відомий лише ініціатору, тоді як хеш H(x) буде поширений на контрагента і, зрештою, опублікований в блокчейні.

Крок 2: Ініціатор створює транзакції

Ініціатор створює дві транзакції:

- TX1: Основна транзакція для відправлення BTC контрагенту, заблокована хешем
- TX2: Транзакція повернення коштів на випадок, якщо обмін не завершиться

```
// TX1: "Заплатити btcAmount контрагенту, якщо (x для H(x) розкрито і підписано контрагентом) або (підписано обома)"
var tx1 = new Transaction(initiator.Name, counterParty.Name, btcAmount);
tx1.Conditions["HashX"] = secretXHash;
tx1.Conditions["RequiresSignatureFrom"] = counterParty.PublicKey;

// TX2: Транзакція повернення коштів після закінчення часового замка
var tx2 = new Transaction(counterParty.Name, initiator.Name, btcAmount);
tx2.TimeLock = DateTime.Now.AddHours(btcTimelock);
tx2.Conditions["RefundFor"] = "TX1";
tx2.Conditions["SignedByInitiator"] = initiator.SignMessage($"Refund-{btcAmount}-BTC");
```

Крок 3: Контрагент підписує транзакцію повернення

Для безпеки ініціатор спочатку отримує від контрагента підпис на транзакції повернення:

```
// Контрагент підписує транзакцію повернення
tx2.Conditions["SignedByCounterParty"] = counterParty.SignMessage($"Approve-Refund-{btcAmount}-BTC");
```

Крок 4: Ініціатор відправляє TX1 в блокчейн

Після отримання підписаної транзакції повернення, ініціатор відправляє TX1 в блокчейн:

```
string tx1Id = BTCBlockchain.AddTransaction(tx1);
initiator.BTCBalance -= btcAmount; // Зменшити баланс
```

На цьому етапі ініціатор заблокував свої BTC в контракті, який можна витратити лише якщо секрет буде розкрито.

Крок 5: Контрагент створює свої транзакції

Контрагент тепер створює подібні транзакції для своїх ALT-монет:

```
// TX3: "Заплатити altAmount ініціатору, якщо (x для H(x) розкрито і підписано ініціатором) або (підписано обома)"
var tx3 = new Transaction(counterParty.Name, initiator.Name, altAmount);
tx3.Conditions["HashX"] = secretXHash; // Той самий хеш, що і в TX1
tx3.Conditions["RequiresSignatureFrom"] = initiator.PublicKey;

// TX4: Транзакція повернення коштів після закінчення часового замка
var tx4 = new Transaction(initiator.Name, counterParty.Name, altAmount);
tx4.TimeLock = DateTime.Now.AddHours(altTimelock);
tx4.Conditions["RefundFor"] = "TX3";
tx4.Conditions["SignedByCounterParty"] = counterParty.SignMessage($"Refund-{altAmount}-ALT");
```

Крок 6: Ініціатор підписує повернення для контрагента

Контрагент також отримує підпис на своїй транзакції повернення:

```
tx4.Conditions["SignedByInitiator"] = initiator.SignMessage($"Approve-Refund-
{altAmount}-ALT");
```

Крок 7: Контрагент відправляє TX3 в блокчейн

Тепер контрагент відправляє свою транзакцію в ALT-блокчейн:

```
string tx3Id = ALTBlockchain.AddTransaction(tx3);
counterParty.ALTBalance -= altAmount; // Зменшити баланс
```

Крок 8: Ініціатор отримує ALT-монети, розкриваючи секрет

Ініціатор тепер отримує ALT-монети, створюючи транзакцію, яка розкриває секрет:

```
var spendTx3 = new Transaction(counterParty.Name, initiator.Name, altAmount);
spendTx3.Conditions["RevealedSecret"] = secretX;
spendTx3.Conditions["SignedByInitiator"] = initiator.SignMessage($"Claim-{altAmount}-
ALT-with-secret");

string spendTx3Id = ALTBlockchain.AddTransaction(spendTx3);
ALTBlockchain.Transactions[tx3Id].IsSpent = true;

initiator.ALTBalance += altAmount; // Додати до балансу
```

Ця дія розкриває секрет x в ALT-блокчейні.

Крок 9: Контрагент отримує BTC, використовуючи розкритий секрет

Нарешті, контрагент може побачити розкритий секрет в ALT-блокчейні та використати його для отримання BTC:

```
var spendTx1 = new Transaction(initiator.Name, counterParty.Name, btcAmount);
spendTx1.Conditions["RevealedSecret"] = secretX;
spendTx1.Conditions["SignedByCounterParty"] = counterParty.SignMessage($"Claim-
{btcAmount}-BTC-with-secret");

string spendTx1Id = BTCBlockchain.AddTransaction(spendTx1);
BTCBlockchain.Transactions[tx1Id].IsSpent = true;

counterParty.BTCBalance += btcAmount; // Додати до балансу
```

Реалізація користувацького інтерфейсу

Додаток використовує Spectre.Console для забезпечення багатого інтерфейсу командного рядка:

```
public class Program
{
      public static async Task Main(string[] args)
      {
      // Створити заголовок ASCII-арт
```

```
        AnsiConsole.Render(
            new FigletText("Atomic Swap")
                .LeftJustified()
                .Color(Color.Green));

        // Відобразити вступну нотатку
        AnsiConsole.MarkupLine("[italic]Симуляція атомарного обміну для безтрустового крос-ланцюгового обміну криптовалют[/]");
        AnsiConsole.WriteLine();

        // Створити блокчейни
        var btcBlockchain = new Blockchain("Bitcoin");
        var altBlockchain = new Blockchain("Altcoin");

        // Створити гаманці з початковими балансами
        var alice = new Wallet("Alice", btcBalance: 10, altBalance: 0);
        var bob = new Wallet("Bob", btcBalance: 0, altBalance: 500);

        // Відобразити початкові баланси
        AnsiConsole.MarkupLine("[bold]Початкові баланси:[/]");
        DisplayBalances(alice, bob);

        // Запитати у користувача суми для обміну
        decimal btcAmount = AnsiConsole.Prompt(
            new TextPrompt<decimal>("Скільки [green]BTC[/] Аліса має обміняти?")
                .DefaultValue(1.0m)
                .Validate(amount =>
                amount <= 0 ? ValidationResult.Error("Сума має бути додатною") :
                amount > alice.BTCBalance ? ValidationResult.Error($"Аліса має лише {alice.BTCBalance} BTC") :
                ValidationResult.Success()));

        decimal altAmount = AnsiConsole.Prompt(
            new TextPrompt<decimal>("Скільки [blue]ALT[/] Боб має обміняти?")
                .DefaultValue(100.0m)
                .Validate(amount =>
                amount <= 0 ? ValidationResult.Error("Сума має бути додатною") :
                amount > bob.ALTBalance ? ValidationResult.Error($"Боб має лише {bob.ALTBalance} ALT") :
                ValidationResult.Success()));

        // Створити і виконати обмін
        var atomicSwap = new AtomicSwap(btcBlockchain, altBlockchain);

        AnsiConsole.Status()
            .Start("Обробка обміну...", async ctx =>
            {
                ctx.Spinner(Spinner.Known.Star);
                ctx.SpinnerStyle(Style.Parse("green"));

                await atomicSwap.PerformSwap(alice, bob, btcAmount, altAmount);
            });

        // Відобразити кінцеві баланси
        AnsiConsole.MarkupLine("\n[bold]Кінцеві баланси:[/]");
        DisplayBalances(alice, bob);
    }

    private static void DisplayBalances(Wallet alice, Wallet bob)
    {
```

```
        var table = new Table();

        table.AddColumn(new TableColumn("Гаманець").Centered());
        table.AddColumn(new TableColumn("BTC").Centered());
        table.AddColumn(new TableColumn("ALT").Centered());

        table.AddRow($"[green]{alice.Name}[/]", alice.BTCBalance.ToString(),
alice.ALTBalance.ToString());
        table.AddRow($"[blue]{bob.Name}[/]", bob.BTCBalance.ToString(),
bob.ALTBalance.ToString());

        AnsiConsole.Render(table);
        }
}
```

Тестування

Реалізація включає юніт-тести для перевірки функціональності атомарного обміну:

```
[Fact]
public async Task SuccessfulSwap_ShouldTransferAssets()
{
        // Arrange
        var btcBlockchain = new Blockchain("Bitcoin-Test");
        var altBlockchain = new Blockchain("Altcoin-Test");

        var alice = new Wallet("Alice", btcBalance: 10, altBalance: 0);
        var bob = new Wallet("Bob", btcBalance: 0, altBalance: 500);

        decimal btcAmount = 2.5m;
        decimal altAmount = 125m;

        var atomicSwap = new AtomicSwap(btcBlockchain, altBlockchain);

        // Act
        var result = await atomicSwap.PerformSwap(alice, bob, btcAmount, altAmount);

        // Assert
        Assert.True(result.Success);
        Assert.Equal("Swap completed successfully", result.Message);

        // Перевірити, що баланси змінилися коректно
        Assert.Equal(7.5m, alice.BTCBalance); // 10 - 2.5
        Assert.Equal(125m, alice.ALTBalance); // 0 + 125
        Assert.Equal(2.5m, bob.BTCBalance);   // 0 + 2.5
        Assert.Equal(375m, bob.ALTBalance);   // 500 - 125

        // Перевірити, що транзакції знаходяться в блокчейні і позначені як витрачені
        Assert.True(btcBlockchain.Transactions.ContainsKey(result.SwapStatus.TX1Id));
        Assert.True(btcBlockchain.Transactions[result.SwapStatus.TX1Id].IsSpent);

        Assert.True(altBlockchain.Transactions.ContainsKey(result.SwapStatus.TX3Id));
        Assert.True(altBlockchain.Transactions[result.SwapStatus.TX3Id].IsSpent);

        // Перевірити секрет і хеш
        Assert.NotNull(result.SwapStatus.SecretX);
        Assert.NotNull(result.SwapStatus.SecretXHash);
        Assert.Equal(result.SwapStatus.SecretXHash,
```

```
HashingService.ComputeHash(result.SwapStatus.SecretX));
}
```

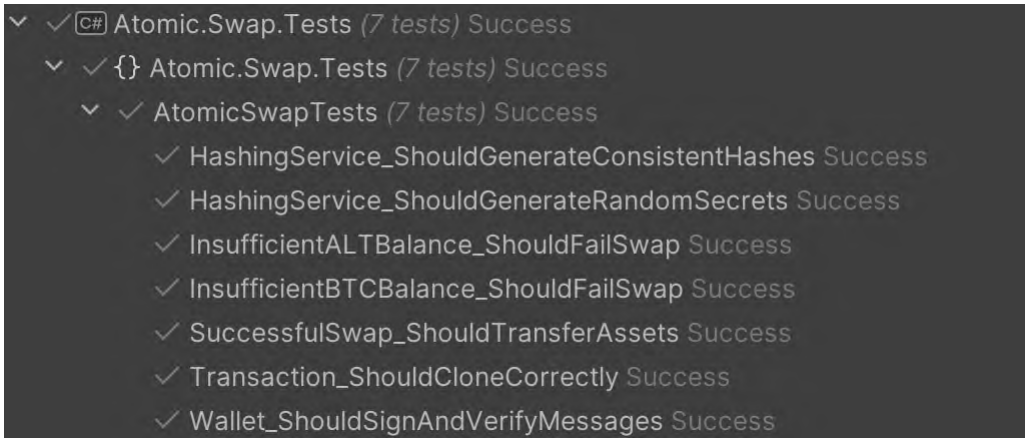

Рис. 2. Результат виконання юніт тестів

Тести також охоплюють сценарії невдачі, такі як недостатній баланс, і перевіряють функціональність хешування та криптографії.

Функції безпеки

Реалізація включає кілька ключових функцій безпеки:

1. **Хешові замки**: Забезпечують, що кошти можуть бути отримані лише якщо секрет розкрито
2. **Часові замки**: Забезпечують механізм безпеки для повернення коштів, якщо обмін не завершується
3. **Транзакції повернення**: Дозволяють обом сторонам повернути свої кошти після закінчення терміну часового замка
4. **Перевірка підпису**: Забезпечує, що лише авторизовані сторони можуть виконувати транзакції

Висновок

Ця реалізація успішно демонструє протокол атомарного обміну з використанням хешованих контрактів з часовим замком. Ключові моменти:

1. **Атомарність**: Обмін або завершується повністю, або не відбувається взагалі, захищаючи обидві сторони
2. **Безтрустовість**: Жодна сторона не може обманути іншу, оскільки протокол забезпечує справедливий обмін
3. **Відсутність третіх сторін**: Обмін відбувається безпосередньо між користувачами без посередників
4. **Крос-ланцюговість**: Працює між різними системами блокчейну

Реалізація показує, як криптографічні примітиви (хеші, часові замки та підписи) можуть бути об'єднані для забезпечення безпечної однорангової торгівлі без необхідності довіри.