МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ ТА ОБЧИСЛЮВАЛЬНІ МЕТОДИ

УДК 004.056.55

Р. Н. Квєтний[1], Є. О. Титарчук [1], А. А. Гуржій[2]

МЕТОД ТА АЛГОРИТМ ОБМІНУ КЛЮЧАМИ СЕРЕД ГРУПИ КОРИСТУВАЧІВ НА ОСНОВІ АСИМЕТРИЧНИХ ШИФРІВ ECC ТА RSA

1 – Вінницький Національний Технічний Університет, м. Вінниця
2 – Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», м. Київ

Анотація. У статті представлено порівняння швидкодії реалізацій методу обміну ключами Діффі-Геллманна для багатьох учасників на основі асиметричних алгоритмів шифрування ECC та RSA. Даний метод дає змогу групі користувачів згенерувати спільний симетричний ключ шифрування у незахищених каналах зв'язку, без необхідності використання централізованого серверу. У роботі надається приклад реалізації алгоритму на кожній з вище названих асиметричних схем шифрування у вигляді математичної моделі та алгоритму на мові програмування C#, проведено порівняння швидкості генерації асиметричних ключів шифрування та спільного симетричного ключа для різної кількості сторін.

Ключові слова: Метод Діффі-Геллмана, Гібридне шифрування, ECC, RSA.

Аннотация. В статье представлено сравнение быстродействия двух реализаций метода обмена ключами Диффи-Хелмана для группы участников на основе ассиметричных шифров ECC и RSA. Данный метод дает возможность группе пользователей сгенерировать общий ключ шифрования в незащищенных каналах связи, без необходимости использования централизованного сервера. В статье приводиться пример реализации алгоритма для каждой из выше названых асимметричных схем шифрования в виде математической модели и на языке программирования C#. Также приводиться сравнение скорости генерации асимметрических ключей шифрования и общего симметрического ключа для разного количества участников.

Ключевые слова: Метод Диффи-Хелмана, Гибридное шифрование, ECC, RSA.

Abstract. The article presents a comparison of performance implementations of key exchange method Diffie-Hellmanna for many participants based on asymmetric codes ECC and RSA. This method allows a group of users to generate a common encryption key in unprotected communication channels, without the need of a centralized server. The article provided an example of the algorithm for each of these encryption schemes in the form of mathematical models and algorithms in the programming language C#. Also, a comparison of the rate of generation of asymmetric encryption keys and of common symmetric key for a different number of participants is given.

Key words: Diffie-Hellman method, Hybrid encrypting, ECC, RSA.

Вступ

Поява асиметричних алгоритмів шифрування зробила можливим обмін ключами у незахищених каналах зв'язку. Безліч сучасних систем авторизації та ідентифікації використовують різноманітні асиметричні шифри для захисту інформації користувачів. Проте важливим обмеженням їх застосування є відносно невисока швидкодія у порівнянні з симетричними алгоритмами. Тому не є дивною поява гібридного шифрування, що дозволяє поєднати найкращі сторони симетричних та асиметричних алгоритмів та нівелювати їх недоліки. Так, для генерації спільного ключа шифрування використовується асиметричний алгоритм, коли ж кожен з учасників встановив спільний ключ шифрування, повідомлення зашифровується одним з симетричних шифрів, наприклад AES.

Прикладом алгоритму встановлення спільного ключа є протокол Діффі-Геллмана, для багатьох учасників. Фактично, будь-яка кількість учасників може узяти участь в узгоджені ключів через ітеративне виконання протоколу узгодження, при цьому проміжні данні можуть бути відкритими. Такий протокол можна реалізувати декількома асиметричними алгоритмами шифрування, найпопулярніші серед яких – RSA та ECC. [2]

Але така генерація спільного ключа шифрування вимагає значних обчислювальних потужностей і залежить від довжини ключа, а також кількості учасників які генерують спільний ключ шифрування. Метою даної статті є визначення оптимального асиметричного шифру для генерації спільних ключів шифрування серед багатьох учасників.

У роботі представлено способи реалізації протоколу Діффі-Гелмана для обміну ключами для двох та більше учасників як на основі алгоритму RSA, так і з використанням алгоритму ECC. [1, 2, 3]

Важливою характеристикою створюваної системи є швидкість генерації ключів, тому у роботі було виконано порівняння двох реалізацій протоколу на мові програмування C# та платформі .NET Framework 4.5 з використанням математичної бібліотеки MPIR [6] для розрахунків з великими числами.

Актуальність

На сьогоднішній день створено багато протоколів обміну інформацією серед груп користувачів, а також програм, що їх використовують. Дані протоколи направлені не тільки (і не стільки) на ефективність обміну інформацією, а на її захист. Проте керування ключами зазвичай відбувається лише на стороні сервера, або ж підтримує лише двох учисників, що робить такі протоколи вразливими. Зарадити

цьому може алгорит обміну ключами серед багатьох учасників, що дозволяє згенерувати спільних ключ використовуючи незахищені канали зв'язку. Тому є актуальною задача визначення оптимального асиметричного алгоритму шифрування для генерації спільних ключів шифрування серед багатьох учасників.

Мета

Метою даної статті є визначення оптимального асиметричного шифру для генерації спільних ключів шифрування серед багатьох учасників.

Задачі

1. Реалізувати протокол обміну ключами Діффі-Гелмана на основі алгоритму ECC.
2. Реалізувати протокол обміну ключами Діффі-Гелмана на основі алгоритму RSA.
3. Порівняти реалізації протоколу.

1 Опис алгоритму

В даному розділі подано протокол обміну ключами в двох реалізаціях: з використанням алгоритму шифрування RSA, та алгоритму шифрування на еліптичних кривих – ECC.

Протокол спілкування клієнтів використовує метод Діффі-Геллмана для утворення симетричного ключа шифрування. Спільні для учасників параметри шифрування генерує та зберігає сервер. Такими параметрами для алгоритму шифрування на еліптичних кривих є власне еліптична крива E(a, b), точка-генератор G, що належить даній кривій та її порядок NG, а також просте число (P_E) – модуль поля кривої. Для алгоритма шифрування RSA – це просте число A, що буде використано для утворення симетричного ключа шифрування, та (P_R), що є модулем поля. [2, 3, 5]

2 Алгоритм ECC

2.1 Генерація асиметричних ключів

Генерація приватного ключа (K_{sec}) з випадкового числа (R) має вигляд:

$$K_{sec} = R \bmod P_E \tag{1}$$

Генерація публічного ключа шифрування (K_{pub}) кожної із сторін на основі їх приватних ключів може бути описана формулою: [6]

$$K_{pub} = K_{sec} \times G \tag{2}$$

Створений на основі формул алгоритм матиме вигляд:

GenerateKey:
```
Random rnd = new Random();
var privateKey = new mpz_t(rnd.Next(GeneratorPoint.PointDimention);
if (privateKey < 0) privateKey = privateKey * -1;
var key = new ECCKey();
key.PrivateKey = privateKey;
key.OpenKey = ECCPoint.Multiply(privateKey, GeneratorPoint);
```

2.2 Генерація спільного ключа

Для генерації спільного ключа шифрування користувачі, незалежно, на основі спільних параметрів обраної криптосистеми, формують асиметричну пару ключів, а потім обмінюючись ними формують спільний секретний ключ.

Спільний ключ шифрування (K_{sym}) для і-того учасника при кількості учасників – N, та виборі алгоритму шифрування ECC має вигляд:

$$K_{sym} = K_{sec_i} \times \prod_{\substack{j=1 \\ j \neq i}}^{N} K_{pub_j} \tag{3}$$

Так, як, в еліптичній криптографії операція множення точки на точку не визначена, при реалізації даної формули відбувається передача часткових ключів між учасниками. Тобто, кожен з учасників відповідаючи на запит іншого учасника, помножує до отриманого ключа свій секретний ключ та повертає ре-

зультат. Таких повторень повинно бути строго N-1, при більшому числі запитів має місце атака Men-in-Middle і зв'язок необхідно розірвати. Також, всі передачі ключів повинні проходити перевірку на автентичність з використанням постійних відкритих ключів шифрування. Після останнього запиту учасник домножує до ключа власний приватний ключ, утворюючи, таким чином, спільний симетричний ключ шифрування.

Реалізація на мові програмування:

GenerateSymetricWithECC:

```
var tempKey = ECCPoint.Multiply(allPersons.Count, keyPair.G);
//now each person have to add his key
foreach (var person in allPersons)
{
   if (person != this)
      person.AddOwnSecretKey(tempKey);
}
//add own private key
var secretKey = ECCPoint.Multiply(keyPair.PrivateKey, symmetricKey.OpenKey);
var leftBytes = secretKey.X.ToByteArray(0);
var rightBytes = secretKey.Y.ToByteArray(0);
var resultSymetricKey = new mpz_t(left.Concat(right).ToArray(0));
```

AddOwnSecretKey:

```
tempKey = ECCPoint.Multiply(keyPair.PrivateKey, tempKey);
```

Якщо підставити формулу 2 у формулу 3, то можна побачити, що після реалізації наведеного вище алгоритму, кожен з учасників отримає однаковий симетричний ключ шифрування:

$$K_{sym} = N \times G \times \sum_{j=1}^{N} K_{sec_j} \quad (4)$$

3 Алгоритм RSA

3.1 Генерація асиметричних ключів

При використанні алгоритму RSA, спочатку утворюємо відкритий ключ, для чого необхідно згенерувати прості числа p та q, та обчислити їх добуток, отримавши модуль поля:

$$n = pq \quad (5)$$

Використовуючи функцію Ейлера ($\varphi(x)$) вибирається такий публічний ключ K_{pub}, що $1 < K_{pub} < \varphi(n)$, та який взаємно простий з $\varphi(n)$ [7]:

$$\varphi(n) = (p-1)(q-1) \quad (6)$$

Закритий ключ знаходимо як число обернене по модулю $\varphi(n)$, до відкритого ключа [8]:

$$K_{pub} \cdot K_{sec} \equiv 1 \pmod{\varphi(n)} \quad (7)$$

Створений на основі формул алгоритм матиме вигляд:

GenerateKey:

```
mpz_t p, q, e, d, n;
p.SetNumber(Generator.Random(2^512 -1, 2^512 -1));
p.RabinMiller();
```

```
q.SetNumber(Generator.Random(2^512 -1, 2^512, bytes));
q.RabinMiller();
n = p.GetPrimeNumber() * q.GetPrimeNumber();
mpz_t eulersPhiFunction = (p.GetPrimeNumber() - 1) * (q.GetPrimeNumber() - 1);
d = MathExtended.ModularLinearEquationSolver(e, 1, eulersPhiFunction);
var key = new RSAKey();
key.PrivateKey = d;
key.OpenKey = e;
key.P = n;
```

3.2 Генерація спільного ключа

Для створення симетричного ключа шифрування, кожен з учасників повинен по запиту підносити переданий йому ключ в степінь свого приватного ключа шифрування. Так, як і в описаному в розділі 2.1 алгоритмі, після останнього запиту, користувач підносить утворений ключ в степінь свого приватного ключа, для утворення спільного симетричного ключа шифрування.

$$K_{sym} = A^{\sum_{j=1}^{N} K_{sec_j}} \bmod P_R \tag{8}$$

Реалізація на мові програмування:

GenerateSymetricWithRSA:

```
var tempKey = publicPrimeA;
foreach (var person in allPersons)
   if (person != this)
      person.AddOwnSecretKey(tempKey);
var Ksym = tempKey.PowerMod(keyPair.PrivateKey, Pr);
```

AddOwnSecretKey:

```
tempKey = tempKey.PowerMod(keyPair.PrivateKey, Pr);
```

4 Порівняння реалізацій протоколу

В процесі порівняння були використані наступні характеристики алгоритмів шифрування: довжина ключа алгоритму RSA – 1024 біти, та ECC з використанням еліптичних кривих P-192 та P-512, з довжинами ключів 192 та 512 бітів відповідно. Еліптичні криві були обрані з рекомендованих національним інститутом стандартів і технологій США (NIST) [10].

Тест було виконано на комп'ютері з наступними характеристиками:

- Процесор – Intel Core i5-3230M 2.60 GHz.
- Об'єм оперативної пам'яті – 8 Gb
- Операційна система – Microsoft Windows 8.1 Pro

Процес генерації симетричного ключа шифрування складається з двох етапів:

1. Генерація асиметричних ключів шифрування кожним з учасників протоколу.
2. Обмін ключами та генерація спільного симетричного ключа шифрування.

Так, як перший з цих етапів може бути виконаний кожним з учасників окремо, то він хоча і має вплив на швидкість встановлення захищеного зв'язку, проте не залежить від кількості учасників. Середній час генерації однієї пари асиметричних ключів шифрування наведено у таблиці 1.

Таблиця 1 – Час генерації асиметричного ключа шифрування

Алгоритм	Час генерації (с)
RSA 1024	0,758
ECC P-192	0,008
ECC P-512	0,034

Отже, загальний час встановлення зв'язку (t_{res}) складається з суми часу генерації пари асиметричних ключів кожним з учасників окремо (t_a) та часу генерації спільного симетричного ключа шифрування (t_s):

$$t_{res} = t_a + t_s \quad (9)$$

Загальній час встановлення зв'язку при кількості учасників (N) від 2 до 20 представлено на графіку на рис. 1.

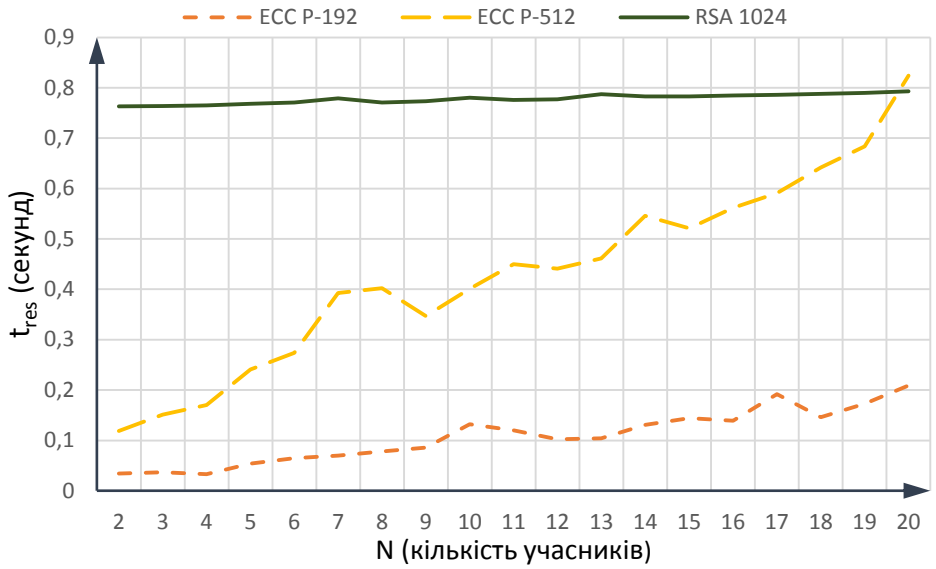

Рисунок 1 – Час встановлення спільного ключа в залежності від кількості учасників

Загальний графік залежності для кількості учасників від 2 до 300 наведено на графіку на рис. 2.

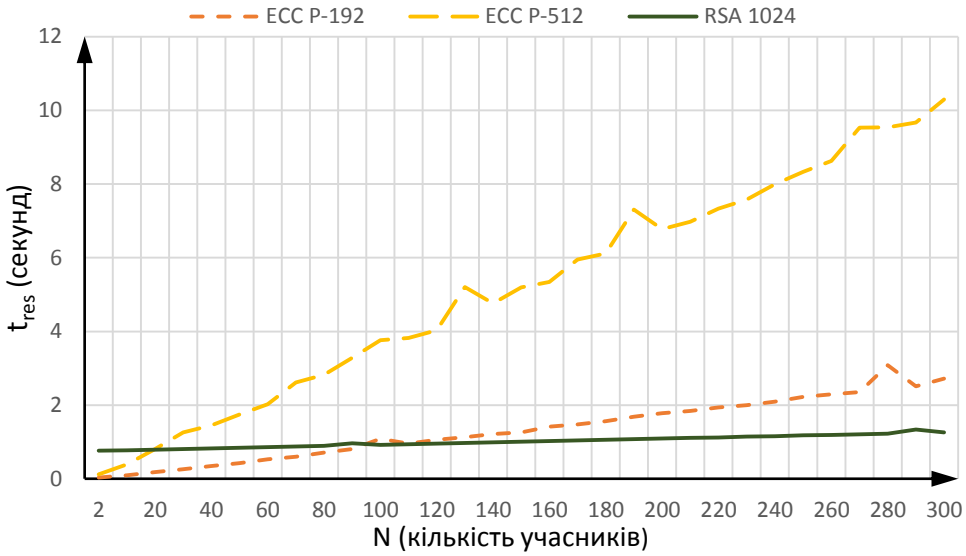

Рисунок 2 – Час встановлення спільного ключа в залежності від кількості учасників

В таблиці 2 наведено час генерації спільного ключа шифрування для 500, 700 та 900 учасників.

Таблиця 2 – Час генерації спільного ключа шифрування

Кількість учасників	Час генерації (с)		
	ECC P-192	ECC P-512	RSA 1024
500	4,353	17,022	1,595
700	7,077	23,097	1,866
900	8,420	30,980	2,387

З даних графіків можна побачити, що завдяки більшій швидкості створення асиметричних ключів алгоритми шифрування на еліптичних кривих більш продуктивні при малій кількості учасників. При цьому, якщо довжина ключа 512 біт, то загальній час встановлення зв'язку менший, ніж в алгоритмі RSA для $N < 20$, якщо ж використати алгоритм ECC з ключем 192 біти, то кількість учасників N для яких він має перевагу, збільшується до 100, що достатньо для більшості з існуючих сценаріїв.

Варто зазначити, що найбільш затратною операцією для алгоритму ECC є множення точки еліптичної кривої на число, для якої було створено декілька оптимізацій [6, 11, 12], а зважаючи на постійний ріст обчислювальних потужностей комп'ютерної техніки і швидший ріст довжини ключа алгоритму RSA, перевага алгоритму ECC буде лише збільшуватись. [1, 2, 10, 11, 12]

Висновки

1. Висновок перший. У алгоритма ECC значно менший час генерації асеметричного ключа шифрування. Найбільш затратною у данному алгоритмі є операція множення точки кривої на число, для якої існують оптимізації, що не були виконані у данній роботі.

2. Висновок другий. Результат порівняння показав, що алгоритм ECC є перспективнішим для застосування у протоколі обміну ключами серед багатьох учасників, проте його елементарна реалізація програє такій з алгоритмом RSA, якщо кількість сторін зростає до 100.

Список літератури

1. E. Titarchuk. Usage of the hybrid encryption in a cloud instant messages exchange system / R. Kvyetnyy, O. Romanyuk, E. Titarchuk, K. Gromaszek, N. Mussabekov // Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2016, 100314S, September 28, 2016

2. W. Diffie, and M. Hellman. New Directions in Cryptography / IEEE Transactions on Information Theory, Vol. IT-22, NO. 6, November 1976

3. Standards for efficient cryptography. SEC 1: Elliptic Curve Cryptography. Version 1.0 / Certicom Corp. 20, September 2000

4. Mpir.Net – Multiple Precision Integers and Rationals http://wezeku.github.io/Mpir.NET/

5. D. Hankerson, A. Menezes, and S. A. Vanstone. Guide to Elliptic Curve Cryptography / Springer-Verlag, New York 2004

6. Lawrence Washington. Elliptic Curves. Number Theory and Cryptography. 2th edition. / University of Maryland College Park, Maryland, U.S.A, 2008

7. R. Rivest, A. Shamir, L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems / Communications of the ACM, February 1978

8. D. Johnson, A. Menezes, S. Vanstone. The Elliptic Curve Digital Signature Algorithm (ECDSA) / Certicom Corporation, 2001

9. Recommended elliptic curves for federal government use / NIST, July1999

10. Bill Buchanan. Diffie-Hellman Example in ASP.NET / Bill's Security Tips, retrieved 2015-08-27 http://buchananweb.co.uk/security02.aspx.

11. Patrick Longa. Accelerating the Scalar Multiplication on Elliptic Curve Cryptosystems over Prime Fields / University of Ottawa, Canada, 2007.

12. R. P. Gallant, R. J. Lambert, S. Vanstone. Faster Point Multiplication on Elliptic Curves with Efficient Endomorphism / University of Waterloo, Canada, 2001.

Відомості про авторів

Квєтний Роман Наумович – д. т. н., професор, завідувач кафедри АІВТ.

Титарчук Євгеній Олександрович – аспірант кафедри АІВТ.

Гуржій Андрій Андрійович – к. т. н., науковий співробітник, Національний технічний університет України «Київський політехнічний інститут імені Ігора Сікорського».