DOI https://doi.org/10.36059/978-966-397-357-9-116

ДОСЛІДЖЕННЯ МЕТОДІВ КРИПТОГРАФІЧНОГО ЗАХИСТУ В ЕЛЕКТРОННОМУ ДОКУМЕНТООБІГУ

Каюков Л. Л.
магістр за спеціальністю 125 – Кібербезпека
Міжнародний гуманітарний університет
Науковий керівник: ***Йона Л. Г.***
кандидат технічних наук, доцент,
завідувач кафедри комп'ютерної інженерії та інноваційних технологій
Міжнародний гуманітарний університет
м. Одеса, Україна

Криптографія – це наука, яка досліджує засоби захисту інформації від несанкціонованого доступу. Що стосується електронного документообігу, використання криптографічних алгоритмів має вирішальне значення для забезпечення конфіденційності та цілісності даних.

Криптографічні алгоритми – це математичні процедури, що використовуються для шифрування та розшифрування інформації, створення та перевірки цифрових підписів. Симетричне та асиметричне шифрування, хеш-функції та цифрові підписи, а також керування ключами є основними компонентами, що забезпечують безпеку електронного документообігу.

Захист основних атрибутів інформації, а саме конфіденційності, цілісності та доступності, забезпечується за допомогою криптографічних алгоритмів. Якщо це забезпечення конфіденційності, тобто запобігання витоку інформації, то це можна вирішити шляхом шифрування публічних повідомлень. Зашифроване повідомлення в криптографічній формі передається одержувачу через незахищений канал. При цьому для шифрування та розшифрування повідомлень необхідно знати спільний секретний ключ шифрування [4, 5].

Бібліотека ***System.Security.Cryptography*** надає функції для виконання криптографічних операцій, таких як шифрування, розшифрування, хешування даних, аутентифікація та розподілу ключів [1–2]. В даній публікації для дослідження методів криптографічних алгоритмів створена можливість практичного використання наступних модулів, реалізованих за допомогою язика програмування c# і бібліотеки ***System.Security.Cryptography***:

– симетричне шифрування AES;
– асиметричне шифрування RSA (рис. 1, рис. 2);

– хеш-функції: SHA-256, SHA-384, та SHA-512;
– цифровий підпис RSA;
– Diffie-Hellman Key Exchange.

Симетричне шифрування: Ці алгоритми використовують один ключ для шифрування та розшифрування. Ось деякі з найпоширеніших алгоритмів симетричного шифрування: *AES (Advanced Encryption Standard); DES (Data Encryption Standard); 3DES (Triple DES); Blowfish; Twofish; IDEA (International Data Encryption Algorithm)*. Застосування симетричних систем, наприклад, **AES**, забезпечує швидке та ефективне шифрування даних, що важливо для швидкісного обміну інформацією в електронному документообігу [3]. **AES** є одним із найбільш надійних алгоритмів шифрування, оскільки він витримує численні атаки, включаючи перебір ключа і криптоаналітичні атаки. Зараз немає загрози, яка б могла швидко ламати **AES**, враховуючи поточні можливості технологій та кіберзахисту.

AES оптимізований для швидкості та легкості використання на різних платформах. Це робить його зручним для застосувань, де швидкість обробки важлива, таких як шифрування даних при передачі мережею. **Переваги**: простий у реалізації; швидкий; можливість використання різних за довжиною ключів. **Недоліки**: незручність використання секретних ключів.

Хеш-функції: *SHA-256, SHA-384, та SHA-512* – це хеш-функції, які генерують фіксований розмірний хеш-код. Вони використовуються для перевірки цілісності даних, наприклад, при збереженні та передачі електронних документів. **SHA-256, SHA-384 та SHA-512** відомі своєю стійкістю до колізій та використовуються для забезпечення цілісності даних. **Переваги**: швидкість генерації хеша; чим більша довжина хеша, тим більша стійкість до атак. **Недоліки**: можливість до колізій.

Цифровий Підпис (ЦП): Цей метод дозволяє підтверджувати автентичність та цілісність даних за допомогою цифрового підпису. В електронному документообігу **ЦП** використовується для підтвердження авторства та недоторканості важливих документів [3]. **Переваги**: забезпечує високий рівень безпеки. **Недоліки**: потребує обчислювальних ресурсів.

Крім того, алгоритми обміну ключами, такі як ***Diffie-Hellman Key Exchange***, дозволяють сторонам безпечно домовлятися про секретний ключ для подальшого симетричного шифрування [3]. ***Diffie-Hellman Key Exchange*** дозволяє двом сторонам узгоджувати спільний секретний ключ без передачі його по незахищеному каналу.

Асиметричне шифрування: У цьому методі використовуються два ключі: публічний та приватний. Ось деякі з найбільш відомих алгоритмів асиметричного шифрування: *RSA (Rivest-Shamir-Adleman);*

DSA (Digital Signature Algorithm); ECDSA (Elliptic Curve Digital Signature Algorithm); Diffie-Hellman Key Exchange (DH); ElGamal; Blum Blum Shub (BBS). **RSA** – один із відомих асиметричних алгоритмів, використовується для забезпечення безпеки обміну ключами та створення електронних цифрових підписів [3]. **RSA** є потужним алгоритмом, особливо для захисту ключів обміну та цифрового підпису. Проте він може бути менш ефективним при шифруванні великих об'ємів даних. **Переваги**: криптостійкість на основі складності факторизації великих чисел; зручність використання. **Недоліки**: потребує більше обчислювальних ресурсів порівняно з симетричним шифруванням.

```
// Зашифрування файлу за допомогою алгоритму RSA
static void EncryptFile(string inputFile, string outputFile, RSAParameters publicKey)
{
    using (RSACryptoServiceProvider rsaEncrypt = new RSACryptoServiceProvider())
    {
        rsaEncrypt.ImportParameters(publicKey);
        using (FileStream inputStream = File.OpenRead(inputFile))
        using (FileStream outputStream = File.Create(outputFile))
        {
            int keySize = rsaEncrypt.KeySize / 8;
            int blockSize = keySize - 11; // Розмір блока для RSA PKCS#1 v1.5
            int bufferSize = blockSize;
            byte[] buffer = new byte[bufferSize];
            int bytesRead;
            while ((bytesRead = inputStream.Read(buffer, 0, bufferSize)) > 0)
            {
                byte[] encryptedBytes = rsaEncrypt.Encrypt(buffer.Take(bytesRead).ToArray(), false);
                outputStream.Write(encryptedBytes, 0, encryptedBytes.Length);
            }
        }
    }
}
```

Рис. 1. Реалізація процесу зашифрування на базі бібліотеки System.Security.Cryptography алгоритму *RSA*

```
// Розшифрування файлу за допомогою приватного ключа
static bool DecryptFile(string inputFile, string outputFile, RSAParameters privateKey)
{
    try
    {
        using (RSACryptoServiceProvider rsaDecrypt = new RSACryptoServiceProvider())
        {
            rsaDecrypt.ImportParameters(privateKey);
            using (FileStream inputStream = File.OpenRead(inputFile))
            using (FileStream outputStream = File.Create(outputFile))
            {
                int keySize = rsaDecrypt.KeySize / 8;
                int blockSize = keySize; // Розмір блока для RSA PKCS#1 v1.5
                int bufferSize = blockSize;
                byte[] buffer = new byte[bufferSize];
                int bytesRead;
                while ((bytesRead = inputStream.Read(buffer, 0, bufferSize)) > 0)
                {
                    byte[] decryptedBytes = rsaDecrypt.Decrypt(buffer.Take(bytesRead).ToArray(), false);
                    outputStream.Write(decryptedBytes, 0, decryptedBytes.Length);
                }
            }
        }
        return true; // Повертаємо true, якщо розшифрування пройшло успішно
    }
    catch (CryptographicException ex)
    {
        Console.WriteLine("Помилка розшифрування: " + ex.Message);
        return false; // Повертаємо false у випадку помилки розшифрування
    }
}
```

Рис. 2. Реалізація процесу розшифровки на базі бібліотеки System.Security.Cryptography алгоритму RSA

Алгоритм RSA базується на складності факторизації великих цілих чисел. Змінюючи біти в ключах RSA з 1024 біт на 2048, 3072 і 4096, це вплине на безпеку та продуктивність алгоритму. Зазвичай, збільшення довжини ключа призводить до підвищення криптостійкості алгоритму, але це також призводить до збільшення обчислювального навантаження. Ось загальні впливи, які можна очікувати при зміні довжини ключа в алгоритмі RSA:

1) **збільшення безпеки**: збільшення довжини ключа забезпечує більшу безпеку, оскільки ускладнює факторизацію чисел і ускладнює атаки злому. З 1024 біт на 2048, 3072 і 4096, безпека значно зросте через ускладнення процесу факторизації;

2) **збільшення обчислювальних витрат**: збільшення довжини ключа призводить до збільшення обчислювальних витрат. Шифрування, дешифрування, підписи та перевірка підписів можуть займати більше часу через більшу довжину ключа;

3) **збільшення ресурсів пам'яті**: більші ключі також потребують більше пам'яті для збереження та обробки;

4) **збільшення захищеності від атак**: із збільшенням довжини ключа, алгоритм стає менш вразливим до обчислювань.

У цілому, зміна довжини ключа з 1024 біт на 2048, 3072 і 4096 забезпечить значне покращення безпеки системи, але збільшить обчислювальне навантаження. Вибір оптимальної довжини ключа повинен бути здійснений з урахуванням балансу між безпекою та продуктивністю системи залежно від конкретних потреб та обмежень. Для оцінки навантаження при зміні з 1024 біт на 2048, 3072 і 4096 будемо використовувати язик програмування c#, бібліотеку **System.Security.Cryptography** та клас **Stopwatch** для вимірювання часу виконання певних ділянок коду, в нашому випадку зашифровка і розшифровка.

Результати можна проаналізувати відносно кожного наступного розміру ключа:

1. Від 1024 біт до 2048 біт:

Збільшення розміру ключа від 1024 біт до 2048 біт призвело до збільшення часу розшифрування (з 1 хвилини 8 секунд до 2 хвилин 41 секунди) і незначного збільшення часу шифрування (з 8.6 секунд до 8.7 секунд).

2. Від 2048 біт до 3072 біт:

Знову збільшення розміру ключа до 3072 біт призвело до помітного збільшення часу розшифрування (з 2 хвилин 41 секунди до 5 хвилин 41 секунди) та збільшення часу шифрування (з 8.7 секунд до 10.3 секунд).

3. Від 3072 біт до 4096 біт:

Збільшення розміру ключа до 4096 біт призвело до подальшого збільшення часу розшифрування (з 5 хвилин 41 секунди до 9 хвилин 10 секунд) та збільшення часу шифрування (з 10.3 секунд до 12.2 секунд).

Таблиця 1

Швидкість зашифрування і розшифрування з різними бітами

Процес	Довжина ключа, біт			
	1024	**2048**	**3072**	**4096**
Зашифрування	8.6 с.	8.7 с.	10.3 с.	12.2 с.
Розшифрування	1 хв. 8 с.	2 хв. 41 с.	5 хв. 41 с.	9 хв. 10 с.

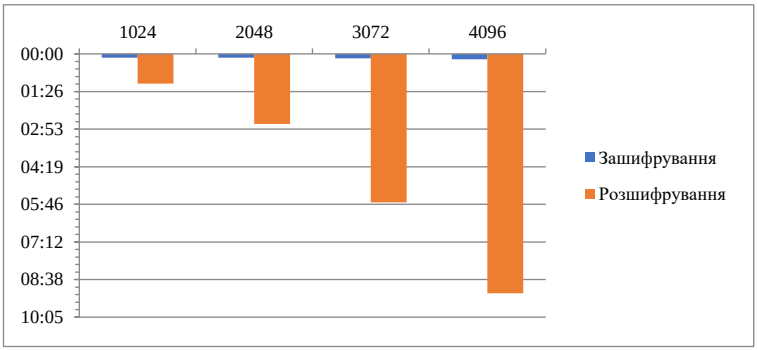

Рис. 3. Діаграма швидкості зашифрування і розшифрування алгоритмом RSA з ключем різної довжини в бітах

Отже, при збільшенні розміру ключа ми отримуємо більшу безпеку, але збільшується і час обробки даних. Можна помітити, що з кожним збільшенням розміру ключа час шифрування та розшифрування зростає, і це зростання не є лінійним. Якщо розглядати ефективність, 2048 біт ключ може бути найбільш оптимальним компромісом між безпекою та продуктивністю. 1024 біт можуть бути недостатніми для вимогливих застосувань, тоді як 4096 біт можуть бути занадто витратними у випадках, де вимагається велика продуктивність.

Бібліотека **System.Security.Cryptography** в **C#** містить готові реалізації різних криптографічних алгоритмів, таких як **RSA, AES, DES, SHA, HMAC**, та інших. Використання цієї бібліотеки має кілька переваг порівняно зі створенням власних реалізацій алгоритмів:

1) **готові реалізації**: бібліотека вже містить перевірені й оптимізовані реалізації алгоритмів, що забезпечує якість та безпеку

ваших криптографічних операцій. Власні реалізації можуть вимагати значних знань і часу для забезпечення адекватної безпеки;

2) **стандарти безпеки**: реалізації в бібліотеці **System.Security.Cryptography** відповідають стандартам безпеки та можуть бути сумісними з іншими реалізаціями, наприклад, з іншими мовами програмування або криптографічними бібліотеками;

3) **оновлення та підтримка**: криптографічні алгоритми постійно еволюціонують для відповіді на нові загрози. Бібліотека має шанс на регулярні оновлення та виправлення вразливостей, тоді як власні реалізації можуть вимагати постійного відстежування та оновлення.

У багатьох випадках, особливо для виробничого коду, використання готових реалізацій з бібліотеки ***System.Security.Cryptography*** є раціональним рішенням, оскільки вони забезпечують якість, безпеку та можливість швидкої інтеграції без необхідності повного розуміння деталей алгоритму.

Отже, дослідження надало усвідомлення про важливість вибору криптографічних методів, а також надало рекомендації стосовно оптимального розміру ключів для досягнення балансу між безпекою та продуктивністю системи в контексті електронного документообігу. Крім того, дослідження підкреслило вигоди використання бібліотек таких як **System.Security.Cryptography** які містять часткові реалізації криптографічних алгоритмів, сприяючи швидкій інтеграції та забезпечення даних в електронному середовищі. Реалізовані алгоритми які використовують бібліотеку **System.Security.Cryptography,** відповідають стандартам, що робить їх відмінним вибором для забезпечення конфіденційності. Проте, необхідно усвідомлювати, що важливо обирати алгоритми захисту інформації в залежності від їх призначення. Отже, для процесу шифрування/розшифрування краще обирати симетричні алгоритми. Асиметричні алгоритми бажано використовувати для шифрування тільки невеликих повідомлень. Для процесу автентифікації документів та управління ключами треба обирати асиметричні криптоалгоритми.

Література:

1. System.Security.Cryptography. URL: https://learn.microsoft.com/en-us/dotnet/api/system.security.cryptography

2. .NET-cryptography model. URL: https://learn.microsoft.com/en-us/dotnet/standard/security/cryptography-model

3. Технології захисту інформації. URL: https://ela.kpi.ua/bitstream/123456789/23896/1/TZI_book.pdf

4. Йона Л.Г. Криптографічний захист електронного документообігу [Текст] / Л. Г. Йона, О. О. Йона, В. С. Терешко. *Цифрові технології*. 2013. № 13. С. 142–146. URL: http://nbuv.gov.ua/UJRN/ct_2013_13_20

5. Йона Л.Г. Аналіз діючих протоколів криптографічного захисту електронних транзакцій[Текст] / / Л. Г. Йона, О. О. Кюне. Ц*ифрові технології*. 2017. Вип. 22. С. 96–102. URL: http://nbuv.gov.ua/ UJRN/ct_2017_22_13

DOI https://doi.org/10.36059/978-966-397-357-9-117

ВИКОРИСТАННЯ МЕТОДУ КОМБІНАЦІЇ ЕЛЕМЕНТІВ ДЛЯ ПОКРАЩЕННЯ ПАРАМЕТРІВ ДАТЧИКІВ ТЕМПЕРАТУРИ З МЕТОЮ ЇХ ЗАСТОСУВАННЯ У ЦИФРОВИХ СЕНСОРНИХ МЕРЕЖАХ

Козинець В. Д.
здобувач вищої освіти другого (магістерського) рівня
за спеціальністю 123 – Комп'ютерні науки
Міжнародний гуманітарний університет
м. Одеса, Україна

Горбачов В. Е.
кандидат технічних наук, доцент,
доцент кафедри інформаційних технологій
Міжнародний гуманітарний університет
м. Одеса, Україна

Більшість датчиків температури, які використаються наприклад, в космічних дослідженнях [1], в медицині [2], для систем раннього виявлення спалахів [3], для контролю енергоефективності будівель [4], мають аналоговий вихід (струм або напруга), що вимагає аналогово-цифрового перетворювача для їхнього інтегрування у цифрові сенсорні мережі. Присутність такого перетворювача неминуче призводить до спотворень у показниках датчика і потребує додаткових ресурсів та енергії. Датчики з частотним виходом позбавлені цих недоліків, тому датчики з частотним виходом становлять особливий інтерес для сучасних сенсорних мереж.

Метою даної роботи є використання методу комбінації для підвищення чутливості, покращення лінійності вихідної