**Анотація**

В бакалаврській роботі розроблено елементи системи захисту інформації у фінансових установах.

Для вирішення завдання було проведено дослідження методів захисту інформації та сфери використання криптографічних методів захисту в сучасному інформатизованому суспільстві. Також досліджено методи крипто-аналізу , блокові шифри та криптографічні хеш-функції.

В основу досліджень взяті блокові шифри, які базуються на матричних перетвореннях .Такі шифри довгий час вважались неперспективними, ручними і низькопродуктивними. З широким поширенням комп’ютерної техніки шифри на основі матричних перетворень отримали свою нішу і перспективу використання.

На базі самостійно розроблених криптографічної функції та блокового шифру було спроектовано систему автентифікації та систему для формування/перевірки електронного цифрового підпису.

Створено, власне, саме програмне забезпечення розроблених елементів системи захисту інформації.

Проведено економічний розрахунок ефективності впровадження та використання розроблених елементів.

Ключові слова: система захисту інформації, електронний цифровий підпис, криптоаналіз, блокові шифри, відкритий текст, шифрований текст, частотні характеристики, криптографічні хеш функції.

**Abstract**

In this bachelor work the elements of information security in financial institutions were developed.

To solve an assignment there was carried out research methods of data protection and the scope of the use of cryptographic protection methods in today's information society. Also the methods of cryptanalysis and block ciphers were analyzed.

As a basis of the research were taken codes based on the matrix transformations. Such ciphers were considered as unpromising, manual and unproductive for a long time. With the wide spread of computer technology codes based on matrix transformations got a niche and perspective of using.

On the basis of independently developed cryptographic function and block cipher have been designed an authentication system and system for creation / verifying digital signatures.

As the matter of fact, software has been designed for developed elements of information security.

The calculation of economic efficiency of implementation and using of the developed elements was conducted.

Keywords: system of information security, digital signature, cryptanalysis, block ciphers, plaintext, ciphertext, frequency characteristics, cryptographic hash functions.

Зміст

[ВСТУП 4](#_Toc452369594)

[РОЗДІЛ 1.ЗАХИСТ ІНФОРМАЦІЇ 8](#_Toc452369595)

[1.1. Загальне уявлення про захист інформації 8](#_Toc452369596)

[1.2. Методи захисту інформації. 9](#_Toc452369597)

[1.3. Сфери використання криптографічних методів захисту 10](#_Toc452369598)

[1.4. Криптоаналіз 11](#_Toc452369599)

[1.5. Класифікація методів криптографічного закриття інформації 13](#_Toc452369600)

[1.6. Висновки до розділу 1 14](#_Toc452369601)

[РОЗДІЛ 2.ВИКОРИСТАННЯ БЛОКОВИХ ШИФРІВ ДЛЯ ЗАХИСТУ ІНФОРМАЦІЇ 15](#_Toc452369602)

[2.1. Загальна характеристика блокових шифрів 15](#_Toc452369603)

[2.2. NIST-тестування як засіб перевірки якості шифрів 21](#_Toc452369604)

[2.3. Шифр Віженера 22](#_Toc452369605)

[2.4. Мережа Фейстеля 27](#_Toc452369606)

[2.5. Шифр Хілла 32](#_Toc452369607)

[2.6. Розробка власного шифру 34](#_Toc452369608)

[2.7. Висновки до розділу 2 39](#_Toc452369609)

[РОЗДІЛ 3.КРИПТОГРАФІЧНІ ХЕШ-ФУНКЦІЇ 41](#_Toc452369610)

[3.1. Хеш-функції. Хешування 41](#_Toc452369611)

[3.2. MD4 47](#_Toc452369612)

[3.3. SHA-1 48](#_Toc452369613)

[3.4. Розробка власної криптографічної хеш-функції 50](#_Toc452369614)

[3.5. Висновки до розділу 3 58](#_Toc452369615)

[РОЗДІЛ 4.РОЗРОБКА ЕЛЕМЕНТІВ СИСТЕМИ ЗАХИСТУ ІНФОРМАЦІЇ 60](#_Toc452369616)

[4.1. Система автентифікації 60](#_Toc452369617)

[4.2. Електронно-цифровий підпис 63](#_Toc452369618)

[4.3. Демонстрація роботи шифру та хеш-функції 66](#_Toc452369619)

[4.4. Висновки до розділу 4 67](#_Toc452369620)

[РОЗДІЛ 5.ЕКОНОМІЧНА ЧАСТИНА 69](#_Toc452369621)

[ВИСНОВКИ 70](#_Toc452369622)

[СПИСОК ЛІТЕРАТУРИ 71](#_Toc452369623)

[ДОДАТОК А 72](#_Toc452369624)

[ДОДАТОК Б 74](#_Toc452369625)

[ДОДАТОК В 76](#_Toc452369626)

[ДОДАТОК Г 78](#_Toc452369627)

[ДОДАТОК Д 80](#_Toc452369628)

[ДОДАТОК Е 81](#_Toc452369629)

# В**СТУП**

Сьогодні відомо декілька сотень шифрів, криптографічних функцій які можуть ефективно використовуватися для захисту інформації. Незважаючи на це багато спеціалістів і організацій неперервно працюють над вдосконаленням відомих шифрів і пошуком нових. Президент України недавно заявив : Кіберпростір перетворився на ще одне поле протистояння і боротьби за незалежність держави. І це застереження без перебільшень дуже важливе. Слід нагадати, що група CERT-UA при Держспецзв'язку 2014 року зафіксувала 216 кібернападів ззовні (більше половини з них - на державні установи). У 2015 році число атак збільшилося у півтора разу. Найнебезпечніша атака відбулась 23 грудня, коли було завдано масованого вдару по диспетчеру компанії «Укренерго» і 6 енергокомпаніям, що призвело до відключення енергії у 103 населених пунктах на Заході країни. Американські експерти встановили, що ця масована кібердиверсія була здійснена з боку РФ . Тому очевидно, що дослідження і вдосконалення криптографічних інструментів захисту інформації є актуальне.

Найбільш складна проблема для звичайного користувача – зрозуміти комплексну ефективність шифру, яка формується з його стійкості і простоти використання, продуктивності роботи при захисті інформації. Класична структура роботи зломщика шифрів – дослідження повторень в шифрованому тексті і дослідження статистичних характеристик частоти використання символів тексту.

Методи криптографічного захисту інформації передбачають як програмне, так і апаратне використання. Програмна реалізація шифрування є дешевшою та більш практичною. В той же час апаратна реалізація характеризується більшою продуктивністю та простотою використання.

При цьому сучасні криптографічні системи повинні задовольняти наступним загальноприйнятим вимогам :

* вихідний текст із зашифрованого тексту можна відтворити лише за допомогою ключа дешифрування;
* послідовний перебір можливих ключів дешифрування з метою відтворення вихідного тексту вимагає великого часу обчислень або приводить до неприйнятно високих затрат на реалізацію цих обчислень;
* інформація про алгоритм шифрування не повинна впливати на стійкість до зламування системи шифрування;
* незначна зміна ключа шифрування повинна приводити до суттєвих змін шифрограми одного і того ж тексту;
* незначна зміна вихідного тексту повинна приводити до суттєвих змін шифрограми при використанні одного і того ж ключа;
* структурні елементи алгоритму шифрування повинні бути незмінними;
* додаткові біти, що вводяться в повідомлення в процесі шифрування повинні бути надійно закритими в зашифрованому тексті;
* не повинно бути простих залежностей між ключами, що послідовно використовуються при шифруванні;
* довільний ключ із множини використовуваних ключів повинен забезпечувати надійність системи шифрування;
* алгоритми шифрування й дешифрування повинні допускати як апаратну, так і програмну реалізацію, при цьому зміни довжин ключів не повинні приводити до якісного погіршення алгоритмів.

Якщо уявити собі фінансову установу, то зрозуміло, що створення елементів системи захисту інформації не є найважливішою задачею установи, такою надання фінансових послуг і отримання прибутку, але вони взаємозалежні.

Ми будемо розглядати проблеми створення елементів системи захисту інформації для невеликої фінансової установи(місцеві фінансові фонди, приватні регіональні банки, організації лотерей і т.д.). Звичайно, що на створення елементів системи захисту інформації захочуть виділити мінімум коштів (в кращому випадку одиниці відсотків від прибутку), а вимоги до елементів системи захисту інформації будуть високі.

Для того щоб захистити інформацію у фінансовій установі потрібно насамперед дослідити існуючі шифри та криптографічні хеш-функції, їхню стійкість. В даній дипломній роботі ми будемо розробляти власні високонадійні шифр та хеш-функцію і на їх основі проектувати елементи системи захисту інформації(систему автентифікації та систему формування/перевірки електронного цифрового підпису).

**Мета дипломного проекту:** на основі відомих блокових шифрів розробити власні шифр та криптографічну хеш-функцію. Розробити елементи системи захисту інформації які фінансова установа зможе використати для захисту конфіденційної інформації. елементи.Розробити програмне забезпечення для реалізації елементів системи захисту інформації.

**Предмет дослідження:** розроблені блоковий шифр та хеш-функція.

**Об’єкт**: процес захисту даних.

**Основний зміст роботи**

**В першому розділі** досліджуються поняття захисту інформації сфери використання криптографічних методів захисту в сучасному суспільстві і методи криптоаналізу. Приймається рішення щодо вибору типу шифрів для подальшого їхнього аналізу при розробці елементів системи захисту інформації.

**В другому розділі** здійснюється детальний аналіз блокових шифрів, а саме шифри Хілла, Віженера, Фейстеля. Представлений детальний опис процесу шифрування і дешифрування. Розробляється власний унікальний шифр на основі одного із базових та проводиться дослідження на стійкість.

**В третьому розділі** здійснюється детальний аналіз блокових хеш-функцій, а саме MD5 та SHA-1. Представлений детальний опис процесу хешування. Розробляється власний унікальний шифр на основі одного із базових та проводиться дослідження на стійкість.

**В четвертому розділі** розроблене ПЗ в якому використовуються розроблені блоковий шифр та криптографічна хеш-функція.

**В п’ятому розділі** економічна частина відображено доцільність проектного рішення (програмного продукту) шляхом економічних розрахунків, а також обґрунтування можливостей використання обраної стратегії. Наведено економічну оцінку розроблення програмного продукту з точки зору його окупності.

# РОЗДІЛ 1.ЗАХИСТ ІНФОРМАЦІЇ

В останні декілька десятків років надзвичайно актуальною стала проблема захисту інформації. В багатьох сферах діяльності інформація – це найцінніший ресурс, тому цілком закономірним є необхідність в захисті інформації. Зокрема в шифруванні даних при передачі через відкриті канали зв'язку, обслуговування банківських пластикових карток, формування електронного цифрового підпису, зберігання та ідентифікація паролів користувачів у інформаційній банківській мережі – все це далеко не повний перелік сфер використання.

Те, що інформація має цінність, люди усвідомили дуже давно. Тоді ж і з’явилося завдання захисту від осіб, для яких ці дані не призначаються. Лише декілька десятиліть тому все змінилося - інформація придбала самостійну комерційну цінність і стала широко поширеною, майже звичайним товаром. Її проводять, зберігають, транспортують, продають і купують, а значить - крадуть і підробляють - тому її необхідно захищати. Зараз існує багато різних способів для роботи з інформацією, в тому числі і конфіденційною ;більше таких способів – більше можливостей знайти лазівку, що дозволяє зловмиснику заволодіти інформацією. Сучасне суспільство все більшою мірою стає інформаційно-обумовленим, успіх будь-якого виду діяльності все сильніше залежить від володіння певною інформацією та від відсутності її у конкурентів.

## 1.1. Загальне уявлення про захист інформації

Сучасні методи обробки, передачі та накопичення інформації сприяли появі загроз, пов'язаних з можливістю втрати, перекручування та розкриття даних, що адресовані або належать кінцевим користувачам. Тому забезпечення інформаційної безпеки комп'ютерних систем і мереж є одним з провідних напрямків розвитку ІТ.

Захист інформації - це діяльність щодо запобігання витоку інформації, що захищається, несанкціонованих і ненавмисних дій відносно захищеної інформації.

Об'єкт захисту - інформація, носій інформації або інформаційний процес, щодо яких необхідно забезпечувати захист у відповідності до поставленої мети захисту інформації.

Мета захисту інформації - це бажаний результат захисту інформації. Метою захисту інформації може бути запобігання шкоди власнику, користувачеві інформації в результаті можливого витоку інформації та / або несанкціонованого і ненавмисного впливу на інформацію.

Ефективність захисту інформації - ступінь відповідності результатів захисту інформації поставленої мети.

Криптографічний метод захист інформації передбачає перетворення інформаційних даних у незрозумілий і нечитабельний вигляд, для того, щоб ворог не зміг її прочитати.

Ідея шифрування полягає в запобіганні перегляду дійсного змісту повідомлення (тексту, файлу і т.д.) тими, у кого немає засобів його дешифрування.

Поруч із криптографією розвивається і криптоаналіз. Це наука про аналіз шифрів, їх властивостей і відкриття методів злому шифру. Успішно виконаний криптоаналіз дає можливість отримати ключ шифрування або відкрити текст (або і ключ, і текст). Деколи криптографію і криптоаналіз об'єднують в одну науку – криптологію. Ця наука займається питаннями взаємного перетворення інформації, оцінкою надійності систем шифрування і аналізом стійкості шифрів.

Для того щоб зрозуміти, які компоненти безпеки та яким чином їх необхідно використовувати, бажано представити собі, від кого вони мають захистити систему і хто може бути зловмисником. Можна виділити 4 групи та кваліфікувати щодо важливості як захисту, так і злому. 1 група – це системи із інформацією найбільш високого рівня, переважно загальнодержавного та військового значення, і захист повинен бути високої якості. 2 група – це системи із інформацією бізнесового та промислового значення для великих бізнес-груп, корпорацій, фірм, банківських структур. 3 група – це системи із інформацією бізнесового та господарського значення малих підприємств, окремих середніх підприємств, які не в стані виділяти великі кошти на безпеку і захист інформації. 4 група – це системи із інформацією бізнесового, господарського та побутового значення окремих осіб (приватна інформація), бізнесменів, які не мають намірів (в деяких випадках і можливостей) витрачати значні кошти для захисту інформації і використовують широковідомі засоби захисту. Кожен рівень вимагає відповідного підходу як у виборі засобів захисту, так і для протидії їм. Дані дисертаційні дослідження орієнтовані на засоби і методи захисту і протидії для 4, 3 і частково 2 групи складності. Визначення рівня складності визначає однозначно методи протидії. Кібервійська переважно використовуються для систем 1 групи складності. Для систем 2, 3 і 4 групи складності доцільно застосовувати перспективні комп’ютерні шифри і компютерні засоби захисту із використанням біометричної інформації, в яких є можливість використовувати їх легку адаптацію при зміні умов захисту (зміна ключів, паролів, додаткових умов протидії злому), які не вимагають великої обчислювальної роботи, які незначно збільшують об’єм шифрованого повідомлення при прийнятній ефективності захисту інформації.

## 1.2. Методи захисту інформації.

На практиці сформувалось багато методів захисту інформації, які можна поділити на такі три групи:

Перша група. Фізичний захист носія інформації від ворога. Носій інформації (даних) може бути папір, комп’ютерний носій інформації (CD, DVD-диски, флеш-карта, магнітний диск HDD, мобільний телефон – його пам’ять). Для реалізації цього способу необхідно мати надійний канал зв’язку, який недоступний для ворога. В різні часи використовувались для цього поштові голуби, спеціальні кур’єри, радіопередачі на секретній частоті. Методи фізичного захисту інформації використовується і в сучасних автоматизованих системах обробки даних.

Друга група. Стеганографічний захист інформації. Цей метод використовували з доісторичних часів. Суть цього методу – приховати від ворога сам факт пересилання інформації, а також приховати наявність інформації, яка може його зацікавити. Від ворога чи зловмисника приховують фізичний носій даних або маскують секретні повідомлення серед відкритих.

У сучасних умовах з використанням інформаційних технологій є багато можливостей для стеганографії. Графічні файли, фотографії, фільми, музика в чиїх молодших розрядах можна ховати шифрований текст.

Третя група. Криптографічний захист інформації. Цей метод передбачає таке перетворення інформації, щоб ворог не зміг її прочитати .

## 1.3. Сфери використання криптографічних методів захисту

В сучасному інформатизованому суспільстві захист інформаційних даних стає невід’ємною частиною життєдіяльності. Список сфер використання криптографічних методів захисту даних в сучасному інформатизованому суспільстві стає дедалі більшим, серед яких є наступні :

- шифрування даних при передачі через відкриті канали зв'язку (інтернет – інформація про клієнта, адреса, телефон, банківські реквізити);

- обслуговування банківських пластикових карт;

- зберігання та ідентифікація паролів користувачів у інформаційній банківській мережі;

- передача бухгалтерських звітів через віддалені канали зв’язку;

- банківське обслуговування підприємств через локальні і глобальну мережу;

- безпечне зберігання даних HDD ПК за рахунок використання методів шифрування інформації(захист від несанкціонованого доступу).

## 1.4. Криптоаналіз

Головною дійовою особою в криптоаналізі виступає порушник(або криптоаналітик). Під ним розуміють особу (групу осіб), метою яких є прочитання або підробка захищених криптографічними методами повідомлень.

Відносно порушника приймається ряд припущень, які, як правило, кладуться в основу математичних чи інших моделей:

1. Порушник знає алгоритм шифрування (або вироблення ЕЦП) і особливості його реалізації в конкретному випадку, але не знає секретного ключа.

2. Порушнику доступні всі зашифровані тексти. Порушник може мати доступ до деяких вихідних текстів, для яких відомі відповідні їм зашифровані тексти.

3. Порушник має в своєму розпорядженні обчислювальні, людські, тимчасові та інші ресурси, обсяг яких виправданий потенційною цінністю інформації, яка буде здобута в результаті криптоаналізу.

Спробу прочитання або підробки зашифрованого повідомлення, вираховування ключа методами криптоаналізу називають криптоатакою або атакою на шифр. Вдалу криптоатаку називають зломом.

Криптостійкістю називається характеристика шифру, що визначає його стійкість до розшифрування без знання ключа (тобто криптоатаці).

Показник криптостійкості - головний параметр будь криптосистеми. В якості показника криптостійкості можна вибрати:

- кількість всіх можливих ключів або можливість вибору ключа за заданий час з заданими ресурсами;

- кількість операцій або час (з заданими ресурсами) , необхідних для злому шифру із заданою вірогідністю;

- вартість обчислення ключової інформації або вихідного тексту.

Всі ці показники повинні враховувати також рівень можливої криптоатаки.

Однак слід розуміти, що ефективність захисту інформації криптографічними методами залежить не тільки від криптостійкості шифру, але і від безлічі інших чинників, включаючи питання реалізації криптосистем у вигляді пристроїв або програм. При аналізі криптостійкості шифру необхідно враховувати і людський фактор. Наприклад, підкуп конкретної людини, в руках якого зосереджена необхідна інформація, може коштувати на кілька порядків дешевше, ніж створення суперкомп'ютера для злому шифру

Сучасний криптоаналіз спирається на такі математичні науки як теорія ймовірностей і математична статистика, алгебра, теорія чисел, теорія алгоритмів і ряд інших. Всі методи криптоаналізу в цілому укладаються в чотири напрямки.

1. Статистичний криптоаналіз - досліджує можливості злому криптосистем на основі вивчення статистичних закономірностей початкових і зашифрованих повідомлень. Його застосування ускладнене тим, що в реальних криптосистемах інформація перед шифруванням піддається стиску (перетворюючи вихідний текст в випадкову послідовність символів), або в разі гамування використовуються псевдовипадкові послідовності великої довжини.

2. Алгебраїчний криптоаналіз - займається пошуком математично слабких ланок криптоалгоритмів. Наприклад, в 1997 р в еліптичних системах був виявлений клас ключів, що істотно спрощує криптоаналіз.

3. Диференціальний (або різницевий) криптоаналіз - заснований на аналізі залежності зміни шифрованого тексту від зміни початкового тексту. Вперше використаний Мерфі, поліпшений Біхемом і Шаміром для атаки на DES.

4. Лінійний криптоаналіз - метод, заснований на пошуку лінійної апроксимації між вхідним і шифрованим текстом. Запропонований Мацуї, також вперше був застосований при зломі DES. Як і диференціальний аналіз в реальних криптосистемах може бути застосований лише для аналізу окремих блоків криптоперетворень.

Досвід зломів криптосистем (зокрема, конкурсів, які регулярно влаштовує RSA Data Security) показує, що головним методом залишається "лобова" атака - проба на ключ. Також як показує досвід криптосистеми більше страждають від недбалості в реалізації.

Прийнято розрізняти кілька рівнів криптоатаки в залежності від обсягу інформації, доступної криптоаналітику. Можна виділити три рівня криптоатаки по наростанню складності.

1. Атака по шифрованому тексту (Рівень КА1) - порушнику доступні всі або деякі зашифровані повідомлення.

2. Атака по парі "вихідний текст - шифрований текст" (Рівень КА2) - порушнику доступні всі або деякі зашифровані повідомлення і відповідні їм вихідні повідомлення.

3. Атака за обраною парою "вхідний текст - шифрований текст" (Рівень КА3) - порушник має можливість вибирати вхідний текст, отримувати для нього зашифрований текст і на основі аналізу залежностей між ними обчислювати ключ.

Всі сучасні криптосистеми володіють достатньою стійкістю навіть до атак рівня КА3, тобто коли порушнику доступний по суті шифрувальний пристрій.

## 1.5. Класифікація методів криптографічного закриття інформації

В даний час відомо велика кількість методів криптографічного закриття інформації. Класифікація методів шифрування (криптоалгоритмів) може бути здійснена за такими ознаками:

• за типом ключів: симетричні криптоалгоритми; асиметричні криптоалгоритми;

• за розміром блоку інформації: потокові шифри; блокові шифри;

• за характером впливів, вироблених над даними: метод заміни (перестановки), метод підстановки; аналітичні методи, адитивні методи (гамування), комбіновані методи.

Кодування може бути смислове, символьне, комбіноване.

Закриття інформації іншими способами може досягатися за допомогою стеганографії, стиснення / розширення, розсічення / рознесення.

## 1.6. Висновки до розділу 1

В першому розділі бакалаврської роботи визначене поняття «захист інформації» і супутні поняття, розглянуто методи захисту інформації і їх сфери використання в сучасному інформатизованому суспільстві, основи криптографії, методи криптоаналізу.

В результаті проведеного аналізу сформульовані основні поняття, які стосуються захисту інформації. Прийнято рішення розглядати блокові шифри, які використовуватимуться при розробленні елементів системи криптографічного захисту інформації.

# РОЗДІЛ 2.ВИКОРИСТАННЯ БЛОКОВИХ ШИФРІВ ДЛЯ ЗАХИСТУ ІНФОРМАЦІЇ

Взагалі захист інформації – це складна багатовекторна задача. Можна використовувати багато різноманітних засобів для захисту інформації. Це можуть бути інженерно-технічні, програмно-апаратні, криптографічні засоби. Багато авторів пропонують різні варіанти і стверджують про їх унікальність і ефективність. Але треба завжди пам’ятати – 100% систему захисту інформації, яку ніхто ніколи не розкриє створити неможливо і найголовніше – нема потреби. Для комерційних підприємств важливо зберегти свою інформацію на період, поки вона несе інформаційно-актуальний характер (періоди патентування, укладення договорів, випуск нової продукції та інші ). Якщо уявити собі підприємство, то зрозуміло, що створення системи захисту інформації не є найважливішим завданням підприємства, таким як виготовлення продукції і отримання прибутку, але вони взаємозалежні. У фінансових установах крім свого стандартного застосування шифри можуть використовуватися як складова електронного цифрового підпису.

## 2.1. Загальна характеристика блокових шифрів

Блоковий шифр - шифр який оперує групами біт фіксованої довжини - блоками, характерний розмір яких змінюється в межах 64-256 біт. Якщо вихідний текст (або його залишок) менше розміру блоку, перед шифруванням його доповнюють. Фактично, блоковий шифр являє собою підстановку на алфавіті блоків, яка, як наслідок, може бути моно- або поліалфавітною. Блоковий шифр є важливим компонентом багатьох криптографічних протоколів і широко використовується для захисту даних, що передаються по мережі.

Режим шифрування – метод застосування блокового шифру, що дозволяє перетворювати послідовність блоків відкритих даних в послідовність блоків зашифрованих даних. При цьому для шифрування одного блоку можуть використовуватися дані іншого блоку. Зазвичай режими шифрування використовуються для модифікації процесу шифрування так, щоб результат шифрування кожного блоку був унікальним без залежності від даних, що шифруються ,і не дозволяв зробити які-небудь висновки про їх структуру. Це обумовлено, перш за все, тим, що блокові шифри шифрують дані блоками фіксованого розміру, через це існує потенційна можливість витоку інформації про повторювані частини даних, що шифруються одним і тим же ключем. Існує декілька стандартних режимів шифрування.

1.Electronic Codebook(ECB)

Повідомлення ділиться на блоки. Кожен блок шифрується окремо(незалежно від інших та одним ключем).Цей режим називається режимом електронної кодової книги, так як існує можливість створити книгу, в якій кожному блоку відкритого тексту буде співставлений блок шифрованого тексту. Проте створити книгу – нетривіальна задача. Якщо розмір блока рівний x біт, то в книзі буде міститися 2х записів і кожна книга буде відповідати одному ключу.

Шифрування може бути описане таким чином:

(1)

Тут Ci та Pi – блоки шифрованого і відкритого текстів відповідно ,а F – функція блокового шифрування. Розшифрування по тій самій схемі. Тобто:

(2)

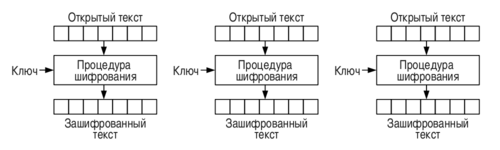


Рис.1.Шифрування в режимі ECB

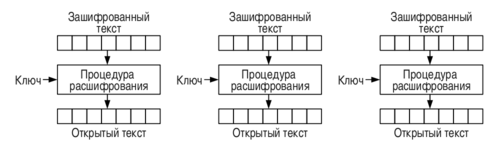


Рис.2. Розшифрування в режимі ECB

Недоліки:

* Ідентичні блоки відкритого тексту шифруються в ідентичні блоки зашифрованого тексту при такому ж ключі; таким чином, цей метод погано приховує структуру даних.
* Блоки можуть з’являтися чи пропадати. Зловмисник може перехопити блок і продублювати його, а зі сторони адресата він буде сприйнятий як правильний. Таким чином, не рекомендується використовувати в криптографічних протоколах.

2.Cipher Block Chaining(CBC)

Для зчеплення використовується механізм зворотнього зв’язку, оскільки результат шифрування попередніх блоків використовується для шифрування поточного блоку. Цей метод називається «режим зчеплення блоків шифротексту». Таким чином, будь-який блок шифру залежить не тільки від вхідного тесту, а й від всіх попередніх блоків тексту. В CBC текст XOR-риться з попереднім зашифрованим блоком перед шифруванням. Дешифруція проводиться аналогічно. Математичний запис:

(3)

(4)

При цьому на початку шифрування використовується вектор ініціалізації для того, щоб будь-яке повідомлення було по-справжньому унікальним(інакше будуть труднощі зі стандартним заголовком).Вектор ініціалізації повинен бути випадковим числом. Його не обов’язково тримати у секреті, можна передавати разом із повідомленням.

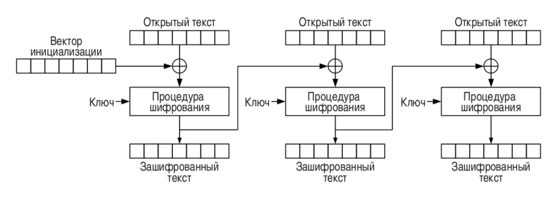


Рис.3.Шифрування в режимі CBC

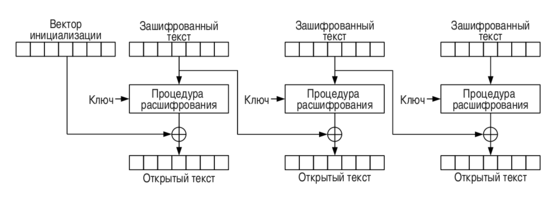


Рис.4.Розшифрування в режимі CBC

Більшість повідомлень не ділиться націло блоки. Зазвичай залишається короткий блок в кінці. Можна по-різному боротися з ним. Найпростіший метод – padding(доповнення до повного блоку).Якщо потрібно потім прибрати сміття, то достатньо просто останнім байтом позначити кількість зайвим байтів.

3.Cipher Feedback(CFB)

Режим (CFB) зворотнього зв’язку шифру - близький родич CBC,що перетворює блоковий шифр в самосинхронізуючий шифр потоку. Цей метод також називається «режим зворотнього зв’язку по шифротексту» .Операція дужа схожа на попередню; наприклад, розшифрування CFB майже ідентична розшифруванню CBC,виконаному навпаки:

(5)

(6)

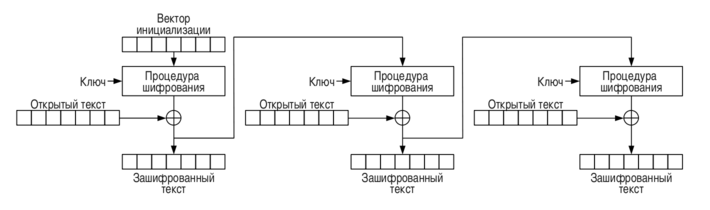


Рис.5.Шифрування в режимі CFB

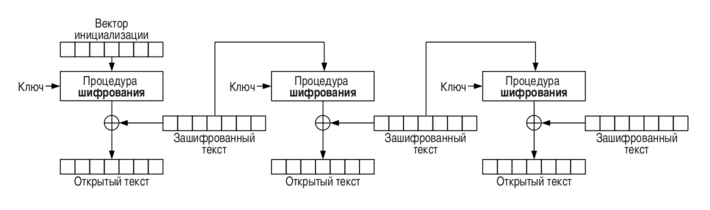


Рис.6.Розшифрування в режимі CFB

Оскільки кожна стадія режиму CFB залежить від зашифрованого значення попереднього зашифрованого тексту операцією складання з поточним значенням відкритого тексту, форма конвеєрної обробки можлива, починаючи з єдиного кроку кодування, який вимагає, щоб відкритий текст був кінцевою операцією XOR .Це корисно для застосувань, які вимагають малого часу очікування між прибуттям відкритого тексту і виводу відповідного зашифрованого тексту.

4.Output Feedback(OFB)

Режим (OFB) зворотнього зв’язку виводу перетворює блочний шифр в синхронний шифропотік: це генерує ключові блоки, які є результатом складання з блоками відкритого тексту, щоб отримати зашифрований текст. Так само, як і з іншими шифруми потоку, дзеркальне відображення в зашифрованому тексті продукує дзеркально відбитий біт у відкритому тексті в тому ж місці розташування. Ця властивість дозволяє багатьом кодам з виправленням помилок функціонувати як звичайно, навіть тоді, коли виправлення помилок застосоване перед кодуванням.

Через симетрію операції XOR шифрування та розшифрування схожі:

(7)

(8)

(9)

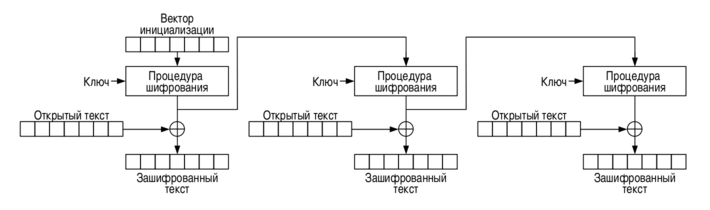


Рис.7.Шифрування в режимі OFB

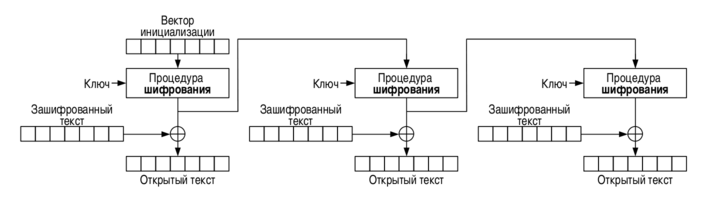


Рис.8.Розшифрування в режимі OFB

Кожна операція блокового шифру зворотнього зв’язку виводу залежить від усіх попередніх ,тому не може бути виконаним паралельно. Однак через те, що відкритий текст чи шифрований текст використовується тільки для фінального складання, операції блокового шифру можуть бути виконані заздалегідь, дозволяючи виконати фінальне шифрування паралельно з відкритим текстом.

Цей метод називається також «режим зворотнього зв’язку по виходу».

## 2.2. NIST-тестування як засіб перевірки якості шифрів

Випадкові послідовності відіграють важливу роль в криптографії. Випадковість та криптографія тісно пов’язані: основна ціль криптографічних систем – перетворення невипадкових осмислених відкритих тестів в послідовність символів, що здається випадковою. Як доказав математик Клод Шенон, шифри з характеристиками, які близькі до рівноймовірних, наближаються до абсолютно стійких

Для перевірки якості методу шифрування можна використати тести NIST USA (The National Institute of Standards and Technology) . Кожний NIST-тест отримує на вхід скінченну бітову послідовність. Потім здійснюється обчислення статистики, яка характеризує певну рису послідовності – це може бути як одиничне значення, так і множина значень. Після цього обчислена статистика порівнюється із еталоном, який дає ідеальна випадкова послідовність.

Основа тестів – поняття нульової гіпотези. Нульова гіпотеза - це припущення ,що послідовність є істинно випадковою(біти якої з’являються рівномірно та незалежно один від одного).

Існує також альтернативна гіпотеза, яка спростовує нульову гіпотезу. Тобто, якщо нульова гіпотеза вірна, то генератор продукує достатньо «хороші» випадкові символи. Так, як ми маємо справу з гіпотезами, то існує 4 варіанти розвитку подій:

1.Зроблено висновок : послідовність випадкова і це вірний висновок.

2.Зроблено висновок : послідовність невипадкова, хоча вона насправді випадкова. Такі помилки називають помилками першого роду.

3.Послідовність визнається випадковою, хоча такою не є насправді. Такі помилки називаються помилками другого роду.

4.Послідовність цілком справедливо відбракована.

Ймовірність помилки першого роду називають рівнем статистичної значимості і позначають α.Тобто α – це ймовірність відбракувати «хорошу» послідовність .Це значення визначається областю застосування. В криптографії прийнято α брати від 0.001 до 0.01.В кожному тесті визначається так зване P-значення: це ймовірність того ,що піддослідний генератор спродукує послідовність не гіршу, ніж гіпотетичний істинний. Якщо P-значення = 1 ,то послідовність ідеально випадкова, якщо воно = 0,то послідовність повністю передбачувана. В подальшому P-значення порівнюється з α,і якщо воно більше за α,то нульова гіпотеза приймається і послідовність визнається випадковою. В іншому випадку – відбраковується.

В NIST-тестах береться α = 0.01. З цього слідує:  
 - якщо P-значення >= 0.01,то послідовність визнається випадковою з рівнем довіри 99%

- якщо P-значення < 0.01,то послідовність відбраковується з рівнем довіри 99%.

Якщо шифр успішно проходить всі тести, то шифр визнається як такий ,що має надзвичайно хорошу якість.

## 2.3. Шифр Віженера

#### ***2.3.1. Опис***

Шифр Віженера – це метод шифрування повідомлень за допомогою серій різних шифрів Цезаря, з використанням ключового слова.

В шифрі Цезаря кожна літера алфавіту зміщується на декілька позицій. Наприклад, в шифрі Цезаря зі зміщенням рівним трьом ,А перетвориться в D,B в E і так далі. Шифр Віженера складається з декількох шифрів Цезаря з різними значеннями зміщення.

Для шифрування може бути використана tabula recta(квадрат Віженера).Він складається з алфавіту, записаного 26 разів в різних рядках так, що кожен алфавітний рядок циклічно зсунутий вліво, порівняно з попереднім рядком. В різних точках процесу шифрування, шифр використовує різні алфавітні рядки з квадрату. Алфавіт, що використовується в кожний момент часу залежить від повторюваного ключового слова.

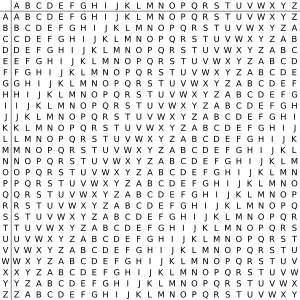


Рис.9.Квадрат Віженера

Наприклад, припустимо, що відкритий текст такий:

ATTACATDAWN

Особа, що надсилає повідомлення вибирає ключове слово і дублює його, доки воно не буде мати довжину рівну довжині відкритого тексту. Наприклад, для ключового слова «LEMON»:

LEMONLEMONLE

Кожен рядок починається з букви ключа. Решта рядка складається з літер від A до Z(у зміщеному порядку).З усіх 26-ти рядків буде використано стільки ключів(різних алфавітів),скільки є унікальних літер у ключовому слові .В прикладі лише 5 букв. Для послідовних літер у повідомленні ми будемо брати послідовно букви із ключового слова і шифрувати кожну літеру повідомлення, використовуючи відповідний рядок в квадраті Віженера.

Приклад:

|  |  |
| --- | --- |
| Відкритий текст: | ATTACKATDAWN |
| Ключове слово: | LEMONLEMONLE |
| Зашифрований текст: | LXFOPVEFRNHR |

Розшифрування проводиться таким чином: знаходимо в таблиці Віженера рядок, відповідний першому символу ключового слова; в цьому рядку знаходимо перший символ зашифрованого тексту. Стовпець, в якому знаходиться даний символ, відповідає першому символу початкового тексту. Наступні символи зашифрованого тексту розшифровуються за таким же принципом.

#### 2.3.2. Математичне представлення

Це може бути представлено алгебрично. Якщо букви A-Z будуть представлені як номери 0-25,шифрування E , використовуючи ключ K, може бути записане в такому вигляді:

(10)

а розшифровка D,використовуючи ключ K:

(11)

де *M=M1…Mn*– це повідомлення ,*C=C1…Cn* –це шифрований текст і *K=K1…Kn* – це ключ, одержаний повторенням ключового слова *n/m* разів, де *m* – довжина ключового слова.

Щоб зашифрувати літеру A=0 за допомогою букви ключового слова L=11 здійснюється таке обчислення:

*11=(0+11) mod 26*

Щоб розшифрувати літеру R=17 ,використовуючи букву E=4:

*13=(17-4) mod 26*

#### 2.3.3. Переваги та недоліки

Ідея шифру Віженера, як і всіх поліалфавітних шифрів, полягає в тому, щоб розмити частотні характеристики символів у тексті, щоб неможливо було застосувати простий частотний аналіз. Проте деякі особливості появи символів у тексті залишаються.

Головна ж слабкість полягає у повторювальному характері ключа. Якщо криптоаналітик правильно вгадує довжину ключа, то зашифрований текст можна розглядати як комбінацію шифрів Цезаря, що легко ламається. Визначити довжину ключа можна за допомогою методів Касіскі та Фрідмана.

#### 2.3.4. Результати NIST-тестування

Frequency (monobit) test

(input) *n =* 1000

(processing) *Sn =* -150

(processing)

(output)

(conclusion) Since *P-value* ≥ *0.01*, accept the sequence as non-random.;

Тест не пройдено,подальше тестування недоцільне.

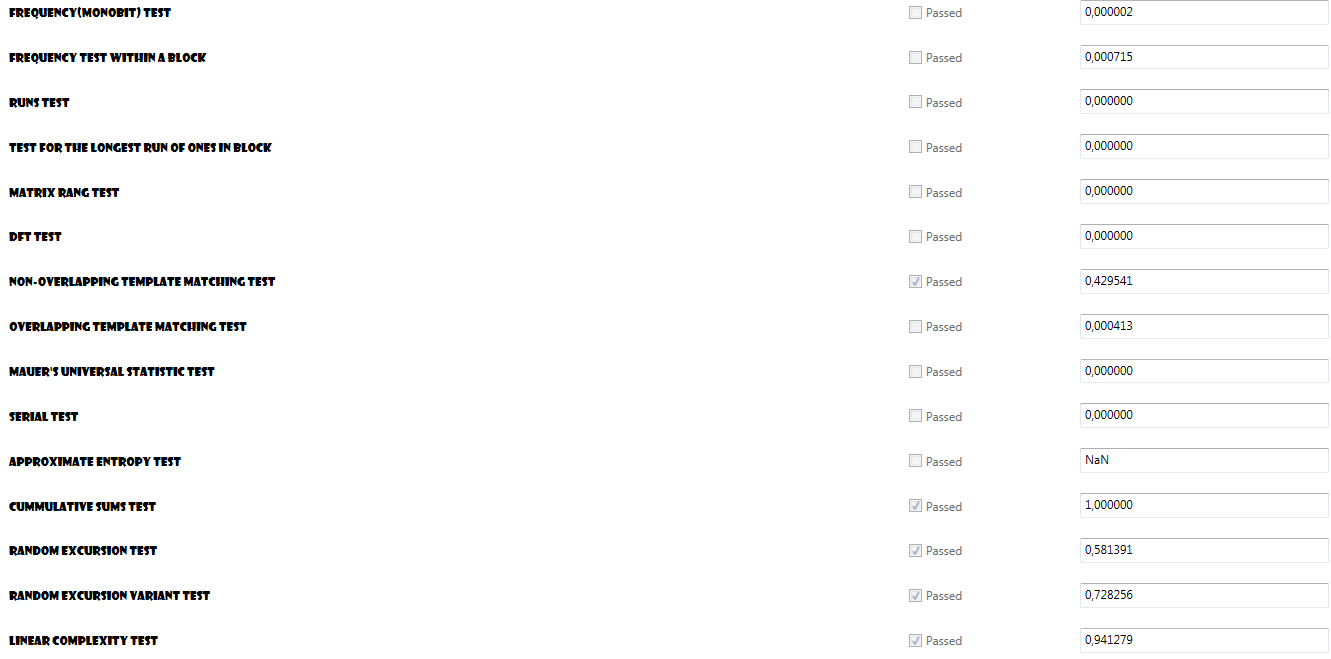


Рис.10.Результати NIST-тестування шифру Віженера

Успішно пройдено 5 тестів, однак не пройдено перший, тому подальше тестування не було потрібне(вже після цього можна зробити висновок про те, що досліджувана послідовність навіть не близько до випадкової).

## 2.4. Мережа Фейстеля

#### 2.4.1. Опис

Мережа Фейстеля - це симетрична структура, що використовується для побудови блокових шифрів. Мережа складається з комірок, що називаються комірки Фейстеля. На вхід кожній комірці надходять дані та ключ. На виході кожної комірки отримують змінені дані та змінений ключ. Всі комірки однотипні, а мережа представляє собою детерміновану структуру з багатократним повторенням. Ключ вибирається в залежності від алгоритму шифрування/розшифрування та змінюється при переході від однієї комірки до іншої. При шифруванні і розшифруванні виконуються одні і ті ж операції, відрізняється лише порядок ключів.

Інформація розбивається на блоки однакової(фіксованої) довжини. Отримані блоки називаються вхідними. У випадку, якщо довжина вхідного блоку менша, ніж розмір, який вибраний алгоритм шифрування здатен зашифрувати одночасно(розмір блоку).Як правило, довжина блоку є степенем двійки.

Шифрування одного блоку здійснюється так:

- вибраний блок ділиться на два підблоки однакового розміру – лівий(*L0*) та правий (*R0*)

- лівий підблок *L0* змінюється функцією *f* з використанням раундового ключа K0

*x=f (L0, K0)*

- результат складується по модулю 2(XOR) з правим підблоком *R0*

*x=x⊕R0*

- результат буде використаний в наступному раунді в ролі лівого підблоку *L1*

*L1=x*

- лівий підблок *L0* поточного раунду буде використаний в наступному раунді в якості правого *R0*

*R1=L0*

- згідно якогось математичного правила вираховується раундовий ключ *K1*, що буде використаний в наступному раунді

Перераховані вище операції виконуються *N-1* раз, де *N* – кількість раундів у вибраному алгоритмі шифрування. При цьому між переходами від одного раунду до іншого змінюються ключі:*K0* заміняється на *K1,K1* – на *K2* і т.д.

Розшифрування інформації відбувається там само, як і шифрування, за винятком того, що ключі ідуть у зворотному порядку.

#### 2.4.2. Математичне представлення

Нехай:

* *X* – вхідний блок даних
* *A* - деяке інволютивне перетворення – взаємно-однозначне перетворення, що є оберненим до себе самого, тобто, для кожного *X* справедливим є вираз:

*AAX = A2X=X*

* *Y* – вихідний блок даних

При однократному застосуванні перетворення *А* до вхідного блоку *Х* отримуємо вихідний блок *Y*:

*Y=AX*.

При застосуванні перетворення *А* до результату попереднього перетворення – *Y* отримаємо :

*AY=AAX=X*

Нехай вхідний блок *Х* складається з двох підблоків *L* і *R* рівної довжини:

*X= (L, R)*

Визначимо два перетворення:

* *G(X,K)* – шифрування даних *Х* з ключем *К*:

*G(X,K)=G((L,R),K)=(L⊕F(K,R),R).*

* *T(L,R)* – перестановка підблоків *L* і *R*:

*T (L, R) =(R, L).*

Введемо позначення :

* Однократне застосування перетворення *G*:

*X`= (L`, R`) =GX;*

* Двократне застосування перетворення *G*:

*X``= (L``, R``) = G2X*.

Доведемо інволютивність двократного перетворення G(G2).Нескладно зауважити, що перетворення G міняє тільки лівий підблок L, залишаючи правий R без змін:

*L`=L⊕F(K,R);*

*R`=R.*

Тому далі будемо розглядати тільки підблок L .Після двократного застосування перетворення G до L отримаємо:

*L``=L`⊕F(K,R`)=L`⊕F(K,R)=L⊕F(K,R) ⊕F(K,R)=L.*

Таким чином:

*G2L=L;*

*G2R=R.*

Тому:

*G2X=X*

і *G* – інволюція.

Доведемо інволютивність двократного перетворення T(T2)

T2X=T2 (L, R) =T(R, L) = (L, R) =X.

Розглянемо процес шифрування. Нехай:

* *Х* – вхідне значення;
* *Gi* – перетворення за ключем *Ki*
* *Yi* – вихідне значення, результат i-го раунду.

Тоді перетворення, що виконується на *i+1* раунді можна записати у вигляді :

*Yi+1=TGiYi*

Перетворення, що виконується у першому раунді:

*Y1= TG1X.*

Тобто, вихідне значення після m раундів шифрування буде дорівнювати:

*Ym=TGmYm-1=TGmTGm-1…TG1X* (12)

Можна зауважити, що на останньому етапі не обов’язково виконувати перестановку *T*.

Розшифрування ведеться застосуванням всіх перетворень в зворотному порядку. Через інволютивність кожного із перетворень зворотній порядок дає початкове значення:

*X=G1TG2T…Gm-1TGmT (Ym) =G1TG2T…Gm-1T (Ym-1) =…G1T (Y1) =X* (13)

#### 2.4.3. Переваги та недоліки

Мережа Фейстеля складається з певного фіксованого числа циклів, яке визначається з міркувань стійкості шифру, що розробляється. При цьому в останньому циклі переставляння місцями половин блоків даних не виконується, бо це не впливає на стійкість шифру. Така структура шифрів має низку переваг, а саме :

* процедури шифрування й розшифрування співпадають, лише базова інформація при розшифруванні використовується в зворотному порядку;
* при реалізації дешифрування можна використовувати ті ж блоки, що й при реалізації шифрування.

Недоліком мережі Фейстеля є те, що в кожному циклі змінюється лише половина блоку тексту, яка обробляється. Це приводить до необхідності збільшувати число циклів для досягнення бажаної стійкості.

#### 2.4.4. Результати NIST-тестування

**Frequency(monobit) test**

(input) *n =* 1000

(processing) *Sn =* -48

(processing)

(output)

(conclusion) Since *P-value* ≥ *0.01*, accept the sequence as random.;

Тест пройдено успішно.

**Frequency test (within a block)**

(input) *n = 1000*

(input) *M = 425*

(processing) *N = 20*

(processing)

(output)

(conclusion) Since *P-value* ≥ *0.01*, accept the sequence as random

Тест пройдено успішно.

**Runs test**

(input) *n* = 1000

(input)

(processing) π = 0,476

(processing)

(output)

(conclusion) Since *P-value* ≥ *0.01*, accept the sequence as random.

Тест пройдено успішно

**Test for the Longest Run of Ones in a Block**

(input) n = 6272 , М=128 ,К=5, N=49

(processing)v0 = 5 ; v1 = 13 ; v2 = 14 ; v3 = 8 ; v4 = 3 ; v5 = 6 ;

(processing)

(output)

(conclusion) Since the *P-value* is ≥ 0.01, accept the sequence as random

Тест пройдено успішно

**Matrix rank test**

(input) *n* = 79872, *M=Q* = 32

(processing) *N* = 78

(processing) *FM* = 78, *FM-1* = 0, N – FM – *FM-1*= 0

(processing)

(output)

(conclusion) Since *P-value* <=*0.01*, accept the sequence as nonrandom.

Тест не пройдено

**DFT test**

(input)n = 12288

(processing) *N1* = 6083

(processing)

(processing)

(output)

(conclusion) Since *P-value* <=*0.01*, accept the sequence as nonrandom.

Тест не пройдено

**Non-overlaping template matching test Test**

(input) m=6, n = 12288,M=100

(processing)

(conclusion) Since the *P-value* is ≥ 0.01, accept the sequence as random

Тест пройдено успішно

**Overlaping template matching test Test**

(input) m=6, n = 12288,M=100

(processing)

(conclusion) Since *P-value* <=*0.01*, accept the sequence as nonrandom.

Тест не пройдено

**Maurer’s “Universal Statistical” Test**

(input) *n* = 475392, *L=* 6, *Q=* 640,m=6

(processing)

*sum*= 399482

(output)

(conclusion) Since *P-value* <=*0.01*, accept the sequence as nonrandom.

Тест не пройдено

**Linear Complexity Test**

(input) *n = 1244000, M = 1000 ,m=6*

(processing) *v0 = 17; v1 = 49; v2 = 189; v3 = 803; v4 = 190; v5 = 119; v6 =3 0*

(processing)

(output)

(conclusion) Since the *P-value* ≥ *0.01*, accept the sequence as random.

Тест пройдено успішно.

**Serial Test**

(input) *m = 6; n = 8500*

(processing)

*;*

*;*

(processing)

(output)

(conclusion) Since both *P-value1* and *P-value2* were <=*0.01*, accept the sequence~~s~~ as non-random for both tests.

Тест не пройдено.

**Approximate Entropy Test**

(input) *m = 6; n* = 8500

(processing) *ApEn(m) = 0.676*

(processing)

(output)

(conclusion) Since *P-value* <=*0.01*, accept the sequence as nonrandom.

Тест не пройдено

**Cumulative Sums (Cusum) Test**

(input) *n* = 1000

(input) *mode* = 0 (forward) || *mode =* 1 (reverse)

(processing) *z* = 359(forward) || *z=* 1329 (reverse)

(output)

(conclusion) Since *P-value > 0.01*, accept the sequence as random.

Тест пройдено успішно.

**Random Excursions Test**

(input) *n = 1244400*

(processing) *J = 49*

Табл.1.Результати тесту “Random excursions”

|  |  |  |  |
| --- | --- | --- | --- |
| **State=x** | χ2 | **P-value** | **Conclusion** |
| **-4** | 13,71 | 0,017 | random |
| **-3** | 3,73 | 0,58 | random |
| **-2** | 22,305 | 0,0004 | non-random |
| **-1** | 2,18 | 0,82 | random |
| **+1** | 3,56 | 0,61 | random |
| **+2** | 38,75 | 0 | non-random |
| **+3** | 15,4 | 0,008 | non-random |
| **+4** | 6,52 | 0,258 | random |

Неможливо дати однозначної відповіді,потібно провести додаткові дослідження

**Random Excursions Variant Test**

(input) *n =1244400*

(processing) *J = 49*

Табл.2.Результати тесту “Random excursions variant”

|  |  |  |  |
| --- | --- | --- | --- |
| **State(x)** | **Counts** | **P-value** | **Conclusion** |
| -9 | 40 | 0,825 | random |
| -8 | 55 | 0,875 | random |
| -7 | 62 | 0,715 | random |
| -6 | 64 | 0,64 | random |
| -5 | 54 | 0,86 | random |
| -4 | 57 | 0,76 | random |
| -3 | 64 | 0,49 | random |
| -2 | 58 | 0,59 | random |
| -1 | 54 | 0,61 | random |
| +1 | 43 | 0,54 | random |
| +2 | 34 | 0,38 | random |
| +3 | 27 | 0,32 | random |
| +4 | 26 | 0,37 | random |
| +5 | 19 | 0,31 | random |
| +6 | 10 | 0,23 | random |
| +7 | 8 | 0,25 | random |
| +8 | 9 | 0,29 | random |
| +9 | 8 | 0,31 | random |

Тест пройдено успішно.

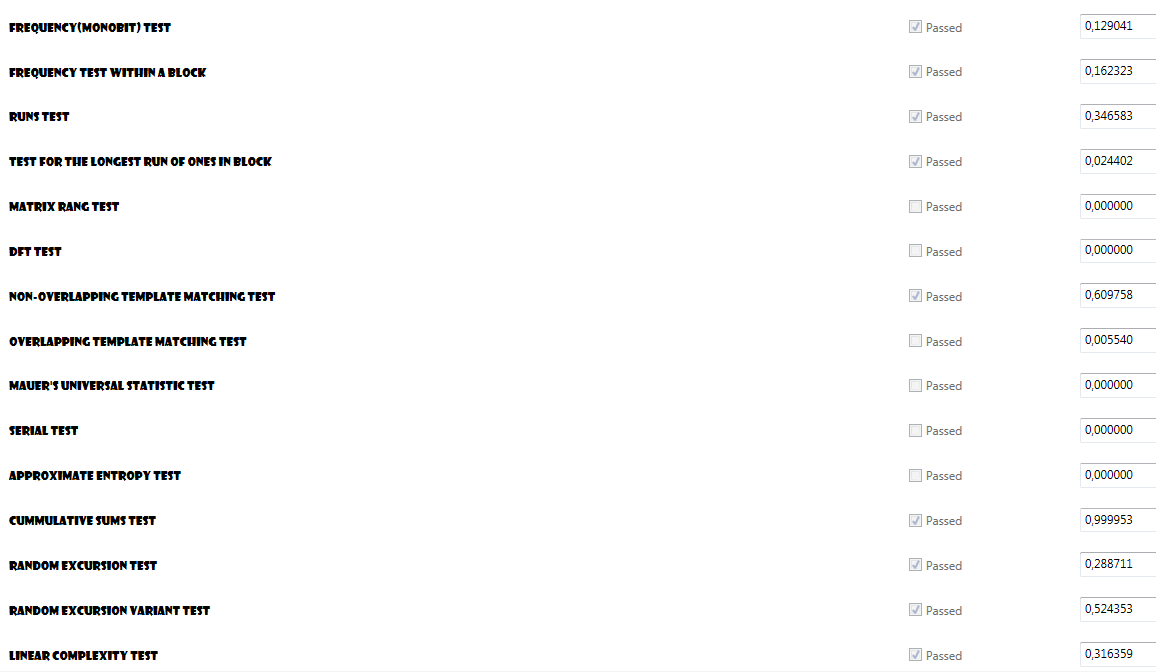


Рис.11.Результати NIST-тестування мережі Фейстеля

Успішно пройдено 9 тестів. Частотні характеристики шифру досить непогані.

## 2.5. Шифр Хілла

#### 2.5.1. Опис

Шифр Хілла – це поліграмний шифр підстановок, що базується на лінійній алгебрі та модульній арифметиці. Це був перший шифр, що на практиці дозволив одночасно оперувати більш ніж з трьома символами. Шифр Хілла не знайшов практичного застосування в криптографії через слабку стійкість до злому.

Шифр Хілла є поліграмним шифром, який може використовувати великі блоки за допомогою лінійної алгебри. У поліграмних шифрух підстановки літери відкритого тексту замінюються не по одній, а групами. Перша перевага такого способу полягає в тому, що розподіл частот груп букв значно рівномірніший, ніж окремих символів. По-друге для продуктивного частотного аналізу потрібно більший розмір зашифрованого тексту.

Кожній букві алфавіту зіставляється число по модулю n, де n – довжина алфавіту. Для латинського алфавіту часто використовується найпростіша схема: A = 0, B = 1, ..., Z = 25, але це не є істотною властивістю шифру. Блок з m букв розглядається як m-мірний вектор і множиться на m × m матрицю по модулю 26. Якщо в якості основи модуля використовується число більше ніж 26, то можна використовувати іншу числову схему для зіставлення буквах чисел і додати прогалини і знаки пунктуації . Елементи матриці є ключем. Матриця повинна бути оборотна , щоб була можлива операція розшифрування.

Стандартна схема отримання інверсної матриці *К-1*:

1.Знаходимо детермінант матриці.

2.Знаходимо обернений до детермінанта по модулю *n* елемент(розширений алгоритм Евкліда)

3.Знаходимо матрицю мінорів для матриці-ключа шифрування.

4.Транспонуємо отриману матрицю.

5.Формуємо матрицю, отриману з остач від ділення елементів на *n*

6.Множимо отриману матрицю на обернений по модулю елемент

7. Формуємо інверсну матрицю, отриману з остач від ділення елементів на *n*

#### 2.5.2. Математичне представлення

Для n=3 система може бути описана так

(14)

Або в матричній формі:

(15)

Чи :

*C=KP (mod 26)* (16)

де *P* і *С* – вектори-стовпці висотою3,що представляють собою відкритий і зашифрований текст відповідно. *К* – матриця 3 х 3,що представляє собою ключ шифрування.

Шифрування:

*C=E (K, P) = KP (mod 26)* (17)

Розшифрування:

*P=D (K, C) =K-1C (mod 26)* (18)

#### 2.5.3. Переваги та недоліки

Переваги:

* розподіл частот груп букв значно рівномірніший, ніж окремих символів.
* для продуктивного частотного аналізу потрібен більший розмір зашифрованого тексту.

Недоліки:

* складність підбору матриці, для якої існує обернена
* вразливий до атаки по обраному відкритому тексту, оскільки повністю лінійний

#### 2.5.4. Результати NIST-тестування

**Frequency(monobit) test**

(input) *n =* 1000

(processing) *Sn =* -34

(processing)

(output)

(conclusion) Since *P-value* ≥ *0.01*, accept the sequence as random.;

Тест пройдено успішно.

**Frequency test (within a block)**

(input) *n = 1000*

(input) *M = 425*

(processing) *N = 20*

(processing)

(output)

(conclusion) Since *P-value* ≥ *0.01*, accept the sequence as random

Тест пройдено успішно.

**Runs test**

(input) *n* = 1000

(input)

(processing) π = 0,483

(processing)

(output)

(conclusion) Since *P-value* ≥ *0.01*, accept the sequence as random.

Тест пройдено успішно

**Test for the Longest Run of Ones in a Block**

(input) n = 6272 , М=128 ,К=5, N=49

(processing)v0 = 5 ; v1 = 13 ; v2 = 14 ; v3 = 8 ; v4 = 3 ; v5 = 6 ;

(processing)

(output)

(conclusion) Since the *P-value* is ≥ 0.01, accept the sequence as random

Тест пройдено успішно

**Matrix rank test**

(input) *n* = 79872, *M=Q* = 32

(processing) *N* = 78

(processing) *FM* = 78, *FM-1* = 0, N – FM – *FM-1*= 0

(processing)

(output)

(conclusion) Since *P-value* <=*0.01*, accept the sequence as nonrandom.

Тест не пройдено

**DFT test**

(input)n = 12288

(processing) *N1* = 6074

(processing)

(processing)

(output)

(conclusion) Since *P-value* <=*0.01*, accept the sequence as nonrandom.

Тест не пройдено

**Non-overlaping template matching test**

(input) m=6, n = 12288,M=100

(processing)

(conclusion) Since the *P-value* is ≥ 0.01, accept the sequence as random

Тест пройдено успішно

**Overlaping template matching test**

(input) m=6, n = 12288,M=100

(processing)

(conclusion) Since *P-value* <=*0.01*, accept the sequence as nonrandom.

Тест не пройдено

**Maurer’s “Universal Statistical” Test**

(input) *n* = 475392, *L=* 6, *Q=* 640,m=6

(processing)

*sum*= 413004

(output)

(conclusion) Since *P-value* ≥ *0.01*, accept the sequence as random.

Тест пройдено успішно

**Linear Complexity Test**

(input) *n = 1244000, M = 1000 ,m=6*

(processing) *v0 = 17; v1 = 49; v2 = 189; v3 = 803; v4 = 190; v5 = 119; v6 =3 0*

(processing)

(output)

(conclusion) Since the *P-value* ≥ *0.01*, accept the sequence as random.

Тест пройдено успішно.

**Serial Test**

(input) *m = 6; n = 8500*

(processing)

*;*

*;*

(processing)

(output)

(conclusion) Since both *P-value1* and *P-value2* were ≥ *0.01*, accept the sequence~~s~~ as random for both tests.

Тест пройдено успішно.

**Approximate Entropy Test**

(input) *m = 6; n* = 8500

(processing) *ApEn(m) = 0.689*

(processing)

(output)

(conclusion) Since *P-value* <=*0.01*, accept the sequence as nonrandom.

Тест не пройдено

**Cumulative Sums (Cusum) Test**

(input) *n* = 1000

(input) *mode* = 0 (forward) || *mode =* 1 (reverse)

(processing) *z* = 218(forward) || *z=* 1312 (reverse)

(output)

(conclusion) Since *P-value > 0.01*, accept the sequence as random.

Тест пройдено успішно.

**Random Excursions Test**

(input) *n = 1244400*

(processing) *J = 1346*

Табл.3.Результати тесту “Random excursions”

|  |  |  |  |
| --- | --- | --- | --- |
| **State=x** | χ2 | **P-value** | **Conclusion** |
| **-4** | - | 0 | non-random |
| **-3** | - | 0 | non-random |
| **-2** | - | 0 | non-random |
| **-1** | - | 0 | non-random |
| **+1** | - | 0 | non-random |
| **+2** | - | 0 | non-random |
| **+3** | - | 0 | non-random |
| **+4** | - | 0 | non-random |

Тест не пройдено.

**Random Excursions Variant Test**

(input) *n =1244400*

(processing) *J = 1346*

Табл.4.Результати тесту “Random excursions variant”

|  |  |  |  |
| --- | --- | --- | --- |
| **State(x)** | **Counts** | **P-value** | **Conclusion** |
| -9 | - | 0 | non-random |
| -8 | - | 0 | non-random |
| -7 | - | 0 | non-random |
| -6 | - | 0 | non-random |
| -5 | - | 0 | non-random |
| -4 | - | 0 | non-random |
| -3 | - | 0 | non-random |
| -2 | - | 0 | non-random |
| -1 | - | 0 | non-random |
| +1 | - | 0 | non-random |
| +2 | - | 0 | non-random |
| +3 | - | 0 | non-random |
| +4 | - | 0 | non-random |
| +5 | - | 0 | non-random |
| +6 | - | 0 | non-random |
| +7 | - | 0 | non-random |
| +8 | - | 0 | non-random |
| +9 | - | 0 | non-random |

Тест пройдено успішно.

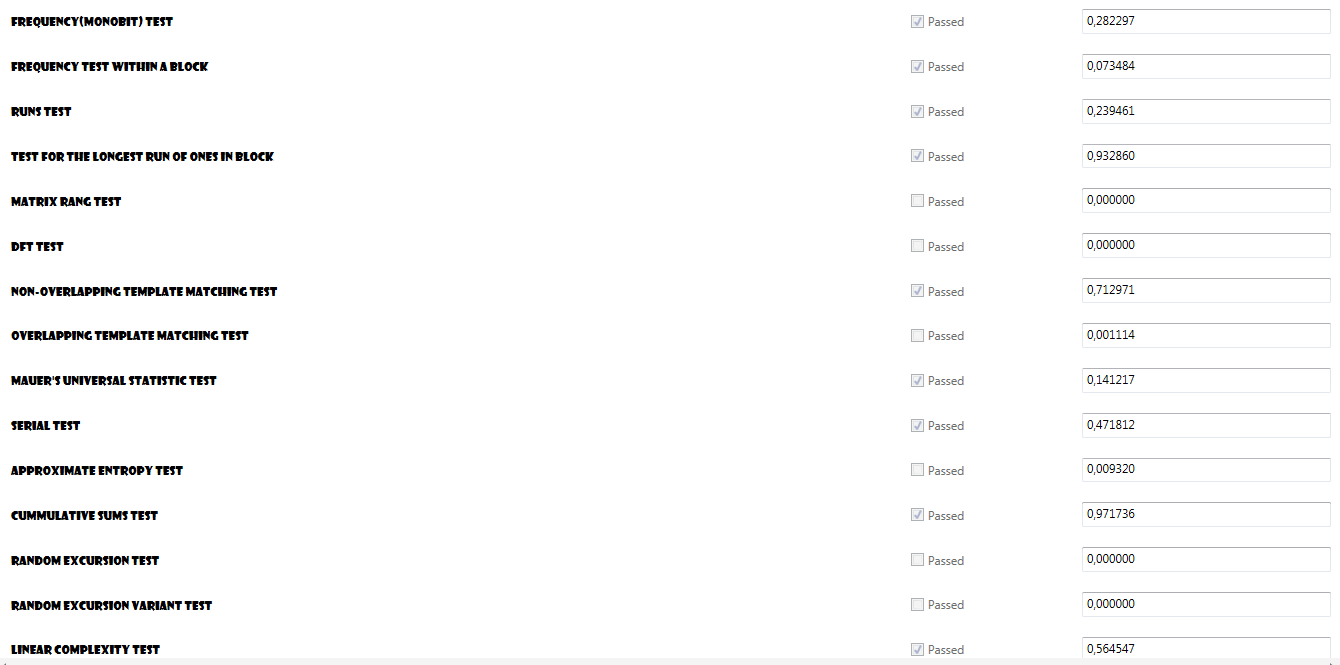


Рис.12.Результати NIST-тестування для шифру Хілла

Успішно пройдено 9 тестів з 15-ти.Частотні характеристики шифру досить непогані.

## 2.6. Розробка шифру на основі матричних перетворень

#### 2.6.1. Опис

Головна ідея полягає, в тому, щоб розробити шифр, який буде достатньо надійним ,стійким та водночас простий у реалізації і такий, що не потребуватиме багато ресурсів.

За основу для проектування я обрав шифр Хілла саме через те, що серед блокових шифрів він один із найслабших та найпростіших. Я хочу довести, що при належному підході практично будь-який шифр можна зробити достатньо надійним. Для досягнення поставленої мети я вирішив нівелювати слабкості стандартного шифру Хілла за допомогою того, шо шифруванню буде підлягати не весь текст та додавання надлишкової інформації до тексту.

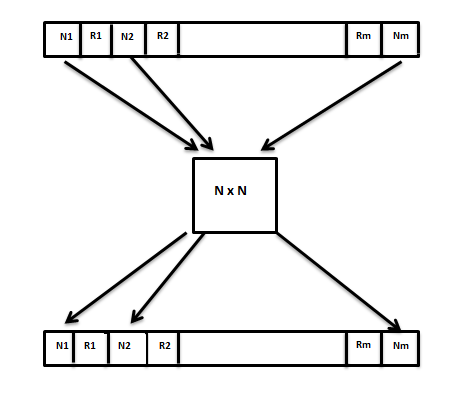


Рис.13.Схема функціонування розробленого алгоритму

Блок *N x N* – матриця-ключ.*Ni* – підблок повідомлення розміром *N*(подається на обробку), *Ri* – підблок повідомлення рандомної довжини(не подається на обробку) .Повідомлення розбивається на множину підблоків *N* та *R*(в такій послідовності). Далі підблоки *N* передаються на обробку, а довжини підблоків *R*(що кожного разу рандомно генеруються на заданому інтервалі) почергово записуються у масив. Результуюче повідомлення – це спотворений певним чином масив довжин підблоків, що не оброблялися та оброблені підблоки *N* і необроблені *R*.Процес повторюється декілька разів(бажано хоча б двічі).

#### 2.6.2. Приклад роботи шифру

Матриця ключ:



Шифруємо текст : «MAHEROVSKYANDRIY» =



\*=

Далі генератор псевдорандомних чисел продукує число 3,отже,підблок,що складається з наступних 3-х символів не обробляється.

\*=

Далі генератор псевдорандомних чисел продукує число 2,отже,підблок,що складається з наступних 2-х символів не обробляється. Довжина залишку повідомлення менша за довжину ключа, тому доповнюємо повідомлення.

\*=

Результат першого етапу шифрування:



Групи підкреслених символів – це підблоки рандомної довжини, що не обробляються. Таких блоків у даному прикладі два: довжиною в 3 символи та в 2 символи.

Результат другого етапу шифрування:



Примітка: перший символ *n* означає кількість рандомних блоків, що не піддаються обробці. Наступних n символів означають довжини послідовних необроблюваних блоків(спотворені певним чином – у даному випадку вибиралися такі числа, щоб поділивши на максимальне можливе значення довжини рандомного блоку, можна було отримати справжні довжини підблоків).

Зашифрований текст: «C1WR7]=ROVA(40ND,;=[ »

Процес повторюється певну визначену кількість разів(бажано хоча б двічі). На кожному раунді бажано використовувати іншу матрицю-ключ.

Розшифровуємо текст : «C1WR7]=ROVA(40ND,;=[ » =



Обернена матриця:



З шифрованого рядка зчитуємо кількість необроблюваних підблоків = 2,далі довжини кожного з них = {3,2}(отримані завдяки зворотньому алгоритму спотворення).

Результат першого етапу розшифрування:



\*=

Пропускаємо 3 символи.

\*=

Пропускаємо 2 символи.

\*=

Відкидаємо зайвий останній символ

Результат другого етапу розшифрування:



Розшифрований текст : «MAHEROVSKYANDRIY»

#### ***2.6.3. Переваги та недоліки***

Шифр зберігає переваги стандартного шифру Хілла:

- шифр розмиває частотні характеристики тексту;

- для продуктивного криптоаналізу потрібен текст більший, ніж для неблокових шифрів.

Водночас володіє деякими оригінальними:

- результати виконання шифрування однакових даних за однаковими ключами відмінні(криптоаналітику буде надзвичайно важко виявити якусь систему, закономірність, залежність в шифрі) .Відмінними навіть будуть довжини шифрованого тексту.

- методи для злому шифру Хілла не працюють з даним шифром. Навіть повний перебір ключів не дасть потрібного результату ,оскільки шифрується лише частина повідомлення. Тобто застосовуючи ключ для розшифрування будуть спотворюватися необроблені блоки.

- наявність в шифрованому тексті не лише безпосередніх результатів шифрування відкритого тексту, що створює хибне враження про частотні характеристики тексту та його довжину.

Недоліки:

- важкість знаходження в якості ключа матриць, для яких можливо обчислити обернені.

#### 2.6.4. Результати NIST-тестування

**Frequency(monobit) test**

(input) *n =* 1000

(processing) *Sn =* -52

(processing)

(output)

(conclusion) Since *P-value* ≥ *0.01*, accept the sequence as random.;

Тест пройдено успішно.

**Frequency test (within a block)**

(input) *n = 1000*

(input) *M = 425*

(processing) *N = 20*

(processing)

(output)

(conclusion) Since *P-value* ≥ *0.01*, accept the sequence as random

Тест пройдено успішно.

**Runs test**

(input) *n* = 1000

(input)

(processing) π = 0,474

(processing)

(output)

(conclusion) Since *P-value* ≥ *0.01*, accept the sequence as random.

Тест пройдено успішно

**Test for the Longest Run of Ones in a Block**

(input) n = 6272 , М=128 ,К=5, N=49

(processing)v0 = 5 ; v1 = 13 ; v2 = 14 ; v3 = 8 ; v4 = 3 ; v5 = 6 ;

(processing)

(output)

(conclusion) Since *P-value* <=*0.01*, accept the sequence as nonrandom.

Тест не пройдено

**Matrix rank test**

(input) *n* = 79872, *M=Q* = 32

(processing) *N* = 78

(processing) *FM* = 78, *FM-1* = 0, N – FM – *FM-1*= 0

(processing)

(output)

(conclusion) Since *P-value* <=*0.01*, accept the sequence as nonrandom.

Тест не пройдено

**DFT test**

(input)n = 6498

(processing) *N1* = 3223

(processing)

(processing)

(output)

(conclusion) Since *P-value* <=*0.01*, accept the sequence as nonrandom.

Тест не пройдено

**Non-overlaping template matching test**

(input) m=6, n = 6400,M=100

(processing)

(conclusion) Since the *P-value* is ≥ 0.01, accept the sequence as random

Тест пройдено успішно

**Overlaping template matching test**

(input) m=6, n = 6400,M=100

(processing)

(conclusion) Since the *P-value* is ≥ 0.01, accept the sequence as random

Тест пройдено успішно

**Maurer’s “Universal Statistical” Test**

(input) *n* = 475392, *L=* 6, *Q=* 640,m=6

(processing)

*sum*= 413004

(output)

(conclusion) Since *P-value* <=*0.01*, accept the sequence as nonrandom.

Тест не пройдено

**Linear Complexity Test**

(input) *n = 1244000, M = 1000 ,m=6*

(processing) *v0 = 19; v1 = 60; v2 = 188; v3 = 795; v4 = 400; v5 = 101; v6 =34*

(processing)

(output)

(conclusion) Since the *P-value* ≥ *0.01*, accept the sequence as random.

Тест пройдено успішно.

**Serial Test**

(input) *m = 6; n = 8500*

(processing)

*;*

*;*

(processing)

(output)

(conclusion) Since both *P-value1* and *P-value2* were ≥ *0.01*, accept the sequence~~s~~ as random for both tests.

Тест пройдено успішно.

**Approximate Entropy Test**

(input) *m = 6; n* = 8500

(processing) *ApEn(m) = 0.684*

(processing)

(output)

(conclusion) Since *P-value* <=*0.01*, accept the sequence as nonrandom.

Тест не пройдено

**Cumulative Sums (Cusum) Test**

(input) *n* = 1000

(input) *mode* = 0 (forward) || *mode =* 1 (reverse)

(processing) *z* = -4004(forward) || *z=* 567 (reverse)

(output)

(conclusion) Since *P-value > 0.01*, accept the sequence as random.

Тест пройдено успішно.

**Random Excursions Test**

(input) *n = 1244400*

(processing) *J = 1*

Табл.5.Результати тесту “Random excursions”

|  |  |  |  |
| --- | --- | --- | --- |
| **State=x** | χ2 | **P-value** | **Conclusion** |
| **-4** | 63,10 | 0 | non-random |
| **-3** | 34,97 | 0 | non-random |
| **-2** | 15,55 | 0,008 | non-random |
| **-1** | 2,99 | 0,7 | random |
| **+1** | 0,99 | 0,96 | random |
| **+2** | 0,88 | 0,97 | random |
| **+3** | 0,2 | 0,99 | random |
| **+4** | 0,14 | 0,99 | random |

Неможливо зробити однозначний висновок.Необхідні подальші дослідження.

Random Excursions Variant Test

(input) *n =1244400*

(processing) *J = 1*

Табл.6.Результати тесту “Random excursions”

|  |  |  |  |
| --- | --- | --- | --- |
| **State(x)** | **Counts** | **P-value** | **Conclusion** |
| -9 | 1 | 1 | random |
| -8 | 1 | 1 | random |
| -7 | 1 | 1 | random |
| -6 | 1 | 1 | random |
| -5 | 1 | 1 | random |
| -4 | 1 | 1 | random |
| -3 | 1 | 1 | random |
| -2 | 1 | 1 | random |
| -1 | 1 | 1 | random |
| +1 | 0 | 0,47 | random |
| +2 | 0 | 0,68 | random |
| +3 | 0 | 0,75 | random |
| +4 | 0 | 0,78 | random |
| +5 | 0 | 0,81 | random |
| +6 | 0 | 0,83 | random |
| +7 | 0 | 0,84 | random |
| +8 | 0 | 0,85 | random |
| +9 | 0 | 0,86 | random |

Тест пройдено успішно.

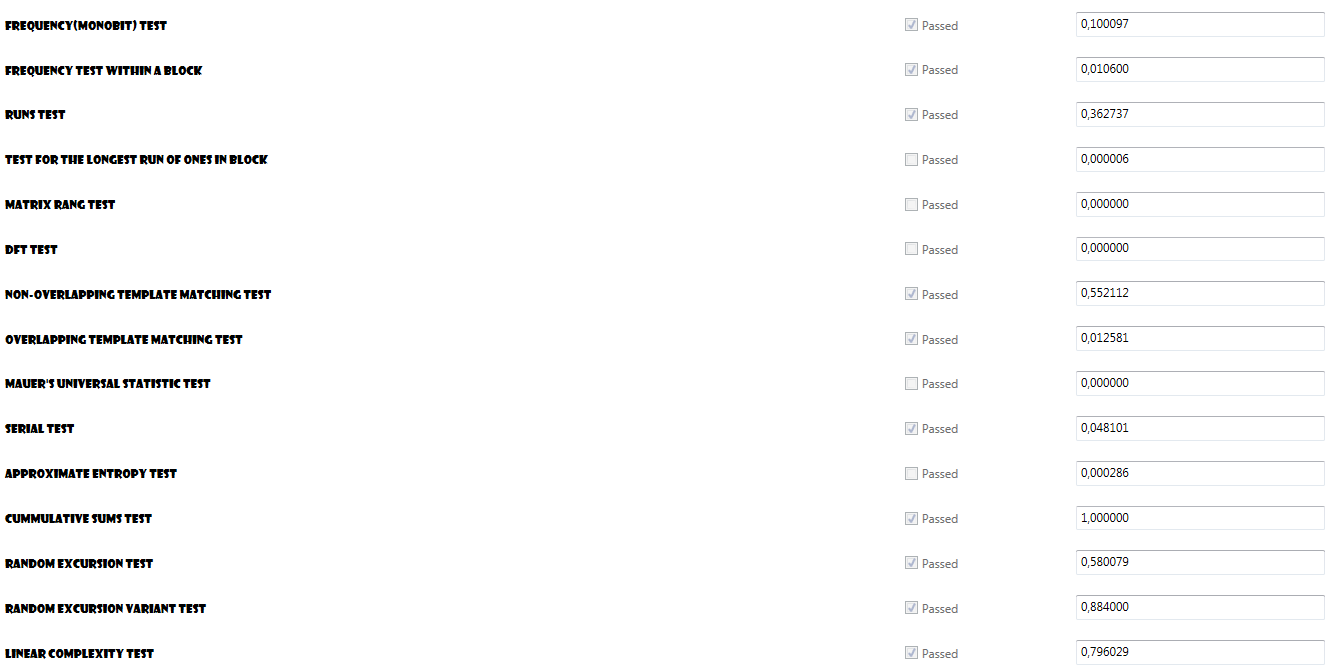


Рис.14.Результати NIST тестування.

Успішно пройдено 10 тестів з 15-ти.Частотні характеристики досить хороші.

## 2.7. Висновки до розділу 2

В другому розділі бакалаврської дипломної роботи здійснений детальний аналіз блокових шифрів, а саме шифрів Хілла, Віженера і Фейстеля. Зокрема, для кожного із перелічених, представлений детальний опис процесу шифрування і дешифрування, математичне представлення шифрів а також наведені їх основні переваги та недоліки.

Жоден із вищеперелічених шифрів не має стійкості, достатньої для використанні при захисті інформації в фінансових установах. Тому був розроблений новий шифр, що базується на стандартному шифрі Хілла, який володіє такими перевагами, що роблять його набагато стійкішим ніж розглянуті раніше шифри. В подальшому цей шифр буде використаний для створення елементів системи захисту інформації.

# РОЗДІЛ 3.КРИПТОГРАФІЧНІ ХЕШ-ФУНКЦІЇ

Для фінансових установ існує необхідність збереження конфіденційності даних на досить тривалий час(наприклад, паролі користувачів), забезпечення перевірки важливих документів на справжність(електронний цифровий підпис).Використання шифрів може бути недостатньо для забезпечення такого рівня захисту.

## 3.1. Хеш-функції. Хешування

Хешування - перетворення вхідного масиву даних довільної довжини в вихідний бітову рядок фіксованої довжини, за певним алгоритмом. Таке перетворення також називається хеш-функція ( функція згортки), а його результат - хеш, хеш-код, хеш-сума ( зведення повідомлення).

Застосування хешування:

- Побудова асоціативних масивів;

- Пошук дублікатів в серіях наборів даних;

- Побудова досить унікальних ідентифікаторів для наборів даних;

- Контрольне підсумовування даних - для подальшого виявлення в них помилок (випадкових або навмисних), що виникають при зберіганні і (або) передачі сигналу;

- Зберігання паролів в системах захисту (в хешованому вигляді, - для відновлення з якого потрібна обробка функцією, яка є «зворотною» по відношенню до використаної хеш-функції);

- При виробленні електронного підпису (на практиці, - часто, - підписується не саме повідомлення, а його хеш-образ).

У загальному випадку (принцип Діріхле) - немає однозначної відповідності між вихідними даними і хеш-кодом, тому що значення хеш-функцій менш різноманітні, ніж значення вхідного масиву: різний зміст може дати однаковий хеш-код - що може привести до так званої колізії. Імовірність виникнення колізій грає важливу роль в оцінці якості хеш-функцій.

Як відомо, криптографічні хеш-суми відрізняються від звичайних хеш-сум тим, що крім основних властивостей, що вимагаються від будь-якої хеш-функції:

- здатності перетворення вхідного значення (зазвичай тексту) довільної довжини в вихідне значення фіксованої довжини,

- статистичної рівномірності випадаючих вихідних значень,

- хорошим «розкидом» (розбіжністю приблизно в половині біт) вихідних значень навіть для текстів, різниця між якими дуже незначна (можливо тільки в 1 біті);

до криптографічних хеш-алгоритмів пред'являються додаткові вимоги:

- розрядність вихідних значень повинна знаходитися далеко за межами можливостей повного перебору на сучасній техніці як за швидкістю обробки, так і за обсягом зберігання (на практиці це - розрядність в 128, 160, 256 і більше біт);

- не повинно існувати способу (істотно більш ефективного, ніж повний перебір вхідних значень) обчислити будь-яку пару вхідних текстів, що дають на виході однакове хеш-значення (неважливо яке) - успішна атака на цю вимогу називається «колізією» хеш-функції;

- не повинно існувати способу (істотно більш ефективного, ніж повний перебір вхідних значень) за значенням хеш-функції підібрати будь-який вхідний текст, що дає на виході алгоритму це хеш-значення - успішна атака на цю вимогу називається «оборотністю» хеш-функції.

Три перерахованих вище вимоги до крипостійкості хеш-функцій перетворюють їх за фактом в ідентифікатори фіксованої довжини для текстів, файлів і взагалі довільних блоків інформації. При цьому ці ідентифікатори:

- унікальні на всьому просторі сучасного інформаційного суспільства;

- незворотні, тобто не розкривають абсолютно нічого про зміст вихідного документа.

Це дозволяє використовувати хеш-функції в задачах:

- Перевірки цілісності файлів, архівів, збірок тощо (Досить довіреністю передати одержувачу хеш-суму файлу і тоді будь-які несанкціоновані зміни в файлі негайно змінять його контрольну хеш-суму);

- Зберігання паролів у незворотному вигляді (в сховище поміщаються не самі паролі, а хеш-суми від рядка, наприклад, виду «константа (соль)» + «пароль», що дозволяє при розкритті хешу зберегти власне словникову парольну фразу в секреті);

- Електронно-цифрового підпису документів (власне ЕЦП встановлюється завжди не на сам файл, що по-перше, було б дуже повільно, і по-друге, несло б деякі проблеми з безпекою самого ключа підпису, а на його хеш-суму, забезпечуючи такий ж рівень захисту від модифікацій).

На сьогоднішній день переважну частку застосувань хеш-функцій «беруть на себе» алгоритми MD5, SHA-1, SHA-256. Звичайно, існує і безліч інших менш відомих, або поширених тільки у вузьких співтовариствах алгоритмів (наприклад, RIPEMD, TIGER, PANAMA і ін.)

У криптографії сіль – це випадкова послідовність, що використовується в якості додаткових вхідних даних для хешування. Основна функція солі –захист від атак проти списку вихідних паролів за допомогою райдужних таблиць чи одного проходу повного перебору.

Нова сіль генерується випадковим чином для кожного пароля. У загальному випадку, сіль і пароль з’єднуються та обробляються за допомогою криптографічної хеш-функції ,і результуюча вихідна послідовність(але не оригінальний пароль) зберігається разом із сіллю в базі даних. Криптографічні солі широко використовуються в багатьох сучасних комп’ютерних системах від системи облікових даних Unix до інтернет-безпеки.

Райдужні таблиці представляють собою застосування більш раннього і простішого алгоритму Мартіна Хеллмана. Райдужна таблиця – це попередньо обчислена таблиця для реверсу криптографічної хеш-функції ,як правило ,для злому хеш-паролів. Таблиці здебільшого використовують для відновлення оригінального паролю певної довжини, що складається з обмеженого набору символів. Це практичний приклад time-memory tradeoff - розумного компромісу між часом пошуку по таблиці і займаною пам'яттю (використовується менше часу для комп’ютерної обробки інформації і більше пам’яті, ніж при brute-force атаці, але більше часу обробки і менше пам’яті, ніж при використанні таблиць пошуку).Використання ключа ,виведеного функцією з використанням солі робить такий вид атаки практично нездійсненним.

У криптографії, лавинний ефект відноситься до бажаної властивості криптографічних алгоритмів, зазвичай блокових шифрів та криптографічних хеш-функцій. Лавинний ефект очевидний у випадку, коли трохи змінена вхідна послідовність (наприклад, перевернутий один біт) спричиняє суттєві зміни вихідного сигналу (наприклад, половина вихідних біт обернулися). У випадку високоякісних блокових шифрів, така невелика зміна в будь-якому ключі або відкритому тексту повинна викликати різку зміну шифротексту. Сам термін був вперше використаний Хорстом Фейстелем , хоча концепція бере початок принаймні у дифузії Шеннона.

Якщо блокові шифри або криптографічна хеш-функція не проявляє лавинний ефект в значній мірі, то він має погану рандомізацію, і, таким чином, криптоаналітик може робити припущення про вхідний текст, якщо буде мати тільки вихідний. Це може бути достатньо, щоб частково або повністю зламати алгоритм. Таким чином, лавинний ефект є бажаною умовою з точки зору конструктора криптографічного алгоритму або пристрою.

При побудові шифру або хеш-функції значний прояв лавинного ефекту є однією з цілей первинного проектування. Це дозволяє невеликим змінам швидко поширюватися через ітерації алгоритму, таким чином, що кожен біт на виході повинен залежати від кожного біта входу до зупинки алгоритму

Криптографічна хеш-функція - будь-яка хеш-функція, яка є крипостійкою, тобто задовольняє ряд вимог, специфічних для криптографічних додатків.

Більшість криптографічних хеш-функцій приймають на виході рядок будь-якої довжини ,а на виході дають хеш-значення фіксованої довжини. Криптографічна хеш-функція повинна бути здатною витримувати всі відомі типи криптоаналітичних атак. Як мінімум вона мусить мати наступні властивості:

* Стійкість до відновлення прообразу(Для даного хеш-значення *h* має бути складно знайти повідомлення *m*,таке ,що *h=hash(m).*Це поняття пов’язане з поняттям односторонньої функції. Функції, які не володіють цією властивістю можуть вразливі до атак відновлення прообразу. Слід зауважити, що на даний момент немає використовуваних хеш-функцій з доведеною односпрямованістю. )
* Стійкість до відновлення другого прообразу(Для даного вхідного значення *m1* має бути неможливо обчислювально знайти відмінне значення *m2* таке, щоб *hash(m1)==hash(m2)*.Функції, що не володіють цією рисою є вразливими до атаки відновлення другого прообразу )
* Стійкість до колізій(Повинно бути неможливо обчислювально знайти два відмінних повідомлення *m1* і *m2* таких, щоб *hash(m1)=hash(m2).*Така пара називається криптографічною хеш-колізією. Ця властивість іноді називають сильна колізійна стійкість. Вона вимагає, що хеш-значення має бути хоча б у 2 рази довше, ніж вимагає стійкість до повного перебору ; в іншому випадку колізії можуть бути знайдені за допомогою атаки «днів народження». Атака «днів народження» - вид криптографічної атаки, що використовує парадокс днів народження для злому шифрів чи пошуку колізій хеш-функцій. Зазвичай саме знаходження способу побудови колізій криптоаналітиками служить першим сигналом старіння алгоритму і необхідності його швидкої заміни.)

Дані вимоги не є незалежними:

- Оборотна функція нестійка до відновлення другого прообразу і колізій.

- Функція, нестійка до відновлення другого прообразу, нестійка до колізій; зворотні твердження невірні.

Слід зазначити, що досі ще не доведене існування незворотних хеш-функцій, для яких обчислення будь-якого прообразу заданого значення хеш-функції теоретично неможливо. Зазвичай знаходження зворотного значення є лише обчислювально складним завданням.

Наявність цих властивостей означає, що зловмисний противник не зможе замінити чи модифікувати вхідні дані без зміни хеш-значення. Тому, якщо два повідомлення мають однаковий хеш-код , то з високим ступенем вірогідності можна стверджувати, що вони ідентичні.

Функції, що відповідають цим критеріям все ж можуть мати інші небажані властивості. На теперішній час популярні хеш-функції можуть бути вразливими до атак довжини-розширення(коли дано значення *hash(m)* та довжина *len(m),*вибираючи підходящі m` зловмисник може обчислити *hash(m ||* *m`*),де *||* означає конкатенацію ).Ця слабкість може бути використана для злому наївних схем автентифікації на основі хеш-функцій.

В ідеалі можна побажати ще сильніших умов. Для противника має бути неможливо знайти два повідомлення з істотно схожими значеннями хеш-коду; або вивести яку-небудь корисну інформацію про дані, з огляду лише на їх хеш-значення. Тому криптографічна хеш-функція повинна поводитися ,наскільки це можливо, як рандомна функція, зберігаючи при цьому детермінованість та ефективність обчислення.

Алгоритми контрольних сум, такі як CRC32 та інші циклічні перевірки надмірності призначенні для задоволення набагато слабших вимог і ,як правило, непридатні в якості криптографічних хеш-функцій. Наприклад CRC був використаний для перевірки цілісності повідомлень в стандарті шифрування WEP,але легко був відкритий тип атак, що використовував лінійність контрольних сум проти цього стандарту.

## 3.2. MD4

#### 3.2.1. Опис

MD4 – криптографічна хеш-функція, що була розроблена професором Массачусетського університету Рональдом Рівестом в 1990 році та вперше описана RFC 1186.Для вхідного повідомлення функція генерує 128-розрядний хеш-код, що називається дайджестом повідомлення.

Алгоритм хешування:

1.Додавання бітів, що не вистачає

2.Додавання довжини повідомлення

3.Ініціалізація MD-буфера

4.Обробка повідомлення блоками по 16 слів

5.Формування хешу

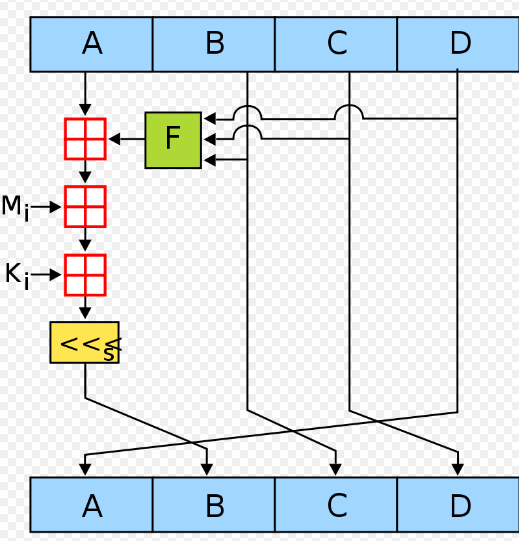


Рис.15.Одна операція MD4

Хешування з MD4 складається з 48 таких операцій, згрупованих в 3 раунда по 16 операцій. – нелінійна функція ;в кожному раунді функція міняється. *Mi* означає 32-бітний блок вхідного повідомлення, а *Ki* – 32-бітна константа, відмінна для кожної операції.

128-бітні MD4 хеші представляють собою 32-х значне число в 16-ковому форматі.

Рівень безпеки, що закладався в MD4 був розрахований на створення достатньо стійких гібридних систем електронного цифрового підпису, заснованих на MD4 та криптосистемі з відкритим ключем. Рональд Рівест вважав, що алгоритм хешування MD4 можна використовувати і в системах, які потребують сильної криптостійкості. Але водночас він відзначав, що MD4 створювався перш за все як дуже швидкий алгоритм хешування, через це він може не відповідати високим вимогам криптостійкості. Як показали подальші дослідження, він був правий, і для систем, де важливою є перш за все криптостійкість почали використовувати алгоритм MD5.

Алгоритм допускає ефективну програмну реалізацію на 32-розрядному процесорі. Він не використовує складних структур даних та підпрограм. Алгоритм оптимізований з точки зору його реалізації на мікропроцесорах типу Intel.

#### 3.2.2. Стійкість алгоритму

Після того, як алгоритм був вперше опублікований, Берт ден Бур та Антон Босселер побудували колізії для останніх двох із трьох раундів, що використовуються в MD4.Біхам опублікував роботу, в якій виклав можливі варіанти застосування методу диференційного аналізу для побудови колізій від повного MD4.Незважаючи на те, що жоден із запропонованих методів побудови колізій не приводить до успіху(тобто, вони виявляються менш ефективними ніж метод грубої сили) для повного MD4,Рівест посилив алгоритм і запропонував нову схему хешування MD5.

Перша колізія була знайдена Гансом Доббертіном в 1996 році.

## 3.3. S**HA-1**

#### 3.3.1. Опис

SHA-1 – алгоритм криптографічного хешування, що описаний в RFC 3174.Для вхідного повідомлення довільної довжини алгоритм генерує 160-бітне хеш-значення, що називається дайджестом повідомлення. Використовується в багатьох криптографічних програмах та протоколах.

Вхідне повідомлення розбивається на блоки по 512 біт в кожному. Останній блок доповнюється до довжини кратної 512 біт. Спочатку добавляється 1(біт),а потім нулі, щоб довжина блока стала рівною (512-64=448) біт. В решту 64 біта записується довжина вхідного повідомлення в бітах(little-endian формат).Якщо останній блок має довжину більшу за 448,але меншу за 512 біт, то доповнення виконується наступним чином: спочатку добавляється 1(біт),далі нулі до кінця 512-бітного блоку; після цього створюється ще один 512-бітний блок, який заповнюється до 448 біт нулями, після чого в решту 64 біти записується довжина вхідного повідомлення в бітах(little-endian формат).Доповнення останнього блоку здійснюється завжди, навіть якщо повідомлення вже має потрібну довжину.

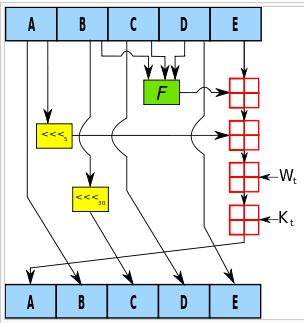


Рис.16.Одна ітерація алгоритму SHA-1

A,B,C,D,E - 32-бітні змінні, *Kt* – константа, *F* – нелінійна операція, *Wt* – 80 32-бітних слів(після обробки оригінальних 16 32-бітових *Mt*)

#### 3.3.2. Стійкість алгоритму

В січні 2005 року Вінсент Ріджмен та Елізабет Освальд опублікували повідомлення про атаку на урізану версію SHA-1(53 раунди замість 80),котра дозволяє знаходити колізії менш, ніж за 280 операцій.

В лютому 2005 року Сяоюнь Ван,Іцюнь Ліза Інь і Хунбо Юй представили атаку на повноцінний SHA-1,яка вимагає менш, ніж 269 операцій. Пізніше Крістоф де Каньєр та Крістіан Рехберг представили вдосконалену версію атаки на SHA-1,за що отримали нагороду за найкращу статтю на конференції ASIACRYPT 2006.Ними була представлена двоблокова колізія на 64-раундовий алгоритм з обчислювальною складністю близько 235 операцій.

Хоча SHA-1 вважається зламаним, на практиці подібний злом займе близько 5-ти мільярдів років. Бурт Калінскі, голова дослідницького відділу в «лабораторії RSA» передбачає, що перша атака по знаходженню прообразу буде успішно здійснена в найближчі 5-10 років.

## 3.4. Розробка власної криптографічної хеш-функції

#### 3.4.1. Опис

Головна ідея полягає, в тому, щоб розробити хеш-функцію, яка буде достатньо надійна ,стійка та водночас проста у реалізації і така, що не потребуватиме багато ресурсів.

За основу для проектування я обрав шифр Хілла. Я взяв довжину блока в 32 символи(достатньо, щоб зробити брут-форс атаку практично нездійсненною).Також за допомогою встановлення зв’язків між підблоками забезпечив необхідний лавинний ефект. А особливості самого шифру забезпечують підходящі частотні характеристики для тексту.

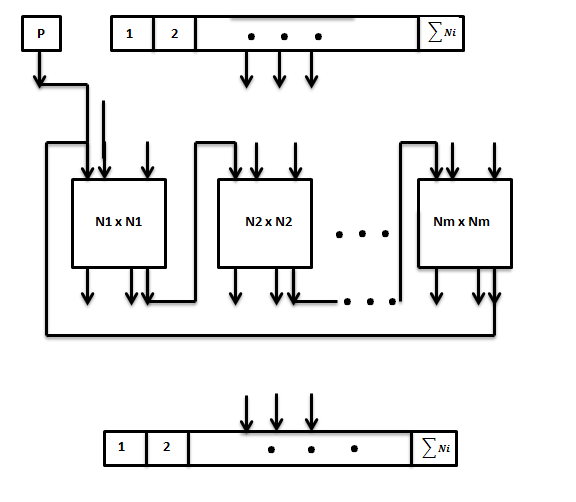


Рис.17.Схема функціонування алгоритму

Блок *Ni x Ni* - одна з матриць-ключів.*P* – символ ,що подається на вхід незалежно від даних. Вхідний блок повідомлення доповнюється до потрібної довжини *∑Ni-m* (якщо потрібно).Далі він дробиться на підблоки довжиною *Ni -1* і подається на обробку. Відбувається множення рядка(перший символ – так званий ,вхідний, решта – відповідний підблок) на відповідну матрицю-ключ з масиву матриць. На виході отримуємо рядок(останній символ – вихідний, решта результуючий підблок).Як можна побачити із рис.15,окремо ми подаємо лише перший вхідний символ, а далі вхідний символ підблоку – це вихідний символ попереднього підблоку.З вихідних рядків беремо результуючі підблоки та даємо їх на вихід алгоритму(або на наступне коло обробки).Кількість циклів обробки попередньо вказуємо.

Оскільки криптографічна хеш-функція повинна продукувати значення сталого розміру, необхідно застосовувати стиснення. Стиснення відбувається за дещо спрощеною схемою Міагучі-Пренеля:

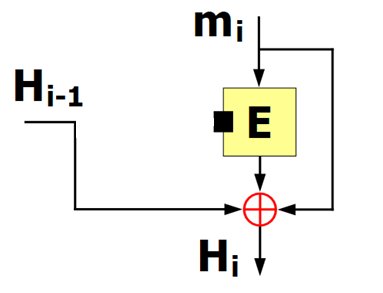


Рис.18.Спрощена схема Міагучі-Пренеля

Одностороння функція стиснення Міагучі-Пренеля є розширеним варіантом аналогічної функції Матіаса-Мейєра-Осеаса. Це була функція, незалежно запропонована Бартом Пренелем і Сьодзо Міагучі.

Функція подає кожен блок вихідного повідомлення (*mi*) в якості тексту, який буде зашифрований. Далі виконується операція XOR зашифрованого тексту з тим же блоком вхідного повідомлення (*mi*), а потім також операція XOR з попереднім значенням хеш-функції (*Hi-1*) для отримання наступного значення хешу (*Hi*).

Попереднє значення хеш-функції (*Hi-1*) подають як ключ для блочного шифру. У першому раунді, коли немає попереднього значення хеш, він використовує постійне заздалегідь обумовлений початкове значення (*H0*).

Якщо блоковий шифр має різні розміри блоку і ключа, тоді значення хеш-функції (*Hi-1*) буде мати неправильний розмір для використання в якості ключа. Хеш-функція подається на функцію *g* для перетворення, щоб відповідати як ключ для шифрування.

Дана схема не призначення для ключів-матриць. Слідуючи їй було б потрібно кожного проходу використовувати функцію, що перетворювала б ключ-рядок в ключ-множину матриць. Це спричиняє зайві труднощі та затрати ,тому в моєму варіанті ключ залишається незмінним та є множиною матриць різних розмірів.

Математично це може бути представлено так:

(19)

#### 3.4.2. Приклад роботи хеш-функції

Блоки – складові ключа:

М1=;М2=;М3=;М4=

P = 0, к-ть циклів – 1,ключ –{М2,M1,M4,M3,M2,M3,M4,M1,M3}

Вхідне повідомлення: «MAHEROVSKYANDRIY» =



Повідомлення доповнюється до необхідного розміру блоку:





\* =

\*=

\*=

\*=

\* =

\*=

\*=

\*=

\*=

Примітка: якщо було б задано більше, ніж один раунд, останній символ останнього результуючого підблоку подавався би на вхід першого у наступному колі.

Результат:





= «,6+,9>}.%N].}.MRE52B7,@>I#6}(ES=»

#### 3.4.3. Стійкість розробленого алгоритму

1.Стійкість для відновлення образів.

Довжина одного блоку хеш-коду – 32 , символів, доступних для формування послідовності – 64.Тобто,можливих значень – 6432 ≈ 6.3 \*1057.Припустимо,що можна перебирати значення зі швидкістю 1012 штук за секунду. Тоді для повного перебору даної кількості значень буде необхідно 6.3\*1045секунд = 175\*1040 годин = 20\*1037 років. Оскільки алгоритм новий, досі ще не існує ефективніших способів, ніж повний перебір для знаходження потрібного значення. Тобто хеш-функція досить добре протистоїть атакам відновлення образів.

2.Стійкість до колізій.

Найпоширеніша атака проти цієї властивості – атака днів народження. Одним із наслідків парадоксу днів народження є те, що для n-бітового блокового шифру повторювані появи блоку шифротексту можуть очікуватися з вірогідністю близько 0,63 при наявності лише 2n/2 випадкових відкритих текстів, зашифрованих на одному ключі (незалежно від розміру ключа).Тобто, щоб з високою ймовірністю можна було знайти колізії, необхідно перебрати 2n/2 варіантів. У даному випадку n =192,тому потрібно перебрати 286 можливих комбінацій ≈7.7 \*1025. Припустимо, що можна перебирати значення зі швидкістю 1012 штук за секунду. Тоді для повного перебору даної кількості значень буде необхідно 7.7\*1013секунд = 213\*108 годин = 24\*105 років. Тобто хеш-функція досить добре протистоїть колізійним атакам.

3.Лавинний ефект.

Результати виконання хешування даних, що мінімально відрізняються одне від одного.

Табл.7.Результати виконання хешування

|  |  |  |
| --- | --- | --- |
| Вхідне значення | Хеш-значення | К-ть змінених бітів |
| MAHEROVSKYA | #<$\_P$.2G!XZ\_Q>YGCD=1[MY,PD0I3}U | 0 із 192 |
| MAHEROVSKYB | "|!Z<U1Z1N$ST H"05N]%-6J;$AE[]LS | 101 із 192 |
| MAHEROVSKYC | })N\*3)7Q=.LL94UG>IX,DRA>-T\_8O\*8Q | 85 із 192 |
| MAHEROVSKYD | {(.RN {HX95E!}1;U[1MUE ['\*>M$?FO | 103 із 192 |
| MAHEROVSKYE | ]\*9{)\_^?%)!\_V|}&}O[=5)&,0X)#UH2M | 68 із 192 |
| MAHEROVSKYF | [&)J10<\*NFZ([C+LI$\*](@ON1<&U)Q!K | 102 із 192 |
| MAHEROVSKYG | @^F5LD!]8T^"BMI22U!,H;8?2.$| Z,I | 92 із 192 |
| MAHEROVSKYH | 9%TB\*6F3D1N8XWV<=)JMYQC"3\_},\_2<G | 106 із 192 |
| MAHEROVSKYI | 8$1;'JL -"71{02QW T=9D,04'[A0[WE | 52 із 192 |
| MAHEROVSKYJ | 7#"<J]RR<\_B D@"7#\_;]>((R5B84D&^C | 108 із 192 |
| MAHEROVSKYK | 6"\_V^PXIUL.TZ&=?K07,L9QC635I6\_QA | 86 із 192 |
| MAHEROVSKYL | 5}L$;%;!"Z\*M"?JV4D$M,,@^7F2]JG{? | 92 із 192 |
| MAHEROVSKYM | 4{ZNHV3(K7PFFIW]?6+={PE487'Q]PK= | 72 із 192 |
| MAHEROVSKYN | 3]79$|9{5\*9?.S3DYJF]?C-V9J,&PY7> | 100 із 192 |
| MAHEROVSKYO | 2[\*F.."4ADD)$,# %]P,P\*|G@[ZY%1E| | 85 із 192 |
| MAHEROVSKYP | 1@D1F?\*..R;#H6\_$MPZM08S)[NW=V@1( | 98 із 192 |
| MAHEROVSKYQ | 09R\_"1+S(')9;#KI6%3=$.]8]"T0|^?& | 38 із 192 |
| MAHEROVSKYR | '8'ZZEBJR{R2^>X'AV{]COGZ{RQE.=.% | 100 із 192 |
| MAHEROVSKYS | -7{\*D7HA]+[.JE4( |),TB0K}^N8?F|# | 89 із 192 |
| MAHEROVSKYT | ;6+R{KN)HJFU'O$N&.BM4&>+"VKM1OV} | 100 із 192 |
| MAHEROVSKYU | ,5J{X{T}2X'N\*Y?4O?L=\*7U]#)H#EX%] | 66 із 192 |
| MAHEROVSKYV | .4XJBQZ5\_5<GL2L+81V]G };$ZEU70P@ | 102 із 192 |
| MAHEROVSKYW | 355[^',Y^T!1]YSCE',XNIO%+B|K9]8 | 76 із 192 |
| MAHEROVSKYX | Z2^BVW5T^B{|)(59,79M8A2!^;?,{%J6 | 108 із 192 |
| MAHEROVSKYY | Y1B;!<[KOPH$NA%A(K^=<^=#&!+AQ+64 | 54 із 192 |
| MAHEROVSKYZ | X0P<9,$B9;1@3K!XQ{\_]K6W1\*1|4^ED2 | 112 із 192 |
| MAHEROVSKY | W';VT!)|E[+3<UM}@QH,.Z#S(D\*IWN00 | 88 із 192 |
| MAHEROVSKY. | V-[$\_2\_"'<V,P-ZFE^RM]MKD)5%]<W\_- | 90 із 192 |
| MAHEROVSKY, | U;<N7FD6>H"V586,-W.=\_!4&|H"Q,' , | 70 із 192 |
| MAHEROVSKY; | T,H9R8J;VVJO+%^^|<5]O%?5<9]&!8) | 108 із 192 |
| MAHEROVSKY- | S.VF+LPU#33HR=AKS,",'5YW>L9Y2$UY | 92 із 192 |
| MAHEROVSKY' | R 315}VLL$\_A7GN1]!<M#Y%H+{6=F>$W | 105 із 192 |
| MAHEROVSKY0 | QZ$\_PR.C6!X<\_Q |G2D=BLM|=P308DOU | 17 із 192 |
| MAHEROVSKY1 | PY!Z<&1<BN$%T 7P0FN]S?69\_$0ELM[S | 100 із 192 |
| MAHEROVSKY2 | OXN\*3X7#,.L[94&6>8X,3$A ?T;8}VIQ | 88 із 192 |
| MAHEROVSKY3 | NW.RN>{7)954!}B\_UL1M&4 L!\* MR-5O | 102 із 192 |
| MAHEROVSKY4 | MV9{);^-S)!;V|OU}}[=FX&=AXX#&7CM | 73 із 192 |
| MAHEROVSKY5 | LU)J1A<V{FZW[C.[IR\*]WKO{B<UUX#'K | 107 із 192 |
| MAHEROVSKY6 | KTF5L3!MIT^PBM8C2&!,7\_8-C.R|><=I | 85 із 192 |
| MAHEROVSKY7 | JSTB\*GFD31NIXW\*Z=XJM|#CPD\_O,;CZG | 103 із 192 |
| MAHEROVSKY8 | IR1;'9L>?"7B{0C#W>T=J3,AE'LAAL(E | 51 із 192 |
| MAHEROVSKY9 | HQ"<JMR$Z\_B>D@PH#;;] W($FBI43UTC | 113 із 192 |
| MAHEROVSKY@ | GP\_V^"X8&L.^Z&,-KA7,[JQ2G3FIG;#A | 85 із 192 |
| MAHEROVSKY[ | FOL$;S;'PZ\*]"?9\*43$M==@THFC]96N? | 93 із 192 |
| MAHEROVSKY] | ENZNH\*3W@7P5FI(M?G+=N"EEI7!QM"@= | 69 із 192 |
| MAHEROVSKY{ | DM79$Y9NF\*9-.SD3Y9F]-2-\*JJ=&"|H> | 107 із 192 |
| MAHEROVSKY} | CL\*F.+"E0DDX$,Q>%MP,"V|6K[<YSB4| | 88 із 192 |
| MAHEROVSKY" | BKD1F-\*++R;QH6;RM"ZMAISXLN(=\*KB( | 105 із 192 |
| MAHEROVSKY# | AJR\_"B+%W')J;#@86S3=R+]IM"^0YT-& | 37 із 192 |
| MAHEROVSKY$ | !I'ZZ4B9${RC^>)!A\*{]2}G<NR#E+,+% | 99 із 192 |
| MAHEROVSKY% | ?H{\*DHH0M+[+JEEW Y),^10@O^{8-5Y# | 86 із 192 |
| MAHEROVSKY^ | \_G+R{@NX7JF&'OR{&+BMEU>.PV@MB}\*} | 103 із 192 |
| MAHEROVSKY& | =FJ{XNTOCX'{\*Y-EO-L=VHUMQ)7#4)S] | 69 із 192 |
| MAHEROVSKY\* | +EXJB#ZF;5<6L2[.8BV]6>}\_RZ4UHA"@ | 97 із 192 |
| MAHEROVSKY( | >D55[T'=|^T'1]|%C4',){I}S+1|@JM8 | 79 із 192 |
| MAHEROVSKY) | <C^BV(5^TB{Y)(FJ,H9MI02'T;-,NS96 | 99 із 192 |
| MAHEROVSKY| | |BB;!Z[@}PHRNAS0(@^=ZT=QU!.A#.G4 | 59 із 192 |
| MAHEROVSKY< | )AP<9=$1J;1K3K')QN\_]@GWBV1Y4T432 | 103 із 192 |
| MAHEROVSKY> | (!;VT')Y4[+D<U]O@#H,+<#%WDVI({A0 | 85 із 192 |
| MAHEROVSKY+ | \*?[$\_C\_P!<V=P-<5ETRMM]K3X5S]Z(;- | 89 із 192 |
| MAHEROVSKY= | &\_<N75DG H"\*58G=-(.=;'4UYHPQ=!>, | 69 із 192 |
| MAHEROVSKY\_ | ^=H9RIJ\_\*VJ}+%TT|Z5]}S?FZ9M&'IX | 105 із 192 |
| MAHEROVSKY? | %+VF+[P&Q337R=0@S=",!FY( LJYCR&Y | 87 із 192 |
| MAHEROVSKY! | $>315OV[[$\_07G{B]'<MQ|%7.{G=5 RW | 104 із 192 |

Як видно з наведеної вище таблиці, внаслідок зміни одного біта у вхідній послідовності були спричинені значні зміни у вихідній. Тобто дана хеш-функція явно демонструє лавинний ефект.

4.Рівномірність розподілу частот.



Рис.19.Частотний розподіл символів у тексті, шифрованому хеш-функцією(без стиснення).

Для даного аналізу було використано шифрувальний алгоритм хеш-функції без стиснення(щоб можна було оцінити частотні характеристики результатів)

Визначимо середнє інтегральне відхилення за допомогою формули ...

(20)

Середнє інтегральне відхилення дорівнює 12.52 % (для порівняння – стандартний метод Хілла дає відхилення близько 30%).По рис…. можна побачити, що частотний розподіл символів достатньо близький до рівномірного, водночас процес хешування не вимагає значних ресурсів для виконання.

## 3.5. Висновки до розділу 3

В третьому розділі бакалаврської дипломної роботи здійснений детальний аналіз криптографічних хеш-функцій, а саме хеш-функції MD5 та SHA-1. Зокрема, для кожної із перерахованих, представлений детальний опис процесу хешування, а також наведені оцінки їх стійкості.

Жодна із вищеперелічених хеш-функцій не має стійкості, достатньої для використанні при захисті інформації в фінансових установах. Тому було розроблено нову криптографічну хеш-функцію, що базується на стандартному шифрі Хілла, яка володіє такими властивостями, що роблять її дуже стійкою. В подальшому ця криптографічна хеш-функція буде використана для створення елементів системи захисту інформації.

# РОЗДІЛ 4.РОЗРОБКА ЕЛЕМЕНТІВ СИСТЕМИ ЗАХИСТУ ІНФОРМАЦІЇ

Буде розроблено такі елементи системи захисту інформації :система автентифікації та система для формування/перевірки електронного цифрового підпису. Вони будуть базуватися на самостійно розроблених на основі шифру Хілла криптографічної функції та шифру. Крім безпосередньої реалізації цих елементів також буде забезпечено можливість застосування на практиці самих лише шифру та хеш-функції.

## 4.1. Система автентифікації

#### 4.1.1. Загальний принцип роботи системи автентифікації

Автентифікація — процедура встановлення належності користувачеві інформації в системі пред'явленого ним [ідентифікатора](https://uk.wikipedia.org/wiki/%D0%86%D0%B4%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D1%96%D0%BA%D0%B0%D1%82%D0%BE%D1%80).

В більшості випадків паролі не зберігаються на цільових об’єктах, а зберігається лише їх хеш-код. Зберігати парольні фрази недоцільно, тому що у випадку несанкціонованого доступу до файлу з фразами, зловмисник дістане інформацію про всі паролі та зможе ними скористатися , а при збереженні хеш-коду він здобуде лише цей код, який не може бути перетворений у вхідні дані, в даному випадку в парольну фразу. Під час процедур автентифікації обчислюється хеш-значення введеної парольної фрази і порівнюється із збереженим.

Прикладом в даному випадку служать ОС Linux та Windows.В них зберігаються лише хеш-значення парольних фраз із облікових записів користувачів.

Така система має на увазі передачу повідомлення по захищеному каналу, тобто по каналу, із якого криптоаналітик не може перехопити повідомлення чи послати своє. В іншому випадку він зможе перехопити пароль та використовувати його для подальшої нелегальної автентифікації. Захищатися від подібних атак можна за допомогою методу «виклик-відповідь». Цей метод я буду використовувати при розробці системи автентифікації.

#### 4.1.2. Принцип роботи розробленої системи автентифікації

Нехай деякий користувач з іменем *NAME* проводить автентифікацію по парольній фразі,*PASS* .У базі даних зберігається значення розробленої криптографічної хеш-функції *HASH(PASS,S),*де *S* - сіль(псевдовипадкова послідовність визначеної довжини).Клієнт посилає запит *(NAME,R),*де *R* – псевдовипадкова послідовність певної довжини. У відповідь з бази даних отримується значення *S*.Користувач вираховує значення хеш-функції *HASH(R,HASH(PASS,S))* і посилає на обробку. Обробник також обчислює значення *HASH(R,HASH(PASS,S))* та порівнює його з отриманим. Якщо значення співпадають – автентифікація вірна.

При такому способі пароль не зберігається відкрито в базі даних і ,навіть перехопивши всі повідомлення між користувачем та обробником, криптоаналітик не в змозі відновити значення паролю ,а хеш-значення, що передається, кожен раз інше.

#### 4.1.3. Програмна реалізація

При запуску системи з’являється вікно автентифікації. У відповідних полях необхідно ввести логін та пароль. Далі система згідно наведеного раніше принципу роботи здійснить перевірку введених даних. Якщо користувач з таким іменем існує в базі даних і йому відповідає саме такий пароль – запуститься головне вікно програми. Код, що реалізовує таку перевірку наведений у додатку Д.

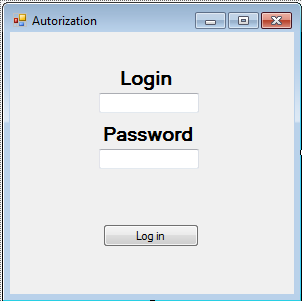


Рис.20.Вікно авторизації

У головному вікні ,у вкладці “Performance” після кліку по кнопці “CreateAccount”,з’являється вікно створення нового аккаунту. Користувач вводить дані у відповідні поля, система обробляє їх згідно наведених вище принципів роботи та записує в базу даних(код в додатку Г ).

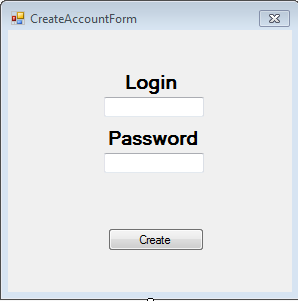


Рис.21.Вікно створення нового аккаунту

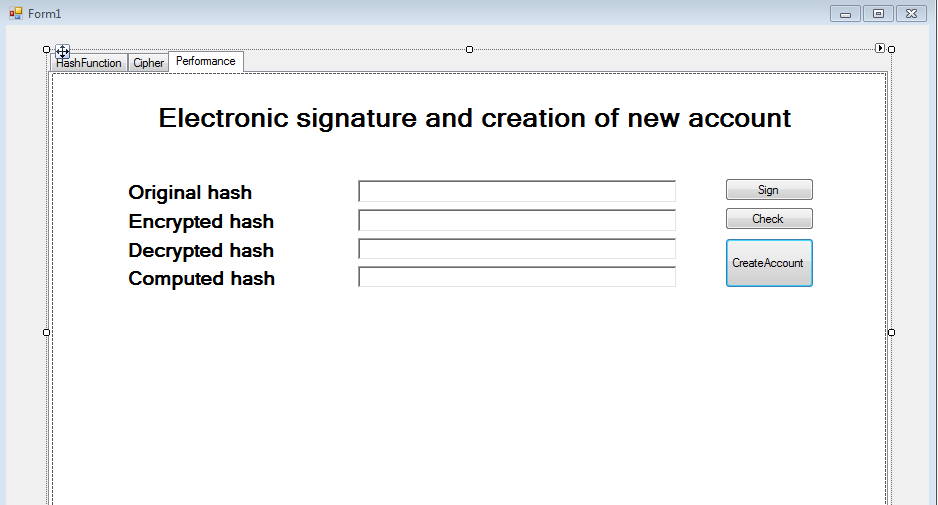


Рис.22.Вкладка “Performance”

## 4.2. Електронно-цифровий підпис

#### 4.2.1. Загальний принцип роботи електронно-цифрового підпису

Електронний цифровий підпис (ЕЦП) — вид електронного підпису, отриманого за результатом криптографічного перетворення набору електронних даних, який додається до цього набору або логічно з ним поєднується і дає змогу підтвердити його цілісність та ідентифікувати підписувача. Електронний цифровий підпис накладається за допомогою особистого(секретного) ключа та перевіряється за допомогою відкритого ключа.

Варто зауважити один дуже важливий аспект: електронний цифровий підпис не існує в готовому вигляді ніколи. Він не існує у вигляді файлу чи чого-небудь ще. Електронний цифровий підпис насправді з’являється, створюється тільки в той момент часу, коли документ підписується секретним закритим ключем. Тобто, документ не підписується електронним цифровим підписом. Це в корені неправильний вираз. Документ підписується тільки закритим ключом, а не електронним цифровим підписом тільки після підписання секретним ключем ,тільки в цей момент часу створюється електронний цифровий підпис. Загальний механізм створення такий :береться будь-який документ, будь-якої довжини, розраховується його контрольна сума(хеш значення),і це значення шифрується закритим ключом. У підсумку, отриманий шифрований рядок і є електронним цифровий підписом.

#### 4.2.2. Принцип роботи розробленої системи створення/перевірки електронного цифрового підпису

*Створення ЕЦП*

Для документа за допомогою спеціального комп’ютерного забезпечення вираховується ,так званий, дайджест повідомлення. В даному випадку це хеш-значення(може бути контрольною сумою тощо).

Дайджест характеризується такими властивостями:

1.Фіксована,чітко визначена довжина, що не залежить від розміру повідомлення

2.Відбиток повідомлення(дайджест) є унікальним для кожного повідомлення

3.По дайджесту неможливо відновити повідомлення

Таким чином, якщо документ був модифікований, то зміниться і його відбиток, що відобразиться при перевірці електронного цифрового підпису.

Далі за допомогою розробленого алгоритму шифрування відбиток документу кодується закритим ключем. Отриманий електронний цифровий підпис передається окремо разом із документом.

Розшифрувати електронний цифровий підпис і отримати початковий дайджест повідомлення можна тільки використовуючи відкритий ключ автора.

Тобто, обчислення дайджеста документа захищає його від модифікації сторонніми особами після підписання, а шифрування секретним особистим ключем підтверджує авторство документу.

*Перевірка справжності підписаного документа*

Перевірка електронного цифрового підпису документа здійснюється в три етапи:

1.Перший етап. Адресат, використовуючи необхідне програмне забезпечення та відкритий ключ автора, розшифровує підписує відбиток і отримує дайджест початкового документа.

2.Другий етап. За допомогою відповідної програми та спеціальної математичної функції з отриманого документа обчислюється його дайджест.

3.Третій етап. При перевірці електронного цифрового підпису порівнюються дайджести одержаного та початкового документів. В результаті перевірки ми можемо зробити висновок про справжність документа.

#### 4.2.3. Програмна реалізація

При натисканні кнопки “Sign” у вкладці “Performance” головного вікна програми відбувається формування електронного цифрового підпису для документа,що обирається у діалоговому вікні. При натисканні кнопки “Check” здійснюється перевірка документа на справжність за допомогою електронного цифрового підпису(код в додатку Е).

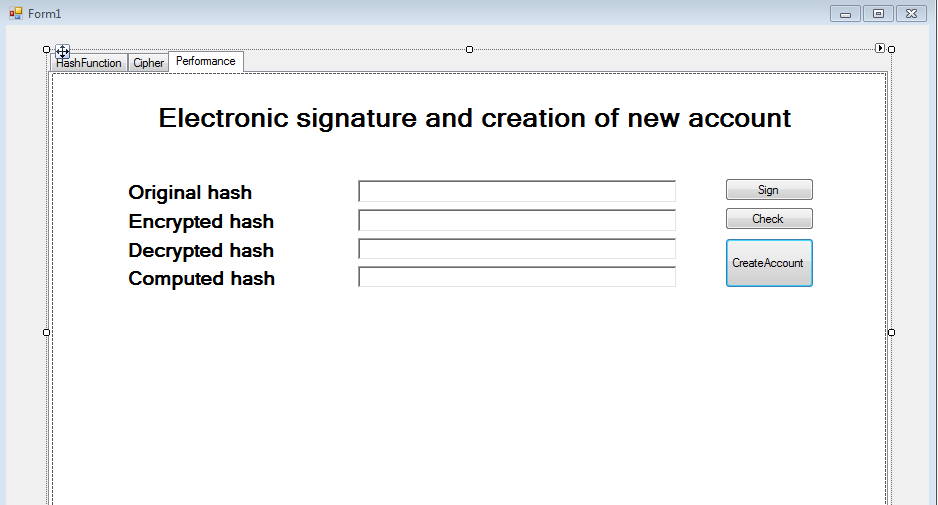


Рис.23.Вкладка “Performance” головного вікна програми

## 4.3. Демонстрація роботи шифру та хеш-функції

В головному вікні у вкладці “HashFunction” в текстове поле “Symbol” вводимо номер символу, який буде вхідним, після цього клікаємо кнопку “Ok”.Далі вводимо текст, який необхідно хешувати в текстове поле “Input” та після того, як буде натиснута кнопка “Perform” у текстове поле “Output” буде виведено результати роботи криптографічної хеш-функції(код в додатку Е).

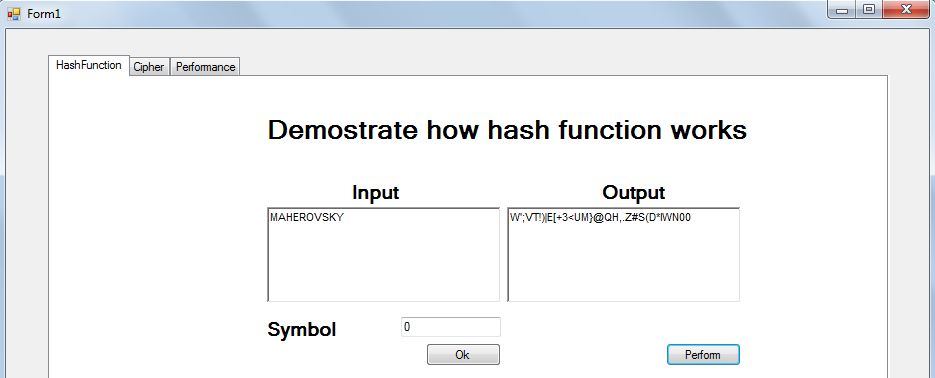


Рис.24.Результат хешування

В головному вікні у вкладці “Cipher” натискаємо на кнопку “Encrypt” і у діалоговому вікні вибираємо текстовий файл для шифрування. Щоб розшифрувати текст необхідно натиснути кнопку “Decrypt”.

У першому текстовому полі виводиться текст з оригінального файлу(кольором позначені підблоки, що не піддавалися процедурі шифрування);у другому – результат шифрування після першого кола(з виділеними необроблюваними підблоками);у третьому – поточний результат шифрування(кольором позначені підблоки, що не будуть оброблятися на наступному колі);у четвертому – фінальний результат шифрування(з виділеними необроблюваними підблоками);у п’ятому- результат виконання розшифрування(після натискання відповідної кнопки).

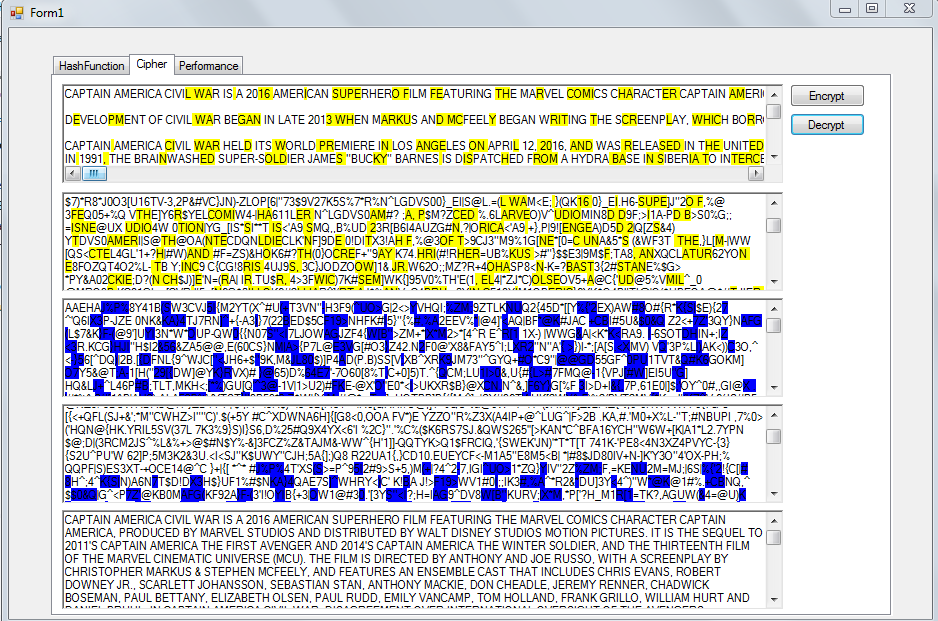


Рис.25.Результат шифрування/дешифрування

## 4.4. Висновки до розділу 4

В четвертому розділі бакалаврської дипломної роботи представлені розроблені елементи системи захисту інформації у фінансовій установі(система автентифікації та система формування/перевірки електронного цифрового підпису).

Був розроблений зручний та зрозумілий інтерфейс для демонстрації роботи створених елементів системи захисту інформації та їх складових – розроблених блокового шифру та криптографічної хеш-функції.

# РОЗДІЛ 5.ЕКОНОМІЧНА ЧАСТИНА

# ВИСНОВКИ

В бакалаврській дипломній роботі був здійснений детальний аналіз блокових шифрів, а саме шифрів Хілла, Віженера і Фейстеля. Зокрема, для кожного із перелічених, представлений детальний опис процесу шифрування і дешифрування, математичне представлення шифрів а також наведені їх основні переваги та недоліки.

Жоден із вищеперелічених шифрів не має стійкості, достатньої для використанні при захисті інформації в фінансових установах. Тому був розроблений новий шифр, що базується на стандартному шифрі Хілла, який володіє такими перевагами, що роблять його набагато стійкішим ніж розглянуті раніше шифри.

В бакалаврській дипломній роботі був здійснений детальний аналіз криптографічних хеш-функцій, а саме хеш-функції MD5 та SHA-1. Зокрема, для кожної із перерахованих, представлений детальний опис процесу хешування, а також наведені оцінки їх стійкості.

Жодна із вищеперелічених хеш-функцій не має стійкості, достатньої для використанні при захисті інформації в фінансових установах. Тому було розроблено нову криптографічну хеш-функцію, що базується на стандартному шифрі Хілла, яка володіє такими властивостями, що роблять її дуже стійкою.

В бакалаврській дипломній роботі були представлені розроблені елементи системи захисту інформації у фінансовій установі(система автентифікації та система формування/перевірки електронного цифрового підпису).

Був розроблений зручний та зрозумілий інтерфейс для демонстрації роботи створених елементів системи захисту інформації та їх складових – розроблених блокового шифру та криптографічної хеш-функції.

# СПИСОК ЛІТЕРАТУРИ

1. Магеровський А.В. Автоматизація формування та обліку експедитерської документації транспортного підприємства / Курко Ю.В., Магеровський А.В., Вівчар Д.Р. // ACIT’2016/ “Сучасні компютерні інформаційні технології”. ТНЕУ. -  Тернопіль. 20-21 травня 2016. – C. 188-190.
2. Andriy Maherovsky. Research block ciphers based on matrix // Andriy Maherovsky, Yuri Kurko, Danylo Vivchar // Perspective technologies and methods in MEMS design: Materials XII International conference MEMSTECH-2016, April 20 – 24, 2016, Polyana, Ukraine / National  University “Lviv Polytechnic”, 2016 – P. 202 – 204.
3. Бабичев С. Г., Гончаров В. В., Серов Р. Е. Основы современной крипто-графии. – Москва: Горячая линия-Телеком, 2001. – 120с.
4. Вербіцький О. В. Вступ до криптології. - Львів: Видавництво науково-технічної літератури, 1998. – 248с.
5. Гундарь К. Ю., Гундарь А. Ю., Янишевский Д. А. Защита информации в компьютерных системах. – Київ: Корнійчук, 2000. – 152с.
6. Ємець В., Мельник А. Сучасна криптографія. Основні поняття. – Львів:  2007. – 148 с.
7. Жельников В. Появление шифров. — М.:ABF, 1996. — 335 с.
8. [U.S. Patent 1 845 947](http://www.google.com/patents/US1,845,947). Лестер С. Хилл. Пристрій для шифрування. 1929.
9. Спосіб шифрування інформації. Патент України на корисну модель №99073. Бюл. № 9 від 12.05.2015. Ігнатович А.О., Іванців В. Р., Іванців Р-А. Д., Павич Н. Я.
10. Shannon C. E. Communication Theory of Secrecy Systems // [Bell System Technical Journal](https://ru.wikipedia.org/w/index.php?title=Bell_System_Technical_Journal&action=edit&redlink=1). — 1949.
11. NIST. American national standard for information systems – data format for the interchange of fingerprint, facial, and, scar mark and tattoo (smt) information, ansi-itl 1-2000 (nist special publication 500-245), September 2000.
12. Vicki Stanfield, Roderick W. Smith. Linux System Administration (Craig Hunt Linux Library). — 2. — Sybex, 2002. — 656 с
13. Еві Немет, Гарт Снайдер, Скотт Сібасс, Трент Р. Хейн. Unix, інструкція системного адміністратора./Пер с англ. – СПб.: Питер: K.Издательская группа BHV,2002. – 928 c.
14. Оглтрі Т.Елементарне введення в еліптичну криптографію:протоколи криптографії на елептичних кривих. – М.: КомКнига, 2006. – 280 с.
15. Нільс Фергюсон, Брюс Шнайер Практична криптографія. Вільямс, 2005 г., 424 с.

# ДОДАТОК А

Код класу «MathOperations»для виконання матричних операцій та перетворень на мові C#.

class MathOperations

{

//Транспонування матриці

public static T[,] TransponeMatrix<T>(T[,] input)

{

T[,] output = new T[input.GetLength(1), input.GetLength(0)];

for (int i = 0; i < output.GetLength(0); i++)

{

for (int j = 0; j < output.GetLength(1); j++)

{

output[j, i] = input[i, j];

}

}

return output;

}

//Створити меншу матрицю

public static T[,] CreateSmallerMatrix<T>(T[,] input, int i, int j)

{

int order = (int)Math.Sqrt(input.Length);

T[,] output = new T[order - 1, order - 1];

int x = 0, y = 0;

for (int m = 0; m < order; m++, x++)

{

if (m != i)

{

y = 0;

for (int n = 0; n < order; n++)

{

if (n != j)

{

output[x, y] = input[m, n];

y++;

}

}

}

else

{

x--;

}

}

return output;

}

//Знак мінора

public static int SighOfElement(int i, int j)

{

if ((i + j) % 2 == 0)

{

return 1;

}

else

return -1;

}

//Детермінант

public static double Determinant(double[,] input)

{

int order = (int)Math.Sqrt(input.Length);

if (order > 2)

{

double value = 0;

for (int j = 0; j < order; j++)

{

double[,] Temp = CreateSmallerMatrix(input, 0, j);

value += input[0, j] \* (SighOfElement(0, j) \* Determinant(Temp));

}

return value;

}

else if (order == 2)

{

return ((input[0, 0] \* input[1, 1]) - (input[1, 0] \* input[0, 1]));

}

else

{

return input[0, 0];

}

}

//Матриця мінорів

public static double[,] MinoreMatrix(double[,] input)

{

int order = (int)Math.Sqrt(input.Length);

double[,] output = new double[order, order];

for (int i = 0; i < order; i++)

{

for (int j = 0; j < order; j++)

{

double[,] Temp = CreateSmallerMatrix(input, i, j);

output[i, j] = Determinant(Temp) \* SighOfElement(i, j);

}

}

return output;

}

//Формування оберненої матриці

public static double[,] FormInverseMatrix(double[,] input, int coef, int alphabet)

{

int order = (int)Math.Sqrt(input.Length);

double[,] output = new double[order, order];

output = MinoreMatrix(input);

output = TransponeMatrix(output);

for (int i = 0; i < order; i++)

{

for (int j = 0; j < order; j++)

{

output[i, j] = output[i, j] % alphabet;

if (output[i, j] < 0)

output[i, j] += alphabet;

}

}

for (int i = 0; i < order; i++)

{

for (int j = 0; j < order; j++)

{

output[i, j] \*= coef;

output[i, j] = output[i, j] % alphabet;

}

}

return output;

}

//Знайти коеф

public static int FindCoef(int a, int m)

{

int x, y;

int g = GCD(a, m, out x, out y);

if (g != 1)

throw new ArgumentException();

return (x % m + m) % m;

}

//Розширений алгоритм Евкліда

public static int GCD(int a, int b, out int x, out int y)

{

if (a == 0)

{

x = 0;

y = 1;

return b;

}

int x1, y1;

int d = GCD(b % a, a, out x1, out y1);

x = y1 - (b / a) \* x1;

y = x1;

return d;

}

//Помножити вектор на матрицю

public static double[] MultiplyMatrices(double[] vector, double[,] matrix, int alphabet)

{

int order = (int)Math.Sqrt(matrix.Length);

double[] newVector = new double[vector.Length];

for (int j = 0; j < order; j++)

{

for (int i = 0; i < order; i++)

{

newVector[j] += matrix[j, i] \* vector[i];

}

newVector[j] = newVector[j] % alphabet;

}

return newVector;

}

}

# ДОДАТОК Б

Код класу «Preparation»для виконання дій та перетворень тексту на мові C#.

public class Preparation

{

public const string alphabet = Form1.Alphabet;

public const int SYMBOLS\_FOR\_LEN = 1;

public const int BASE = 5;

private static Random rnd = new Random();

//count amount of symbols(bi,-threegrams)

public static Dictionary<string, int> UniquesDict(string str, int amount)

{

Dictionary<string, int> dict = new Dictionary<string, int>();

if (amount == 1)

{

foreach (char ch in alphabet)

{

dict.Add(ch.ToString(), 0);

}

}

for (int i = 0; i < str.Length - amount + 1; i++)

{

string value = str.Substring(i, amount);

if (dict.ContainsKey(value))

dict[value]++;

else

dict[value] = 1;

}

return amount != 1 ? dict.OrderBy(x => x.Key).ToDictionary(x => x.Key, y => y.Value) : dict;

}

//Стрічку в цифровий формат str -input string,digit - result(int list)

public static List<int> FormDigitString(String str)

{

List<int> digit = new List<int>();

foreach (char ch in str)

{

if (alphabet.IndexOf(ch) != -1)

digit.Add(alphabet.IndexOf(ch));

}

return digit;

}

//digit list into string(actually,stringbuilder)

public static StringBuilder FormStringFromDigit(List<int> list)

{

StringBuilder sb = new StringBuilder();

foreach (int el in list)

sb.Append(alphabet[el]);

return sb;

}

//make chart

public static void MakeChart(Chart chart, Dictionary<string, int> dict)

{

chart.Visible = true;

chart.Series[0].Points.Clear();

chart.ChartAreas[0].AxisX.Interval = 1;

chart.ChartAreas[0].AxisX.Maximum = dict.Count + 0.5;

foreach (KeyValuePair<string, int> kvp in dict)

{

chart.Series[0].Points.AddXY(kvp.Key, kvp.Value);

}

}

//замалювати пробіли рандомні

public static void ColorText(RichTextBox rtb, List<int> random, int order, Color color, int shift)

{

int sum = order + shift;

foreach (int el in random)

{

rtb.Select(sum, el);

sum += el + order;

rtb.SelectionBackColor = color;

}

}

//фільтр

public static void FilterText(ref string text)

{

text = text.ToUpper();

StringBuilder sbuild = new StringBuilder();

sbuild.Append(text);

for (int i = 0; i < sbuild.Length; i++)

{

if (sbuild[i] != '\n')

{

if (!alphabet.Contains(sbuild[i]))

{

sbuild.Remove(i, 1);

i -= 1;

}

}

}

text = sbuild.ToString();

}

//середнє інтегральне відхилення

public static double FindDeviation(Dictionary<string, int> dict, int count)

{

double dev = 0;

double Max = double.MinValue;

List<double> frequency = new List<double>();

foreach (KeyValuePair<string, int> kpv in dict)

{

frequency.Add(kpv.Value / (double)count);

}

Max = frequency.Max();

double Sum = frequency.Select(x => (Max - x) / Max).Sum();

dev = Sum \* 100 / alphabet.Length;

return dev;

}

public static int FindBiteDiff(List<int> standard, List<int> input)

{

if (standard.Count != input.Count)

throw new ArgumentException("Different lengths.Impossible to perform.");

string standStr = FormBiteString(standard);

string inputStr = FormBiteString(input);

return standStr.Where((x, i) => x != inputStr[i]).Count();

}

private static string FormBiteString(List<int> input)

{

string result = "";

foreach (int el in input)

{

string middle = Convert.ToString(el,2);

while (middle.Length != 6)

{

middle = "0" + middle;

}

result += middle;

}

return result;

}

public static double[,] FormSuitableKey(string str)

{

double order=Math.Sqrt(str.Length);

int iorder=(int)order;

if (order!= (int)order)

throw new ArgumentException("It isn't key,fool!");

double[,] output=new double[iorder,iorder];

string []str2=new string[iorder];

for(int i=0;i<iorder;i++)

str2[i]=string.Concat(str.Skip(iorder\*i).Take(iorder));

for(int i=0;i<output.GetLength(0);i++)

for(int j=0;j<output.GetLength(1);j++)

output[i,j]=alphabet.IndexOf(str2[i][j]);

return output;

}

//сформувати рядок,що буде позначати кількість елементів в списку рандомів

public static int[] CalcRandomList(List<int> rand,int symbolsForLen)

{

int[] result = new int[symbolsForLen];

string str = rand.Count.ToString();

if (str.Length > symbolsForLen)

throw new ArgumentException("Your random list is too long");

int[] part2 = str.Select(x => (int)Char.GetNumericValue(x)).ToArray();

int[] part1 = Enumerable.Range(0, symbolsForLen - str.Length).Select(x => 0).ToArray();

return part1.Concat(part2).ToArray();

}

//знайти підходяще хєрове число

public static int FindMaskNumber()

{

int res = rnd.Next(0, alphabet.Length - BASE);

return res % BASE == 0 ? res : FindMaskNumber();

}

//спотворити рандомний список

public static List<int> FormMaskedRandomList(List<int> rand)

{

return rand.Select(x => x + FindMaskNumber()).ToList();

}

public static List<int> FormUnmaskedRandomList(List<int> masked)

{

return masked.Select(x => x % BASE).ToList();

}

}

# ДОДАТОК В

Код класу «Code»,у якому реалізовано розроблені шифр та хеш-функцію на мові C#.

public class Code

{

private const string ALPHABET = Form1.Alphabet;

private static int alphabet = Form1.Alphabet.Length;

private static double[,] matrix = Form1.KeyMatrix;

private static double[,] matrix2 = Form1.SKeyMatrix;

private static double[,] matrix3 = Form1.TKeyMatrix;

private static double[,] matrix4 = Form1.FKeyMatrix;

private static List<double[,]> matrList = new List<double[,]>();

private const int BASE = 5;

private const int NUM\_IN\_BLOCK = 2,K0=15,N=12,COEF\_K=-1;

public static List<int> HashFunction(List<int> input,int numOfRounds,int id,bool needToPinch)

{

List<int> output = new List<int>();

int innerid = id;

List<int> middle = new List<int>();

List<int> reserve = new List<int>();

matrList.Clear();

reserve.AddRange(input);

matrList.AddRange(new []{matrix,matrix4,matrix3,matrix2,matrix,matrix2,matrix3,matrix4,matrix2});

int amountInBlock = matrList.Select(x=>x.GetLength(0)-1).Sum();

while (reserve.Count != 0)

{

if (reserve.Count < amountInBlock)

{

while (reserve.Count != amountInBlock)

reserve.Add(26);

}

middle = BlockAction(reserve.Take(amountInBlock).ToList(), ref innerid);

for (int i = 0; i < numOfRounds-1; i++)

{

middle = BlockAction(middle, ref innerid);

}

if (needToPinch)

{

if (output.Count == 0)

{

output.AddRange(middle);

}

else

{

List<int> result = Miyaguchi\_Preneel(output, middle, reserve.Take(amountInBlock).ToList());

output.Clear();

output.AddRange(result);

}

}

else

{

output.AddRange(middle);

}

innerid = id;

reserve.RemoveRange(0, amountInBlock);

}

return output;

}

private static List<int> BlockAction(List<int>arr,ref int id)

{

List<int> output = new List<int>();

for (int j = 0; j < matrList.Count; j++)

{

int order = matrList[j].GetLength(0);

double[] temp = new double[order];

Array.Copy(arr.Take(order - 1).ToArray(), 0, temp, 1, order - 1);

temp[0] = id;

temp = MathOperations.MultiplyMatrices(temp, matrList[j], alphabet);

output.AddRange(temp.Take(order - 1).Select(x => (int)x).ToArray());

arr.RemoveRange(0, order - 1);

id = (int)temp.Last();

}

var str = string.Join(" ",output.Select(x=>x.ToString()));

return output;

}

private static List<int> Miyaguchi\_Preneel(List<int> H\_prev,List<int> H\_curr,List<int> m)

{

if (H\_prev.Count != H\_curr.Count && H\_curr.Count!=m.Count)

throw new ArgumentException("Different sizes!");

List<int> middle = m.Select((x, i) => H\_curr[i]^x).ToList();

return middle.Select((x, i) => x ^ H\_prev[i]).ToList();

}

public static List<int> HillPlusRandomEncrypt(List<int> list, double[,] matrix, int alphabet, List<int> random)

{

random.Clear();

List<int> outList = new List<int>();

List<int> reserve = new List<int>();

reserve.AddRange(list);

Random rnd = new Random();

int order = (int)Math.Sqrt(matrix.Length);

double[] Temp = new double[order];

while (reserve.Count != 0)

{

if (reserve.Count < order)

{

int resC = reserve.Count;

for (int i = 0; i < order - resC; i++)

reserve.Add(26);

}

Temp = MathOperations.MultiplyMatrices(reserve.Take(order).Select(x => (double)x).ToArray(), matrix, alphabet);

outList.AddRange(Temp.Select(x => (int)x));

reserve.RemoveRange(0, order);

if (reserve.Count != 0)

{

int ran = rnd.Next(0, BASE);

ran = ran > reserve.Count ? reserve.Count : ran;

outList.AddRange(reserve.Take(ran));

random.Add(ran);

reserve.RemoveRange(0, ran);

}

}

return outList;

}

public static List<int> HillPlusRandomDecrypt(List<int> list, double[,] matrix, int alphabet, List<int> random)

{

List<int> outList = new List<int>();

List<int> reserve = new List<int>();

List<int> resRand = new List<int>();

resRand.AddRange(random);

reserve.AddRange(list);

int order = (int)Math.Sqrt(matrix.Length);

double[] Temp = new double[order];

while (reserve.Count != 0)

{

if (reserve.Count < order)

break;

Temp = MathOperations.MultiplyMatrices(reserve.Take(order).Select(x => (double)x).ToArray(), matrix, alphabet);

outList.AddRange(Temp.Select(x => (int)x));

reserve.RemoveRange(0, order);

if (resRand.Count != 0)

{

outList.AddRange(reserve.Take(resRand.First()));

reserve.RemoveRange(0, resRand.First());

resRand.RemoveAt(0);

}

}

return outList;

}

}

# ДОДАТОК Г

Код класу «CreateAccountForm»,у якому реалізовано створення аккаунту з використанням розробленої криптографічної хеш-функції на мові C#.

public partial class CreateAccountForm : Form

{

private static string dataBase = "Database.txt";

private string login, password;

private string[] dbArray;

private List<DataBase> dbClass=new List<DataBase>();

private static Random rnd=new Random();

private const int SALT\_LENGTH = 13;

private const string ALPHABET = Form1.Alphabet;

public class DataBase

{

private int id=-1;

private string name,hash,salt;

public DataBase(int id,string name,string hash,string salt)

{

this.name = name;

this.hash = hash;

this.salt = salt;

this.id = id;

}

public int ID

{

get { return id; }

}

public string NAME

{

get { return name; }

}

public string HASH

{

get { return hash; }

}

public string SALT

{

get { return salt; }

}

public override string ToString()

{

return string.Format("{0}\\{1}\\{2}\\{3}",this.id,this.name,this.hash,this.salt);

}

}

public CreateAccountForm()

{

InitializeComponent();

}

private void createButton\_Click(object sender, EventArgs e)

{

login = loginTextBox.Text;

password = passwordTextBox.Text;

if (File.Exists(dataBase))

{

dbArray = File.ReadAllLines(dataBase);

for (int i = 0; i < dbArray.Length; i++)

{

string[] line=dbArray[i].Split(new []{'\\'});

dbClass.Add(new DataBase(Convert.ToInt32(line[0]),line[1],line[2],line[3]));

}

if (dbClass.Count > 0)

{

if (!dbClass.Select(x => x.NAME).Contains(login))

{

AddNewAccount(login, password);

}

}

else { AddNewAccount(login, password); }

}

else

{

File.Create(dataBase);

MessageBox.Show("Database can not be founded.New Database was created.");

}

}

private void CreateAccountForm\_Load(object sender, EventArgs e)

{

passwordTextBox.UseSystemPasswordChar = true;

}

private int FindSuitableId(List<DataBase>array)

{

int[] idArray = array.Select(x => x.ID).ToArray();

int i = 0;

while (idArray.Contains(i))

{

i += 1;

}

return i;

}

private void AddNewAccount(string login,string password)

{

string newSalt = String.Concat(Enumerable.Range(0, SALT\_LENGTH).Select(x => ALPHABET[rnd.Next(0, ALPHABET.Length)]));

string newHash = Preparation.FormStringFromDigit(Code.HashFunction(Preparation.FormDigitString(password + newSalt), 7, 0,true)).ToString();

dbClass.Add(new DataBase(FindSuitableId(dbClass), login, newHash, newSalt));

dbClass = dbClass.OrderBy(x => x.ID).ToList();

using (StreamWriter stream = new StreamWriter(dataBase, false))

{

foreach (DataBase db in dbClass)

{

stream.WriteLine(db.ToString());

}

}

}

}

# ДОДАТОК Д

Код класу «Autorization»,у якому реалізовано процедуру автентифікації наоснові розробленої криптографічної хеш-функції на мові C#.

namespace HashFunction

{

using DataBase = CreateAccountForm.DataBase;

public partial class Autorization : Form

{

private List<DataBase> dbClass = new List<DataBase>();

private string dataBase="Database.txt";

private string[] dbArray;

private const int RAND\_LENGTH = 13;

private static Random rnd = new Random();

private const string ALPHABET = Form1.Alphabet;

private Form1 mainForm;

public Autorization()

{

InitializeComponent();

}

private void log\_in\_Button\_Click(object sender, EventArgs e)

{

DataBase db = dbClass.Where(x => x.NAME == loginTextBox.Text).First();

if (db != null)

{

string salt = db.SALT;

string hashValue = db.HASH;

string randomSeq = string.Concat(Enumerable.Range(0,RAND\_LENGTH).Select(x=>ALPHABET[rnd.Next(0,ALPHABET.Length)]));

string computedHashValue = Preparation.FormStringFromDigit(Code.HashFunction(Preparation.FormDigitString(passwordTextBox.Text+salt),7,0,true)).ToString();

string actual = Preparation.FormStringFromDigit(Code.HashFunction(Preparation.FormDigitString(randomSeq+computedHashValue),7,0,true)).ToString();

string expected = Preparation.FormStringFromDigit(Code.HashFunction(Preparation.FormDigitString(randomSeq + hashValue), 7, 0,true)).ToString();

if (actual.Equals(expected))

{

mainForm = new Form1();

Hide();

mainForm.ShowDialog();

this.Close();

}

else { MessageBox.Show("You wrote the wrong password or login,please try again"); }

}

}

private void Autorization\_Load(object sender, EventArgs e)

{

if (File.Exists(dataBase))

{

dbArray = File.ReadAllLines(dataBase);

for (int i = 0; i < dbArray.Length; i++)

{

string[] line = dbArray[i].Split(new[] { '\\' });

dbClass.Add(new DataBase(Convert.ToInt32(line[0]), line[1], line[2], line[3]));

}

}

else { MessageBox.Show("Database can not be founded!"); }

passwordTextBox.UseSystemPasswordChar = true;

}

}

}

# ДОДАТОК Е

Код класу «Form1»,у якому реалізовано створення та перевірку електронного цифрового підпису, а також демонстрацію роботи розроблених криптографічної хеш-функції та блокового шифру, що лежать в його основі на мові C#.

public partial class Form1 : Form

{

public const string Alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ .,;-'0123456789@[]{}\"#$%^&\*()|<>+=\_?!";

public static double[,] KeyMatrix = { { 12, 0, 7, 4 }, { 17, 14, 21, 18 }, { 10, 24, 0, 13 }, { 3, 17, 8, 24 } };

public static double[,] SKeyMatrix = { { 1, 2, 1, 7, 1 }, { 2, 3, 4, 5, 6 }, { 1, 2, 8, 3, 4 }, { 5, 4, 3, 2, 1 }, { 6, 7, 5, 4, 2 } };

public static double[,] TKeyMatrix = { { 1, 2, 1, 7, 1, 0 }, { 2, 3, 4, 5, 6, 1 }, { 1, 2, 8, 3, 4, 5 }, { 5, 4, 3, 2, 1, 3 }, { 6, 7, 5, 4, 2, 4 }, { 0, 1, 7, 1, 11, 1 } };

public static double[,] FKeyMatrix = { { 13, 14, 0 }, { 18, 8, 17 }, { 12, 11, 21 } };

private const string openKey = "L)?O7.}H3,]YS Z=P>${%+<|)MJ'EX[B|MD?]{H<Z";

private static double[,] InverseMatrix, SInverseMatrix;

private List<int> encrypted = new List<int>(), encryptedF = new List<int>(), encrypted2 = new List<int>(), encrypted2F = new List<int>();

private List<int> decrypted = new List<int>(),decrypted2=new List<int>(), digit = new List<int>();

private string testOfRandomness = "";

private const int SYMBOLS\_FOR\_LEN = 1,SYMBOLS\_FOR\_LEN2=6;

private int inputC=-1;

private string filePath = "";

public Form1()

{

InitializeComponent();

testOfRandomness = File.ReadAllText("randomness.txt");

Preparation.FilterText(ref testOfRandomness);

string result = Preparation.FormStringFromDigit(Code.HashFunction(Preparation.FormDigitString(testOfRandomness), 7, 0,false)).ToString();

}

private void button1\_Click(object sender, EventArgs e)

{

string text = richTextBox1.Text;

Preparation.FilterText(ref text);

richTextBox1.Text = text;

if (inputC != -1)

{

richTextBox2.Text = Preparation.FormStringFromDigit(Code.HashFunction(Preparation.FormDigitString(text), 7, inputC,true)).ToString();

}

else

{

MessageBox.Show("Can't be performed");

}

}

private void Ok\_Click(object sender, EventArgs e)

{

if (textBox1.Text.Length != 0)

{

int input;

if (Int32.TryParse(textBox1.Text, out input))

{

if (input >= 0 && input < Alphabet.Length)

{

inputC = input;

}

else { MessageBox.Show("In interval from 0 to 63"); }

}

else { MessageBox.Show("You inputed not number"); }

}

else { MessageBox.Show("Field is empty"); }

}

private void CreateAccount\_button\_Click(object sender, EventArgs e)

{

var craForm = new CreateAccountForm();

craForm.Owner = this;

craForm.ShowDialog();

}

private void signButton\_Click(object sender, EventArgs e)

{

MyEvents me = e as MyEvents;

int symbolLen;

OpenFileDialog ofd = new OpenFileDialog(){ Filter = "Текстовые файлы(\*.txt)|\*.txt" };

if (ofd.ShowDialog() == DialogResult.OK)

{

filePath = ofd.FileName;

}

if (me == null)

{

symbolLen = SYMBOLS\_FOR\_LEN;

testOfRandomness = File.ReadAllText(filePath);

Preparation.FilterText(ref testOfRandomness);

string result = Preparation.FormStringFromDigit(Code.HashFunction(Preparation.FormDigitString(testOfRandomness), 7, 0,true)).ToString();

digit = Preparation.FormDigitString(result);

}

else

{

symbolLen = SYMBOLS\_FOR\_LEN2;

testOfRandomness = rt1.Text.Length == 0 ? File.ReadAllText(filePath) : rt1.Text;

Preparation.FilterText(ref testOfRandomness);

rt1.Text = testOfRandomness;

digit = Preparation.FormDigitString(testOfRandomness);

}

encryptedF.Clear(); encrypted2F.Clear();

InverseMatrix = Preparation.FormSuitableKey(string.Concat(openKey.Take(16)));

SInverseMatrix = Preparation.FormSuitableKey(string.Concat(openKey.Skip(16)));

List<int> random = new List<int>(), random2 = new List<int>();

//1-й раунд шифрування

encrypted = Code.HillPlusRandomEncrypt(digit, KeyMatrix, Alphabet.Length, random);

int[] count = Preparation.CalcRandomList(random, symbolLen);

List<int> maskedR = Preparation.FormMaskedRandomList(random);

encryptedF.AddRange(count); encryptedF.AddRange(maskedR); encryptedF.AddRange(encrypted);

//2-й раунд шифрування

encrypted2 = Code.HillPlusRandomEncrypt(encryptedF, SKeyMatrix, Alphabet.Length, random2);

int[] count2 = Preparation.CalcRandomList(random2, symbolLen);

List<int> maskedR2 = Preparation.FormMaskedRandomList(random2);

encrypted2F.AddRange(count2); encrypted2F.AddRange(maskedR2); encrypted2F.AddRange(encrypted2);

if (me == null)

{

signedTextBox.Text = Preparation.FormStringFromDigit(encrypted2F).ToString();

hashTextBox.Text = Preparation.FormStringFromDigit(digit).ToString();

checkHash.Enabled = true;

}

else

{

rt2.Text = Preparation.FormStringFromDigit(encryptedF).ToString();

Preparation.ColorText(rt1, random, KeyMatrix.GetLength(0), Color.Yellow, 0);

Preparation.ColorText(rt2, random, KeyMatrix.GetLength(0), Color.Yellow, count.Length + maskedR.Count);

rt3.Text = Preparation.FormStringFromDigit(encryptedF).ToString();

rt4.Text = Preparation.FormStringFromDigit(encrypted2F).ToString();

Preparation.ColorText(rt3, random2, SKeyMatrix.GetLength(0), Color.Blue, 0);

Preparation.ColorText(rt4, random2, SKeyMatrix.GetLength(0), Color.Blue, count2.Length + maskedR2.Count);

}

}

private void checkHash\_Click(object sender, EventArgs e)

{

MyEvents me = e as MyEvents;

int symbolLen;

if (me == null)

{

symbolLen = SYMBOLS\_FOR\_LEN;

}

else

{

symbolLen = SYMBOLS\_FOR\_LEN2;

}

//1-й раунд розшифровки

int count2 = Convert.ToInt32(String.Concat(encrypted2F.Take(symbolLen)));

List<int> random2 = Preparation.FormUnmaskedRandomList(encrypted2F.Skip(symbolLen).Take(count2).ToList());

encrypted2 = encrypted2F.Skip(symbolLen + random2.Count).ToList();

decrypted2 = Code.HillPlusRandomDecrypt(encrypted2, SInverseMatrix, Alphabet.Length, random2);

//2-й раунд розшифровки

int count = Convert.ToInt32(String.Concat(decrypted2.Take(symbolLen)));

List<int> random = Preparation.FormUnmaskedRandomList(decrypted2.Skip(symbolLen).Take(count).ToList());

encrypted = decrypted2.Skip(symbolLen + random.Count).ToList();

string gfdsf = Preparation.FormStringFromDigit(decrypted2).ToString();

decrypted = Code.HillPlusRandomDecrypt(encrypted, InverseMatrix, Alphabet.Length, random);

if (me == null)

{

decrypted = decrypted.Take(32).ToList();

decryptedHashTextBox.Text = Preparation.FormStringFromDigit(decrypted).ToString();

string testOfRandomness2 = File.ReadAllText(filePath);

Preparation.FilterText(ref testOfRandomness2);

string result = Preparation.FormStringFromDigit(Code.HashFunction(Preparation.FormDigitString(testOfRandomness2), 7, 0,true)).ToString();

computedHashTextBox.Text = result;

if (decryptedHashTextBox.Text.Equals(result))

MessageBox.Show("Document is original!");

else

MessageBox.Show("Document is fake!");

}

else

{

rt5.Text = Preparation.FormStringFromDigit(decrypted).ToString();

}

}

private class MyEvents : EventArgs

{

}

private void Encrypt\_Click(object sender, EventArgs e)

{

this.signButton\_Click(this, new MyEvents());

Decrypt.Enabled = true;

}

private void Decrypt\_Click(object sender, EventArgs e)

{

this.checkHash\_Click(this,new MyEvents());

}

}

}