Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)
Факультет комп'ютерно-інформаційних систем і програмної інженерії
(назва факультету)
Кафедра кібербезпеки
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(освітній рівень)

на тему: *" Аналіз властивостей алгоритму UMAC-32 для забезпечення автентичності і цілісності даних "*

Виконав: студент (ка)

Спеціальності:
125 «Кібербезпека»
(шифр і назва напряму підготовки, спеціальності)

Мельник С.А.
підпис (прізвище та ініціали)

Керівник *Карташов В.В.*
Підпис (прізвище та ініціали)

Нормоконтроль *Лобур Т.Б.*
підпис (прізвище та ініціали)

Завідувач кафедри *Загородна Н.В.*
підпис (прізвище та ініціали)

Рецензент
підпис (прізвище та ініціали)

м. Тернопіль – 2022

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра кібербезпеки
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Загородна Н.В.
(підпис) (прізвище та ініціали)
«__» _____ 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 125 Кібербезпека
(шифр і назва спеціальності)

Студенту Мельнику Степану Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Аналіз властивостей алгоритму UMAC-32 для забезпечення автентичності і цілісності даних

Керівник роботи Карташов В.В.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «23» 03 2022 року № 4/7-178

2. Термін подання студентом завершеної роботи 17.06.2022

3. Вихідні дані до роботи _____

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Аналіз послуг і механізмів забезпечення безпеки інформації відповідно до міжнародних стандартів ISO 7498, ISO / IEC 10181. 1.1 Аналіз послуг і механізмів. 1.2 Аналіз механізмів автентичності на основі геш-функції. 1.3 Аналіз конкурсантів NESSIE, щодо формування MAC-кодів. 1.4 Висновок. 2 Дослідження методу отримання геш-коду на основі використання UMAC-32. 2.1 Загальна схема кодів автентичності повідомлень UMAC. 2.2 Перший етап гешування. 2.3 Другий етап гешування. 2.4 Третій етап гешування. 2.5 Висновок. 3.Експериментальні досліджен ня ефективності безпечного гешування на основі UMAC-32. 4. Безпека життєдіяльності, основи хорони праці. Висновки. Список використа них джерел. Додаток А. Додаток Б.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Титулка. Актуальність роботи. Загрози інформаційної безпеки. Послуги та механізми забезпечення автентичності. Класифікація функцій гешування. Класифікація MAC-кодів. Європейський конкурс NESSIE. Дослідження швидкості формування MAC-коду. Загальна схема кодів автентичності повідомлень UMAC. Перший рівень гешування. Другий рівень гешування. Третій рівень гешування. Схема формування ключів. Схема формування псевдо-випадкових підложки(PDF). Схема формування MAC-коду. Швидкість вичислення UMAC. Швидкодія алгоритмів формування MAC-кодів. Порівняльна характеристика видів UMAC. Порівняльна характерис тика MAC-кодів.Структурна схема пристрою для формування мас-коду. Результати тестування геш-функцій. Висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи хорони праці	Пулька Ч.В., проф. кафедри МТ		

7. Дата видачі завдання 16.02.2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	16.02 – 19.02	*Виконано*
2.	Підбір літературних джерел	20.02 – 27.02	*Виконано*
3.	Опрацювання джерел в галузі дослідження	28.02 – 16.03	*Виконано*
4.	Розроблення програмного коду	17.03 – 20.03	*Виконано*
5.	Тестування роботи програми та верифікація результатів	20.03-05.04	*Виконано*
6.	Оформлення розділу «Аналіз послуг і механізмів забезпечення безпеки інформації відповідно до міжнародних стандартів ISO 7498, ISO / IEC 10181»	06.03 – 17.04	*Виконано*
7.	Оформлення розділу «Дослідження методу отримання геш-коду на основі використання UMAC-32»	18.04 – 29.04	*Виконано*
8.	Оформлення розділу «Експериментальні дослідження ефективності безпечного гешування на основі UMAC-32»	30.04 – 13.05	*Виконано*
9.	Виконання завдання до підрозділу «Безпека життєдіяльності, основи хорони праці»	14.05 – 21.05	*Виконано*
10.	Оформлення кваліфікаційної роботи	22.05 – 05.06	*Виконано*
11.	Нормоконтроль	06.06 – 12.06	*Виконано*
12.	Перевірка на плагіат	10.06 – 15.06	*Виконано*
13.	Попередній захист кваліфікаційної роботи	16.06 – 19.06	*Виконано*
14.	Захист кваліфікаційної роботи	24.06.2022	

Студент _____ *Мельник С.А.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Карташов В.В.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Аналіз властивостей алгоритму UMAC-32 для забезпечення автентичності і цілісності даних // Кваліфікаційна робота ОР «Бакалавр» // Мельник Степан Андрійович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем та програмної інженерії, кафедра кібербезпеки, група СБс–42 // Тернопіль, 2022 // С. – , рис. – , табл. – , слайдів – , бібліогр. – .

Ключові слова: ГЕШ-ФУНКЦІЯ, UMAC, ГЕШУВАННЯ, ЦІЛІСНІСТЬ, АВТЕНТИЧНІСТЬ.

Об'єктом дослідження є процес забезпечення цілісності і автентичності інформації в інформаційних системах за допомогою алгоритму гешування UMAC-32.

Мета роботи полягає у розробці універсального алгоритму гешування UMAC-32, а також в аналізі механізмів забезпечення автентичності і цілісності інформації при використанні алгоритмів цифрового підпису і гешування, в оцінці їх основних ймовірно-часових характеристик.

Предметом дослідження є методи ключового гешування для забезпечення цілісності й автентичності інформації за допомогою алгоритму UMAC-32.

В результаті виконання роботи отримані наступні результати:

Проведено аналіз сучасних механізмів та протоколів забезпечення цілісності та автентичності даних на основі використання цифрових підписів та альтернативних їм варіантам геш-кодів на основі безключових та ключових геш-функцій.

Проведено аналіз методів побудови ключових геш-функцій (MAC-кодів), які при рівних умовах з MDC-кодами дозоляють інтегровано вирішувати

завдання забезпечення цілісності та автентичності повідомлень без додаткового використання алгоритмів шифрування.

Алгоритм UMAC-32 може бути використаний в інформаційних системах масового призначення, які потребують швидкого обміну інформацією та якісного рівня забезпечення цілісності та автентичності переданої інформації.

ANNOTATION

Analysis of algorithm UMAC-32 vulnerabilities to provide data authenticity and completeness // Qualification thesis of educational level "Bachelor // Melnyk Stepan Andriiovych // Ternopil Ivan Pul'uj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Cyber Security // Ternopil, 2022 // P. -__, Fig. -__, Table - __, Slides - __, References - __.

Keywords: HESH FUNCTION, UMAC, GASHING, INTEGRITY, AUTHENTICITY.

The object of research is the process of ensuring the integrity and authenticity of information in information systems using the hashing algorithm UMAC-32.

The purpose of the work is to develop a universal hashing algorithm UMAC-32, as well as to analyze the mechanisms for ensuring the authenticity and integrity of information using digital signature and hashing algorithms, in assessing their main probabilistic and temporal characteristics.

The subject of the research is the methods of key hashing to ensure the integrity and authenticity of information using the UMAC-32 algorithm.

As a result of performance of work the following results are received:

The analysis of modern mechanisms and protocols for ensuring the integrity and authenticity of data based on the use of digital signatures and alternative variants of hash codes based on keyless and key hash functions.

An analysis of methods for constructing key hash functions (MAC codes), which under equal conditions with MDC codes allow to solve the problem of ensuring the integrity and authenticity of messages without the additional use of encryption algorithms.

The UMAC-32 algorithm can be used in mass information systems that require rapid information exchange and a high level of integrity and authenticity of transmitted information.

ЗМІСТ

ВСТУП

Розширення можливостей використання інформаційних систем вимагає забезпечення їх надійною і стійкою системою інформаційної безпеки. Використання сучасних механізмів захисту не повністю задовольняє вимогам користувачів, оскільки збільшення обсягів даних, що передаються, сповільнюється унаслідок повільної автентифікації (підтвердження) відправника з використанням сучасних механізмів забезпечення цілісності та автентичності інформації. У зв'язку з цим виникає необхідність вирішення проблеми, яка полягає в розробці універсального і головне швидкого алгоритму автентифікації і забезпечення цілісності при передачі інформації в інформаційних системах [1; 3; 5; 10].

Традиційно проблеми контролю цілісності інформації і вироблення коду автентифікації для інформаційних систем вирішуються за допомогою алгоритмів гешування і генерації псевдовипадкових послідовностей, а значить, завдання розробки нових класів таких алгоритмів, що відрізняються від використовуваних в даний час великою швидкодією, ефективнішою програмною і апаратною реалізацією і не поступливим їм з погляду достовірності, є вельми актуальним.

Найбільш широковживаним на даний момент є механізм, у якому використовується електронно-цифровий підпис і коди автентифікації повідомлення.

Аналіз сучасних вимог до забезпечення оперативності, надійності та скритності передачі даних в інформаційних системах показав, що серед розглянутих алгоритмів – переможців міжнародного конкурсу NESSIE, алгоритми формування кодів автентифікації повідомлень повинні забезпечувати можливість формування MAC-коду довільної довжини повідомлення за час, який не перевищує формування цифрового підпису на еліптичних кривих. Таким чином, алгоритм UMAC, який має найвищу швидкість формування MAC-коду, близькою до 10^9 біт/с на процесорі Pentium 4

за дозволений час (час формування цифрового підпису на еліптичних кривих ДСТУ-4145), в цілому забезпечує вимоги до алгоритмів формування геш-коду. Для забезпечення стійкості в якому використовуються процедури CBC блочно-симетричного шифру AES або процедура алгоритму HMAC [3; 5; 10; 22; 24].

Механізм забезпечення цілісності та автентичності інформації в інформаційних системах, побудованих на геш-функції UMAC-32 дозволяє вирішити протиріччя та забезпечити оперативне (своєчасне) формування MAC-коду для повідомлень довільної довжини [1; 5; 17; 20; 24].

Метою бакалаврської роботи є аналіз послуг і механізмів забезпечення безпеки інформації в інформаційних системах, дослідження методу отримання геш-коду на основі використання UMAC-32, експериментальні дослідження ефективності безпечного гешування на основі UMAC-32.

1. АНАЛІЗ ПОСЛУГ І МЕХАНІЗМІВ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ ІНФОРМАЦІЇ ВІДПОВІДНО ДО МІЖНАРОДНИХ СТАНДАРТІВ ISO 7498, ISO / IEC 10181.

1.1 Аналіз послуг і механізмів

На сьогоднішній день існують загрози безпеці інформації. Вони виникли через популярні способи передачі даних, їх обробки та накопичення в системах інформацій.

Загрозою інформаційної безпеки в інформаційній системі називається можливість реалізації інформаційної дії, яка обробляється автоматизованою системою (АС), та призводить до спотворення, знищення, копіювання, блокування доступу до інформації, а також можливість дії на компоненти АС, що призводить до втрати, знищення або збою функціонування носія інформації, засоби взаємодії з носієм або засобу його керування [1–4; 7; 11; 13; 16].

Загальна класифікація небезпек (загроз), що загрожують інформаційним системам наведена на рисунку 1.1.

Оцінюючи стан інформаційної безпеки випливає, їх об’єм і якісні властивості на кожному етапі не можуть задовольнити реальні потреби систем в необхідних ресурсах. Через це визначено основні реальні і потенційні загрози [1; 2; 4].

Аналізуючи рисунок 1.1 ми розуміємо, що джерело випадкових небезпек інформаційних систем можуть бути некоректні дії працівників або їхніх користувачів, непрацюючі програмні та апаратні засоби, випадкові пошкодження в програмно-апаратному забезпеченні і т.д. Ці всі загрози можуть спричинити значні збитки. Але з точки зору ефективності функціонування значними є спеціальні загрози, які завдають збитків інформаційній системі або користувачам [1; 3]. Реалізовуються навмисні загрози завдяки значної тривалості великої акаки вірусами (недійсними апитами), та т.д. Наслідки наступні:

Рисунок 1.1 – Основні типи загроз інформаційної безпеки

Реакція інформації на помилку, яка має змістовну форму та коректний зміст (модифікація даних), втрата інформаційних даних (руйнування інформації), знайомство з потрібною інформацією чужих людей.

Головна ціль засобів захисту комп'ютерних систем та мереж є запобігання вище названим загрозам в інформаційній безпеці.

Засобами захисту інформаційних систем можуть виступати послуги інформаційної безпеки [1].

Опираючись на вимоги міжнародних стандартів в області криптографії (ISO 7498, ISO/IEC 10181) визначаємо 5 базових послуг безпеки: конфіденційність даних, автентифікація, управління доступом, приналежність даних та їх цілісність [1].

Рисунок 1.2 – Послуги інформаційної безпеки

Для кожної конкретної послуги в сфері інформаційної безпеки розроблений спеціальний механізм. Беручи до уваги основні положення сучасної теорії захисту інформації застосовуються різноманітні криптографічні механізми [1-3] (рисунок 1.3).

На рисунку 1.3 представлені механізми інформаційної безпеки, які скеровані, щоб забезпечувати ефективний та необхідний рівень інформаційних даних, що використовуються при передачі. Це відіграє велику роль у сучасних

системах інформаційної безпеки, які повинні виконувати певний перелік вимог, які гарантують наступні послуги та механізми для забезпечення автентичності інформації відповідно до стандарту ISO 7498-2 приведені на рисунку 1.4.

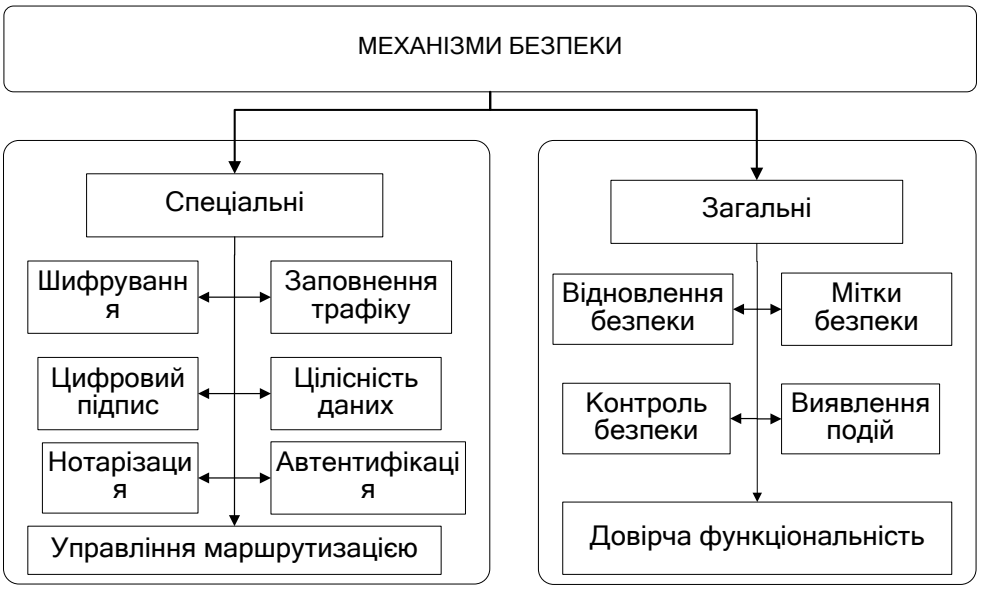

Рисунок 1.3 - Механізми інформаційної безпеки

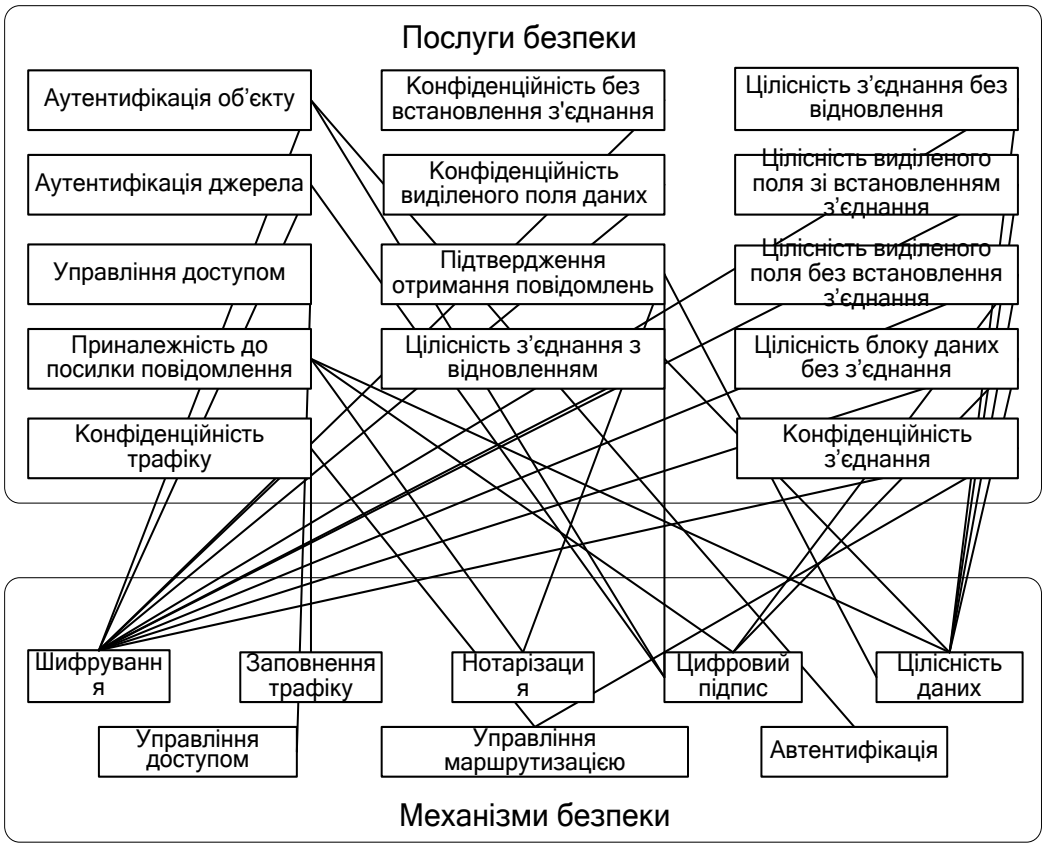

Рисунок 1.4 - Послуги і механізми забезпечення автентичності

1.2 Аналіз механізмів автентичності на основі геш-функції

Найбезпечнішим способом досягнення завдання, в основі якого є аутентифікація інформації та джерела великої кількості повідомлень, є процедури створення ЦП (цифровий підпис). Ці процедури, в свою чергу, сформовані на базі асиметричного криптографічного алгоритму [2].

Для того, щоб зберегти цілісність інформації та її аутентичність, потрібно забезпечити умови, щоб відбувся процес контролю цілісності інформаційних даних та процесу аутентифікації особи, що здійснює відправлення даних. Це все відбувається за допомогою цифрового підпису.

Електронний цифровий підпис (ЕЦП) - це рядок даних, який залежить від таких параметрів, як таємного ключа, що відомий тільки особі, яка підписує; та від повідомлення, яке представлене в цифровому вигляді [2 – 7].

Існує дві процедури, які є в основі системи створення ЕЦП:

- процедура генерації підпису (коли використовується таємний ключ, особою, що відправляє);
- процедура верифікації підпису (коли використовується відкритий ключ особою, що відправляє).

Таємний ключ - це ключ, який відомий тільки тій особі, що створює ЕЦП. А відомий ключ відкритий для всіх користувачів мережі [8; 9].

Щоб забезпечити ефективність автентичності повідомлень та процедури верифікації підпису, потрібно починати використовувати однонаправлені геш-функції.

Результатом роботи однонаправленої функції може виступати значення (геш-код), яке використовується для перевірки цілісності переданих даних і створюється з використанням функції HMAC, де – Hashed Message Authentication Code (HMAC) – гешований код перевірки повідомлення. HMAC схожий на MAC, але при цьому використовується зв’язка геш-кодування-алгоритм разом із загальним секретним ключем. Загальний секретний ключ прикріплюється до даних, які гешуються. Це дозволяє зробити гешування

безпечнішим, оскільки обидві сторони повинні мати однакові секретні ключі для підтвердження достовірності даних.

Класифікація геш-функцій представлена та рисунок 1.5. MDC-коди формують геш-код повідомлення (або стислий його образ), який відповідає необхідним критеріям. Ці критерії розділяються на блочні шифри та модульну арифметику. Гарантія абсолютної цілісності повідомлень забезпечується сукупністю MDC-кодами та інших механізмів. Безключові геш-функції є одним із складових елементів цифрових підписів (рисунок 1.5).

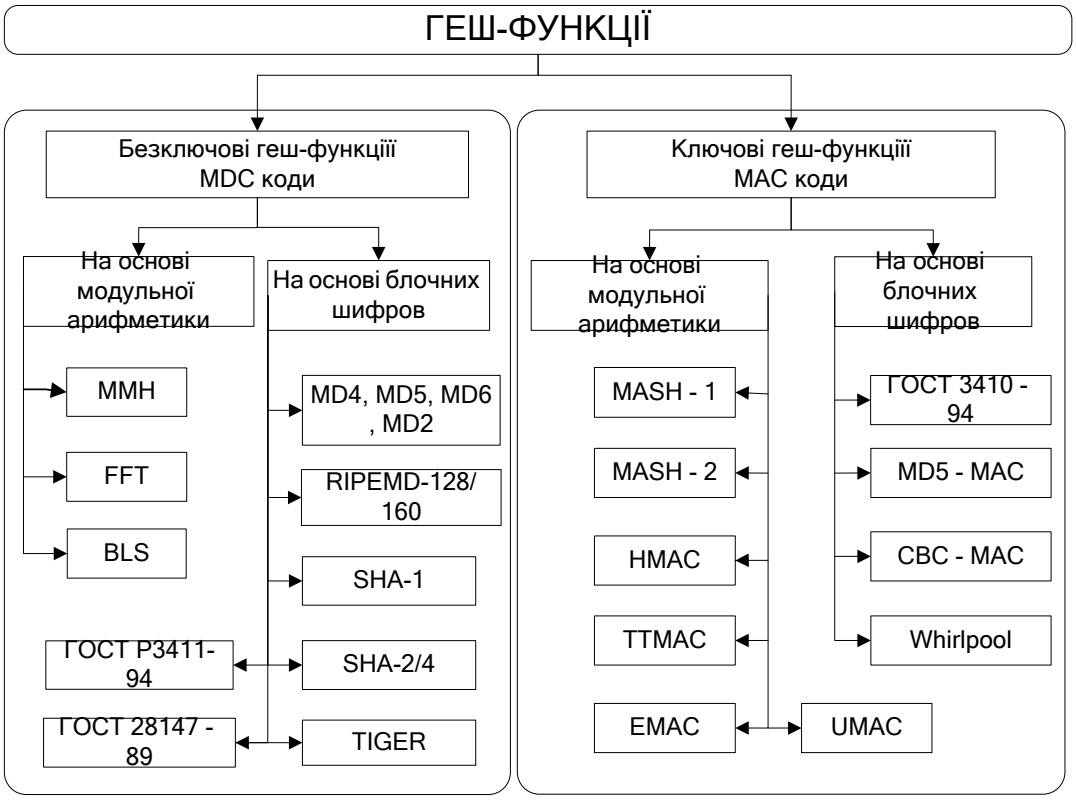

Рисунок 1.5. Класифікація геш-функцій

Порівняльна характеристика безключових геш-функцій наведена у таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика без ключових геш-функцій

Характеристика	MD5	MD6	SHA-1	SHA-2 (256/512)	ГОСТ 28147	RIPEMD-128	RIPEMD-160
Довжина дайджесту, біт	128	512	160	256/512	256	128	160
Розмір блоку обробки, біт	512	512	512	512/1024	512	512	512
Кількість ітерацій	64	168	80	64/80	32	128	160
Кількість елементарних логічних функцій	4	4	3	6/6	8	5	5
Кількість додаткових констант	64	-	4	64/64	-	4	9
Швидкість роботи на Pentium III, Мбіт/с	574,6	-	344,4	135,5 / 68,7	315,27	63.8	39.8

Аналізючи таблицю 1.1 випливає, що для безключової геш-функції важливим недоліком є, що існує можливість утворення дублікату повідомлення з ідентичним геш-кодом, і немає обчислювальної стійкості.

Проведені дослідження [11-16] підтверджують, що існує новітній метод збереження цілісності інформації та її автентичності: застосування надмірного коду, який базується на основі MAC-кодів. Ця процедура підтверджує дійсність повідомлень, що передаються [17,18].

Підходи до побудови MAC-кодів і обумовлюють класифікацію MAC-кодів, яка представлена на рисунку 1.6.

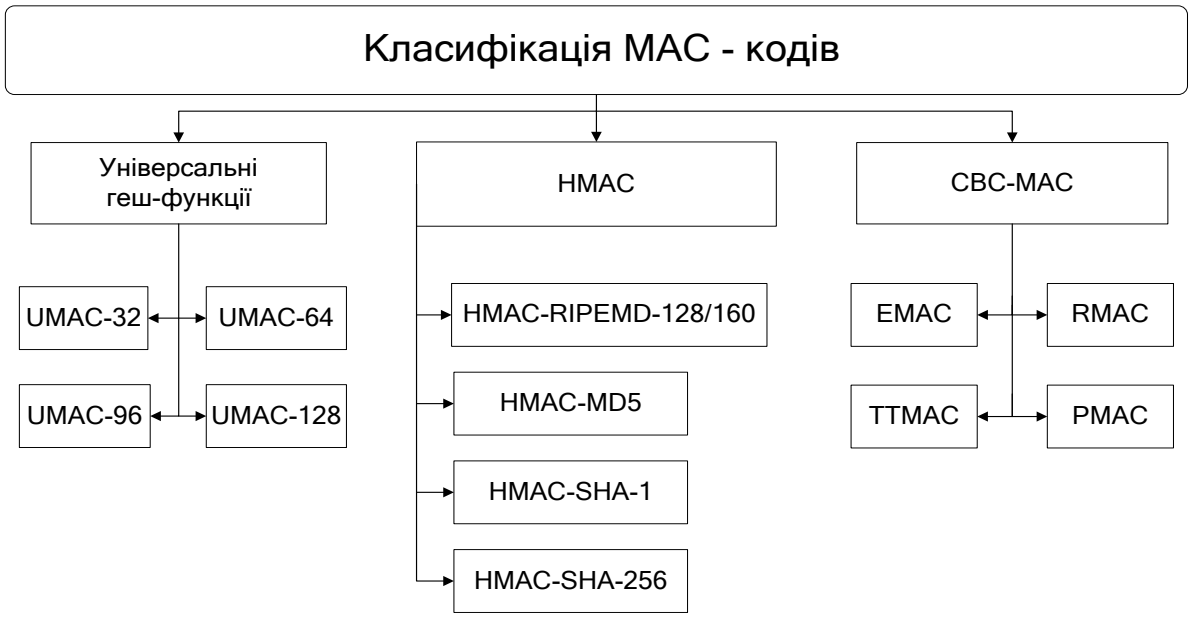

Рисунок 1.6 - Класифікація MAC-кодів

Алгоритми формування MAC-кодів розглядаються як геш-функції з двома вхідними параметрами, а саме повідомленням і секретним ключем. Це формує на виході такого алгоритму двійковий рядок фіксованої довжини. Враховуючи ці деталі,на практиці неможливо сформувати ідентичний рядок без знання ключа.

Функції, що впливають на формування коду автентифікації повідомлень (MAC) є підкласом ключових геш-функцій. Також ці функції володіють додатковою властивістю обчислювальної стійкості.

Обчислювальною стійкістю вважають знаходження геш-даних повідомлень тільки завдяки відомого засекреченого ключа. Це означає, що для ключової геш-функції h і одній або коректніших наборів (xi,h(xi,k)), а також невідомому засекреченому ключу k неможливо здійснити обчислення, яке забезпечить знайти значення іншої пари (x,h(x,k)) для будь-якого значення $x \neq x_i$.

Вимогою обчислювальної стійкості є стійкість ключа, тобто неможливість знаходження секретного ключа k), проте, вимогу стійкості ключа не гарантує те, вимога обчислювальної стійкості буде виконана.

Дуже важливим фактом є різні функції створення MAC-коду і однонаправлені геш-функції, що містять таємний код, цей в свою чергу є частиною повідомлення. У процесах створення MAC-коду таємний код використовується в кожному сегменті даних, а в однонаправлених геш-функціях застосовується префіксний ключ (на початку повідомлення) та постфікс ний ключ (в кінці повідомлення).

Дуже часто терміни MDC і MAC використовуються як взаємозамінні, але насправді між ними існує дуже важлива ріжниця. MDC не використовує таємного коду, тому при передачі даних MDC-код повинен бути завжди закодований, щоб гарантувати цілісність повідомлень. MAC- код завжди використовує таємний код, тому при передачі даних він не потребує процесу кодування, щоб зберегти ту ж саму цілісність інформації.

1.3. Аналіз конкурсантів NESSIE, що до формування MAC-кодів

У січні 2000 року, щоб визначити і оцінити якість криптопримітивів, в Бельгії почався трирічний європейський криптопроєкт NESSIE (New European Schemes for Signatures, Integrity, and Encryption). Метою даного проєкту став відбір криптографічних алгоритмів. В майбутньому ці алгоритми повинні стати основою для формування криптостандартів Європи.

В основі проєкту NESSIE поставлені такі задачі, що відповідають за набір найкращих 10-ти криптографічних примітивів. Цей набір містить алгоритми цифрового підпису, алгоритми різного роду шифрування та інші. Відбір кандидатів відбуваються за певними критеріями: справжня безпека, гнучкість та продуктивність ринку.

MAC-код зберігає автентичність інформації та її цілісність, дозволяючи контролерам виявляти будь-які зміни в початкових даних. Важливою перевагою MAC-код заключається в тому, що один і той же таємний код використовується при отриманні і перевірці інформації. З цього випливає, що

до початку роботи з повідомленнями, одержувач і відправник повинні домовитись про спільний таємний код.

Робота алгоритму MAC-коду заключається в тому, що першічергові сегменти відкритих даних переважно містять звичайну службову інформацію, яка не проходить процес шифрування. Для отримання MAC-коду відкриті дані представляються у вигляді 64-розрядних блоків T(i) (i = 1,2..., m де m визначається обсягом даних, які шифруються).

Загальний принцип роботи MAC-кодів приведено на рисунку 1.7.

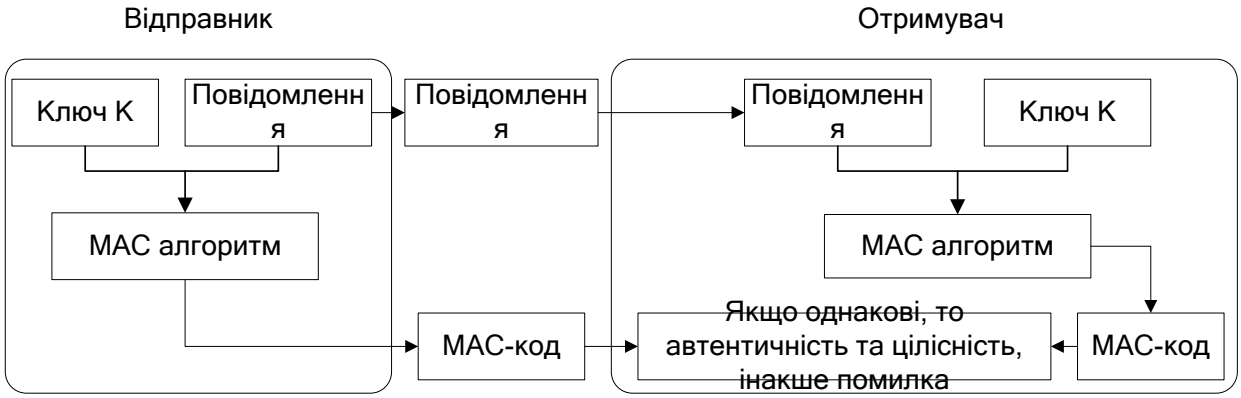

Рисунок 1.7 – Принцип роботи MAC алгоритмів

Важливою позитивною характеристикою даного механізму є найбільш простий алгоритм високої швидкодії [5].

Принцип роботи алгоритму Two-Track-MAC виглядає таким чином, як показано на рисунку 1.8.

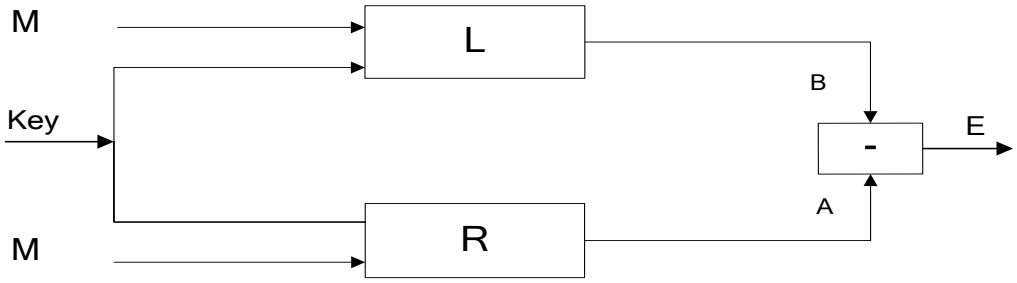

Рисунок 1.8 - Принцип роботи алгоритму Two-Track-MAC

Two-Track-MAC має два паралельних блока перетворень R (K, M) та L (K, M), що беруть на вхід повідомлення M і ключ K. У результаті кожен з блоків працює незалежно один від одного і створює два різних уявлення (A і B на рисунку 1.8) одних і тих же даних.

Розмір повідомлення M становить 512 біт. Розмір ключа K завжди фіксований і становить 160 біт. Для ускладнення перетворень L (K, M) дає на виході п’ять 32-бітових слів (A0, A1, A2, A3, A4). Тобто формально поділяє поки ще попередній варіант ключа на 5 частин однакових розмірів. Аналогічно набір (B_0, B_1, B_2, B_3, B_4) дає на виході функція R (K, M). Потім ці слова віднімаються за модулем 2^{32}. Це свого роду змішування двох значень для приведення до фіксованої довжини 160 біт. Остаточний результат: E = (E_0, E_1, E_2, E_3, E_4), де $E_i = A_i - B_i \bmod 2^{32}$, i = 0, 1, 2, 3, 4. Власне це і є те, заради чого все робилося. E є кодом автентичності повідомлення M.

TTMAC містить найбільшу небезпеку з MAC примітивів, але одночасно має важливу перевагу, яка заключається в короткотривалих даних, а ще містить оптимальну гручкість ключів.

Схема роботи алгоритму CBC-MAC показано на рисунку 1.9.

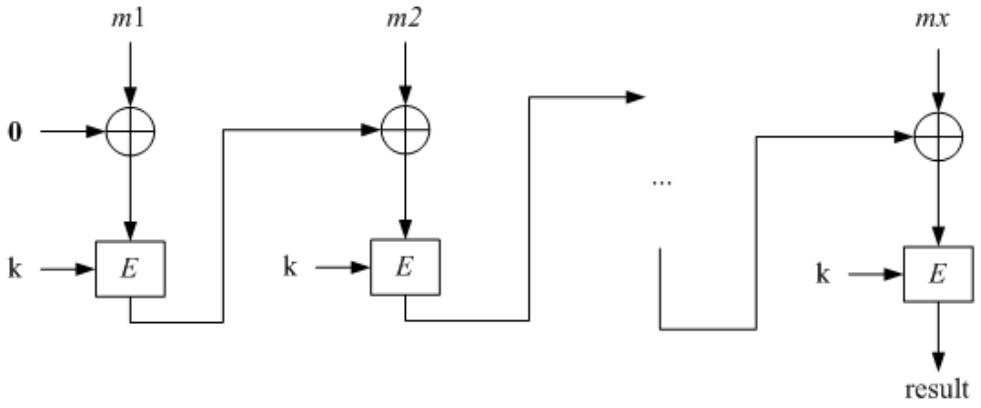

Рисунок 1.9 – Схума алгоритму CBC-MAC

Для того, щоб сформувати MAC з блочного шифру E, будемо застосовувати найпростіший і найвідоміший алгоритм $CBC\ MAC$. При $M = M[1] \circ M[2] \circ ... \circ M[m]$ буде рядок значення, де $|M[1]| = |M[2]| = ... = |M[m]| = n.$ З цього випливає, що $CBC_K(M)$, $CBC\ MAC$ з M з ключами K, визначається як $Y[m]$, де $Y[i] = E_K(M[i] \oplus Y[i-1])$ для $i = 1, ..., m$ і $Y[0] = 0^n$.

EMAC (відомий DMAC) – це алгоритм, який застосовується у формуванні MAC-алгоритмів. Він один з кращих, через можливе повторне застосування існуючих процесів шифрування. Алгоритм EMAC найбільше підходить для малих значень.

Схема роботи алгоритму HMAC показано на рисунку 1.10.

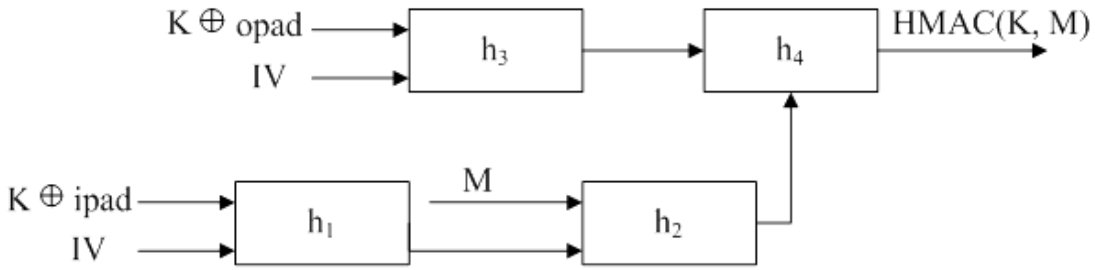

Рисунок 1.10 – Схема роботи алгоритму HMAC

Процес створення MAC-кодів застосовуючи алгоритм HMAC:

$$HMAC(K, M) = h((K \oplus opad) \| h(K \oplus ipad) \| M),$$

де h — геш-функція, K — таємний ключ,

M —значення для ідентифікації, II – конкатенація,

$opad$ — 0x5c5c..5c (довжина, що рівна розміру блока)

$ipad$ — 0x3636..36 (довжина, що рівна розміру блока)

Схема роботи алгоритму UMAC показано на рисунку 1.11.

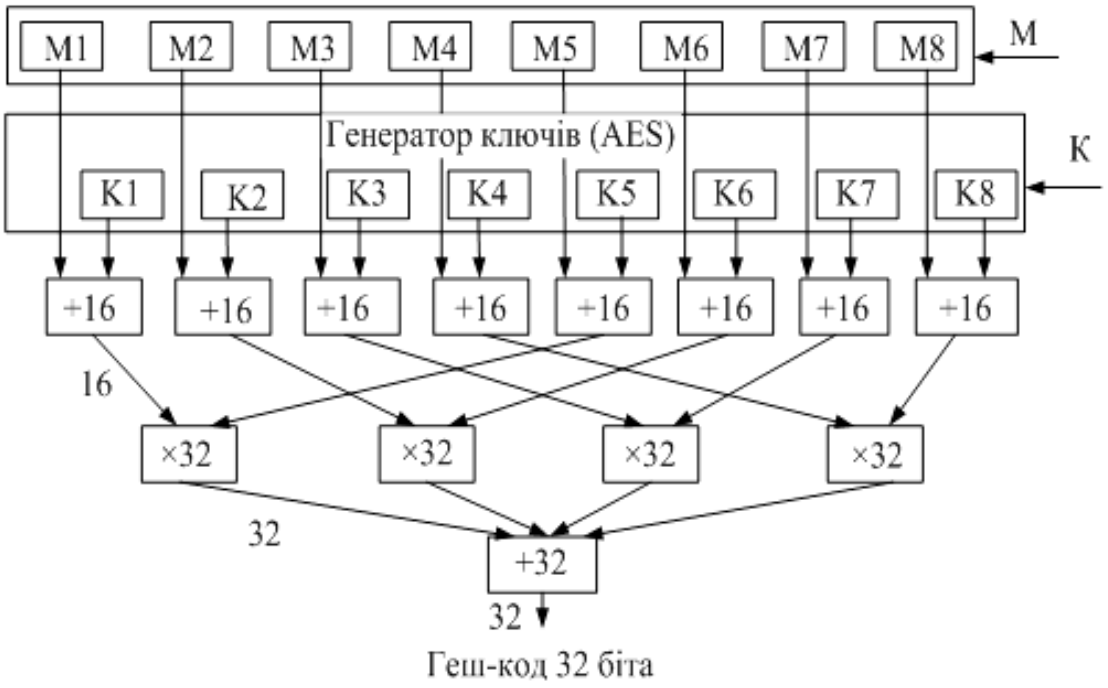

Рисунок 1.11 – Схема роботи алгоритму UMAC

Ключова стійкість UMAC зводиться до стійкості шифру AES. Математичний запис функції UMAC-32 представлений виразом:

UMAC (K, M, Nonce) = (UHASH(K, M) + PDF (K, Nonce))mod2.

Це підтверджує, що MAC алгоритм має обчислювальну стійкість і як наслідок не дозволяє довести теоретично вірогідність колізії.

Алгоритми створення MAC-кодів, що повинні відповідати рекомендаціям проєкту NESSIE, мають такі основні властивості:

- статистичні характеристики розподілів MAC-кодів;
- прискорене застсування алгоритмів створення MAC-кодів;
- високий рівень захисту від стандартних атак.

Для оцінки складності реалізації геш-функцій була виміряна швидкість роботи офіційних реалізацій на мові C, а також для порівняння деякі оптимізовані версії з використанням мови assembler. В якості компілятора

використовувався Microsoft Visual C + + 6.0. Тестування швидкості роботи проводилося на комп'ютерах з різними характеристиками під управлінням операційної системи Microsoft Windows 2000 Professional. Результати тестування якісної роботи алгоритмів гешування представлені в таблиці 1.2.

Таблиця 1.2 – Аналіз тестування швидкості алгоритмів гешування

Функція гешування	Кількість циклів	Мова реалізації	Швидкість роботи на Celeron 600 MHz	Швидкість роботи на Pentium III1000 MHz
Whirlpool	10	C	28,013 Мбіт/с	46,961 Мбіт/с
SHA-2 (512)	80	C	41,159 Мбіт/с	68,701 Мбіт/с
SHA-2 (256)	64	C	81,308 Мбіт/с	135,557 Мбіт/с
ГОСТ 34311-95	-	C+ Assembler	49,408 Мбіт/с	83,056 Мбіт/с
HAVAL	96(128,160)	C	337,842 Мбіт/с	564,809 Мбіт/с
SHA-1	80	C Assembler	206,285 Мбіт/с 361,581 Мбіт/с	344,433 Мбіт/с 605,558 Мбіт/с
RIPEMD-160	160	C	147,465 Мбіт/с	246,568 Мбіт/с
MD5	64	C	278,715 Мбіт/с	574,635 Мбіт/с
MD4	48	C	344,086 Мбіт/с	467,793 Мбіт/с
UMAC	-	C, C+ Assembler	989,371 Мбіт/с 3518,900 Мбіт/с	1648,953 Мбіт/с 5885,057 Мбіт/с
Rijndael CBC–MAC	14	C	139,376 Мбіт/с	231,255 Мбіт/с
ГОСТ 28147-89 (режим 4)	16	C+ Assembler	189,559 Мбіт/с	315,270 Мбіт/с

Завдяки проведеному дослідженню, виявилось, що найбільш перспективним алгоритмом обробки інформації є UMAC. Цей алгоритм має два режими роботи: односпрямована геш-функція і функція вироблення MAC. Хоча для односпрямованої функції він має невисоку стійкість, але зате має найвищу швидкодію, що наближається до 6 Гбіт/с на процесорі Pentium III 1000 MHz.

1.4. Висновок до першого розділу

Завдяки проведеному дослідженні найбільш перспективним алгоритмом обробки інформації є UMAC. Цей алгоритм має два режими роботи:

односпрямована геш-функція і функція вироблення MAC. Хоча для односпрямованої функції він має невисоку стійкість, але зате має найвищу швидкодію, що наближається до 6 Гбіт/с на процесорі Pentium III 1000 MHz. Завдяки застосуванню алгоритму AES, ймовірність пошкоджень зводиться до нуля. Тому перспективним напрямом є використання цього алгоритму у протоколах забезпечення автентифікації та цілісності інформації.

2. ДОСЛІДЖЕННЯ МЕТОДУ ОТРИМАННЯ ГЕШ-КОДУ НА ОСНОВІ ВИКОРИСТАННЯ UMAC-32

2.1. Загальна схема кодів автентичності повідомлень UMAC.

Розглянемо загальну схему формування кодів автентичності повідомлень з використанням алгоритму UMAC. Для цього в першу чергу необхідно проаналізувати основні аналітичні співвідношення, що описують внутрішню структуру і вживані перетворення при формуванні кодів автентичності повідомлень.

Код автентичності повідомлень (позначимо його Tag), враховуючи специфіку алгоритму UMAC, створюється виконуючи наступну функцію:

$$Tag = UMAC(K, M, Nonce, Taglen) = Y \oplus Pad,$$

де: K – таємний ключ;

$Keylen$ - стандартній довжина таємного ключа, що належить множині допустимих значень $\{16, 24, 32\}$байт);

M – значення, у вигляді масиву-рядка, від 1 до 2^{67} біт (2^{64} байт);

$Nonce$ – неповторюване (для всіх впроваджуються інформаційних повідомлень M) восьмибайтове число;

$Taglen$ – ціле число з множини допустимих значень $\{4, 8, 12, 16\}$, що задає довжину коду автентичності повідомлень Tag в байтах;

$Hash(K, M, Taglen)$ – функція стандартного гешування інформаційного повідомлення M з використанням таємного ключа K;

$PDF(K, Nonce, Taglen)$ – функція формування псевдовипадкової підкладки (Pad) по введеному значенню і секретному ключу;

«⊕» – побітове додавання (XOR) результату ключового гешування повідомлення і сформованої підкладки $Pad = PDF(K, Nonce, Taglen)$, тобто

$$Tag = Hash(K, M, Taglen) \oplus PDF(K, Nonce, Taglen).$$

Довжина геш-коду Y, підкладки Pad та коду Tag належить множині $\{32, 64, 96, 128\}$ біт.

Схема роботи алгоритму стандартного гешування для різних версій $UMAC$ показано на рисунку 2.1.

Рисунок 2.1 – Загальна схема універсальних геш-функцій UMAC-16/32

Отже базовою частиною формування MAC-коду є функція UHASH (універсальне гешування), що застосовується в обох версіях UMAC-16 та UMAC-32.

2.2. Перший етап гешування

На першому етапі гешування розбивається масив-рядка M величиною до 2^{64} байт на блоки M_i по 1024 байт, які в результаті стають функцією $NH(K_{L1}, M_i)$. Дані, яі отримали в результаті обчислення $Hash_{L1_i} = NH(K_{L1}, M_i)$ з'єднались у рядок $Y_{L1} = Hash_{L1}(K_{L1}, M)$, який є меншим ніж послідовність в 128 разів.

На першому етапі результатом гешування є цей рядок:

$$Y_{L1} = Hash_{L1}(K_{L1}, M) = NH(K_{L1}, M_0) \| NH(K_{L1}, M_1) \| .. \| NH(K_{L1}, M_{n-1}),$$

де $n = \left\lceil \frac{Length(M)}{1024} \right\rceil$, $\lceil x \rceil$ - ціла частина числа x, $Length(M)$ - довжина повідомлення M в байтах.

Значення функції $Hash_{L1_i} = NH(K_{L1}, M_i)$ обчислюється за наступним чином: інформаційний блок M_i розбивається на чотирьобайтові підблоки так, що

$$M_i = M_{i_1} \| M_{i_2} \| .. \| M_{i_t},$$

де $t = \left\lceil \frac{Length(M_i)}{4} \right\rceil$. У даному випадку $t = \left\lceil \frac{1024}{4} \right\rceil = 256$.

За аналогією ключова послідовність K_{L1} наведена у вигляді послідовностей чотирьобайтових підблоків:

$$K_{L1} = K_{L1_1} \| K_{L1_2} \| .. \| K_{L1_t}.$$

Після чого (беручи початковий стан $Hash_{L1_i} = 0$) для всіх $j = 1,9,17...,t-7$ виконуються наступні операції:

$$Hash_{L1_i} = Hash_{L1_i} +_{64} ((M_{i_{j+0}} +_{32} K_{L1_{j+0}}) \times_{64} (M_{i_{j+4}} +_{32} K_{L1_{j+4}})),$$
$$Hash_{L1_i} = Hash_{L1_i} +_{64} ((M_{i_{j+1}} +_{32} K_{L1_{j+1}}) \times_{64} (M_{i_{j+5}} +_{32} K_{L1_{j+5}})),$$
$$Hash_{L1_i} = Hash_{L1_i} +_{64} ((M_{i_{j+2}} +_{32} K_{L1_{j+2}}) \times_{64} (M_{i_{j+6}} +_{32} K_{L1_{j+6}})),$$
$$Hash_{L1_i} = Hash_{L1_i} +_{64} ((M_{i_{j+3}} +_{32} K_{L1_{j+3}}) \times_{64} (M_{i_{j+7}} +_{32} K_{L1_{j+7}})),$$

де $^{+64}$, $^{+32}$ -процес сумування за модулем 264 і 232; $^{\times 64}$ - процес множення за модулем 264.

У результаті обчислень формується восьмибайтове значення $Hash_{L1_i}$.

На першому етапі виконується розбиття вхідного повідомлення за допомогою метода функції NH гешування по наступній схемі (рисунок 2.2):

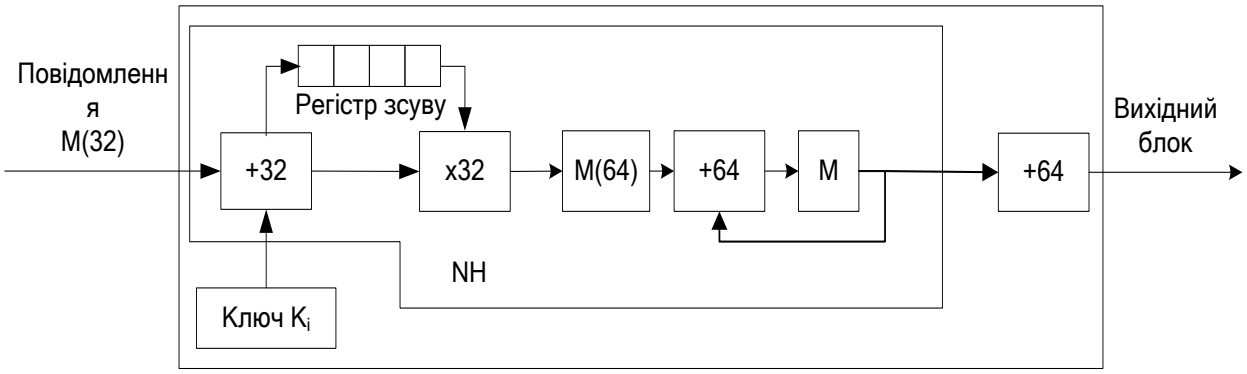

Рисунок 2.2 - Схема роботи першого етапу L1 Hash Function

Обчислюємо вручну ключі для першого етапу гешування:

$$n = \left\lceil \frac{Numbyte}{Blocklen} \right\rceil = \frac{1024 + 16 \times 3}{32} = \frac{1072}{32} = 33{,}5 \approx 33$$

$$=> i = 1,2,...,33$$

$$T_i = Index \| i$$

Для створення ключа, використовувався алгоритм Rijndael , що лежить в основі AES .

На першому етапі гешування $Index = 1, => \mathrm{T_i}$.

Дані, отримані при розрахунку приведені в таблиці 2.1.

Таблиця 2.1 - Розрахунок ключів на першому етапі гешування для UMAC

Блок даних T	Конкатенація $Index \| i$	Результат	Згенерований ключ	Блок даних T	Конкатенація $Index \| i$	Результат	Згенерований ключ
T_1	1 || 1	0000000100000001	329D	T_{18}	1 || 18	0000000100010010	0E02
T_2	1 || 2	0000000100000010	E43B	T_{19}	1 || 19	0000000100010011	0F3C
T_3	1 || 3	0000000100000011	6FC8	T_{20}	1 || 20	0000000100010100	1C04
T_4	1 || 4	0000000100000100	4974	T_{21}	1 || 21	0000000100010101	CEA0
T_5	1 || 5	0000000100000101	AD75	T_{22}	1 || 22	0000000100010110	4AB0
T_6	1 || 6	0000000100000110	EAA0	T_{23}	1 || 23	0000000100010111	0D85
T_7	1 || 7	0000000100000111	4E21	T_{24}	1 || 24	0000000100011000	8EFE
T_8	1 || 8	0000000100001000	C505	T_{25}	1 || 25	0000000100011001	B880
T_9	1 || 9	0000000100001001	F8A1	T_{26}	1 || 26	0000000100011010	03F3
T_{10}	1 || 10	0000000100001010	ABBE	T_{27}	1 || 27	0000000100011011	682E
T_{11}	1 || 11	0000000100001011	0F3F	T_{28}	1 || 28	0000000100011100	0720
T_{12}	1 || 12	0000000100001100	86A3	T_{29}	1 || 29	0000000100011101	5AE8
T_{13}	1 || 13	0000000100001101	C295	T_{30}	1 || 30	0000000100011110	D24D
T_{14}	1 || 14	0000000100001110	4DA4	T_{31}	1 || 31	0000000100011111	FB94
T_{15}	1 || 15	0000000100001111	8996	T_{32}	1 || 32	0000000100100000	9C41
T_{16}	1 || 16	0000000100010000	DCDE	T_{33}	1 || 33	0000000100100001	81FA
T_{17}	1 || 17	0000000100010001	9035				

Ключ формується методом конкатенації всіх розрахованих ключів:

$$K' = K'_1 \| K'_2 \| ... \| K'_n$$

2.3. Другий етап гешування

На другому етапі гешування застосовується поліноміальне ключове гешування [16; 25]. Результатом роботи цього етапу є обчислення геш-коду

$$Y_{L2} = Hash_{L2}(K_{L2}, Y_{L1}) = Poly(Wordbits, Maxwordrange, k, M_P),$$

тобто на вхід гешування другого етапа подається рядок $Y_{L1} = Hash_{L1}(K_{L1}, M)$.

Функція поліноміального гешування використовує такі початкові дані: $Wordbits \in [64, 128]$; $Maxwordrange$ – ціле число, менше $2^{Wordbits}$; k – залежить від ключа K_{L2} ціле число з діапазону $[0,..., prime(Wordbits) - 1]$, $prime(x)$ – найпросте число, менше 2^x; $M_P = Y_{L1} = Hash_{L1}(K_{L1}, M)$ - дані, що гешуються.

По специфікації алгоритму UMAC як $prime(x)$ використовуються наступні константи: $prime(36) = 2^{36} - 5$, $prime(64) = 2^{64} - 59$, $prime(128) = 2^{128} - 159$. Бітову довжину M_P позначимо $Bytelength(M_P)$. У залежності від довжини M_P використовуються наступні властивості:

- якщо величина значення M_P не більше 217 байт, тоді поліноміальне гешування $Poly$ виконується з параметрами $Wordbits = 64$; $Maxwordrange = 2^{64} - 2^{32}$; $k = k64$ – рядок, утворений першими вісьмома байтами ключа K_{L2} та спеціальною восьмибайтовою маскою;

- якщо величина значення M_P не менше 217 байт (але не більше 264 байт), тоді перші 217 байт даних обробляються функцією поліноміального гешування $Poly(64, 2^{64} - 2^{32}, k64, M_P)$, а що залишилися байти даних обробляються функцією $Poly$ з параметрами $Wordbits = 128$; $Maxwordrange = 2^{128} - 2^{96}$; $k = k128$ – рядок, утворений останніми 16 байтами ключа K_{L2} та спеціальною 16 байтною маскою.

Загешовані дані M_P розбиваються на блоки по $Wordbytes = Wordbits/8$ байт:

$$M_P = M_{P_1} \| M_{P_2} \| ... \| M_{P_n},$$

де $n = Bytelength(M_P)/Wordbytes$.

Результатом гешування є значення поліноміальної функції

$$Y_{L2} = \left(M_{P_n} + kM_{P_{n-1}} + \ldots + k^{n-1}M_{P_1} + k^n\right)\mathrm{mod}(p),$$

що знаходиться за ітеративною процедурою (для всіх $i = 1,2,\ldots,n$):

$$Poly_i = \left(kPoly_{i-1} + M_{P_i}\right)\mathrm{mod}(p), \quad Poly_0 = 1, \quad p = prime(Wordbits),$$

Використовуючи схему Горнера, маємо:

$$M_{P_n} + kM_{P_{n-1}} + \ldots + k^{n-1}M_{P_1} + k^n = (((k + M_{P_1})k + M_{P_2})k + \ldots + M_{P_{n-1}})k + M_{P_n}$$

Знайдені значення геш-функції $Y_{L2} = Poly_n$ є повним числом, яке належить такій послідовності $[0,\ldots, prime(Wordbits) - 1]$.

Обчислюємо показники ключів для другого етапу гешування:

$$n = \left\lceil \frac{Numbyte}{Blocklen} \right\rceil = \frac{24 \times 4}{32} = \frac{96}{32} = 3$$

$$=> i = 1,2,3$$

$$T_i = Index \| i$$

Для другого етапу хешування $Index = 2, => \mathrm{T}_i$:

Дані, отримані при розрахунку приведені в таблиці 2.2.

Таблиця 2.2 – Розрахунок ключів на другому етапі гешування для UMAC

Блок даних T	Конкатенація $Index \| i$	Результат	Згенерований ключ
T_1	2 || 1	0000001000000001	608E
T_2	2 || 2	0000001000000010	EC56
T_3	2 || 3	0000001000000011	9FF3

Загальна схема роботи алгоритму цього етапу універсальної геш-функції представлена на рисунок 2.4:

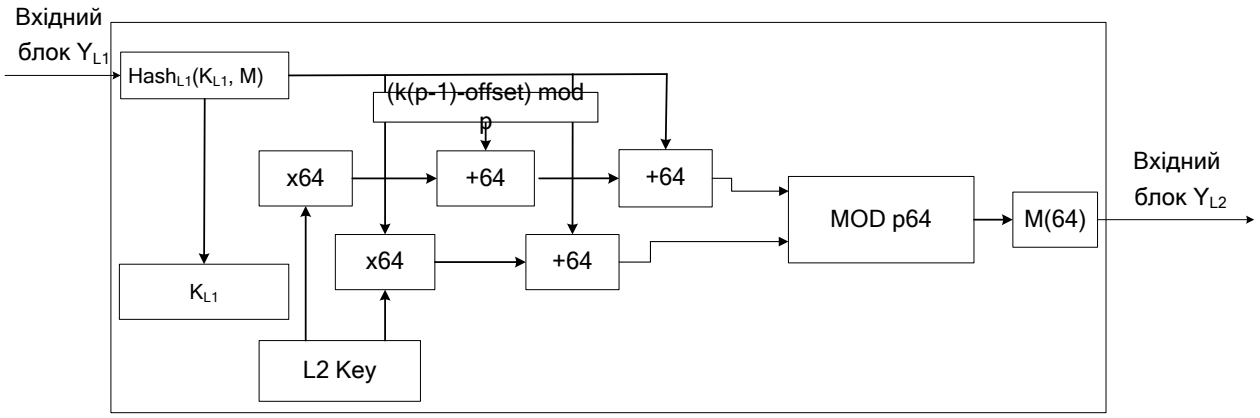

Рисунок 2.4 - Схема роботи другого етапу L2 Hash Function

2.4 Третій етап гешування

На третьому етапі гешування $Hash_{L3}\left(K_{L3_1}, K_{L3_2}, Y_{L2}\right)$ використовується результат поліноміального гешування і перетворює дані на його вхід довжиною до 16 байт в геш-код Y фіксованої довжини 32 біта.

В якості вихідних даних третього етапу гешування виступають дві ключові послідовності K_{L3_1} і K_{L3_2} довжиною 64 і 4 байти відповідно, а також вхідна 16 байтна послідовність Y_{L2}.

Загешовані дані Y_{L2} і ключова послідовність K_{L3_1} рівнопропорційно розбиваються на вісім блоків, кожен з яких постає як ціле число Y_{L2_i} і $K_{L3_{1i}}$, $i = 1,2,...,8$.

Геш-значення Y_{L3} обчислюється таким чином:

$$Y_{L3} = \left(\left(\left(\sum_{i=1}^{m} Y_{L2_i} K_{L3_{1i}} \right) \bmod(prime(36)) \right) \bmod(2^{32}) \right) xor(K_{L3_2}),$$

Розрахунок формування ключів:

Формування KL31:

$$n = \left\lceil \frac{Numbyte}{Blocklen} \right\rceil = \frac{64 \times 8}{16} = \frac{512}{16} = 32$$

$$=> i = 1,2,3,...,32$$

$$T_i = Index \,||\, i$$

Для третього етапі гешування $Index = 3, => T_i$:

Дані, отримані при розрахунку приведені в таблиці 2.3.

Таблиця 2.3 - Розрахунок першого ключа на третьому етапі гешування для UMAC

Блок даних T	Конкатенація $Index \,\|\|\, i$	Результат	Згенерований ключ	Блок даних T	Конкатенація $Index \,\|\|\, i$	Результат	Згенерований ключ
T_1	3 || 1	0000001100000001	8A13	T_{18}	3 || 17	0000001100010001	161F
T_2	3 || 2	0000001100000010	056A	T_{19}	3 || 18	0000001100010010	E6F4
T_3	3 || 3	0000001100000011	534D	T_{20}	3 || 19	0000001100010011	C0A0
T_4	3 || 4	0000001100000100	DE16	T_{21}	3 || 20	0000001100010100	6012
T_5	3 || 5	0000001100000101	91B2	T_{22}	3 || 21	0000001100010101	EC5A
T_6	3 || 6	0000001100000110	6B5E	T_{23}	3 || 22	0000001100010110	C606
T_7	3 || 7	0000001100000111	1E7A	T_{24}	3 || 23	0000001100010111	7823
T_8	3 || 8	0000001100001000	21ED	T_{25}	3 || 24	0000001100011000	F379
T_9	3 || 9	0000001100001001	C060	T_{26}	3 || 25	0000001100011001	7E88
T_{10}	3 || 10	0000001100001010	1B0B	T_{27}	3 || 26	0000001100011010	CE5E

Продовження таблиці 2.3

T_{11}	3 || 11	0000001100001011	6BE0	T_{28}	3 || 27	0000001100011011	027B
T_{12}	3 || 12	0000001100001100	458C	T_{29}	3 || 28	0000001100011100	B60B
T_{13}	3 || 13	0000001100001101	F7A8	T_{30}	3 || 29	0000001100011101	6928
T_{14}	3 || 14	0000001100001110	6F0D	T_{31}	3 || 30	0000001100011110	1CC4
T_{15}	3 || 15	0000001100001111	F91C	T_{32}	3 || 31	0000001100011111	E2C5
T_{16}	3 || 16	0000001100010000	4AF1	T_{33}	3 || 32	0000001100100000	5BA9

K_{L31} = K1 || K2 || K3 .. || K32

Формування K_{L32} :

$$n = \left\lceil \frac{Numbyte}{Blocklen} \right\rceil = \frac{4*8}{16} = \frac{32}{16} = 2 \quad => i = 1$$

$$T_i = Index \,||\, i$$

Для третього етапі гешування $Index = 4, => T_i$:

Дані, отримані при розрахунку приведені в таблиці 2.4.

Таблиця 2.4 – Розрахунок другого ключа на третьому етапі гешування для UMAC

Блок даних T	Конкатенація $Index \,\|\|\, i$	Результат	Згенерований ключ
T_1	4 || 1	0000010000000001	F530
T_2	4 || 2	0000010000000010	E33E

Призначення третього етапу заключається в зміні вхідного вектора B, довжиною 16 байт в рядок рівний 4 байт. Загальна схема роботи третього етапу L3 Hash Function приведена на рисунку 2.5.

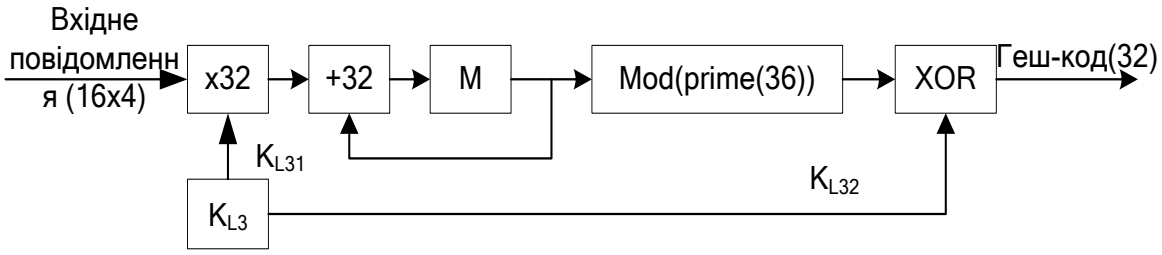

Рисунок 2.5 - Схема роботи третього слою L3 Hash Function

2.5 Висновок до другого розділу

Розглянули та проаналізували загальну схему формування кодів автентичності повідомлень з використанням алгоритму UMAC. Процес формування псевдовипадкової підкладки криптографічно стійким алгоритмом гарантує крипостійкість алгоритму UMAC на рівні стійкості застосовуваного криптоалгоритму. Алгоритм UMAC32 заснований на трьохслойних схемах гешування UHASH 32 і використовує для кодування залишку БСШ AES (Rijndael). Отже, розглянута схема формування UMAC має потенційно високі показниками ефективності.

3. ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ БЕЗПЕЧНОГО ГЕШУВАННЯ НА ОСНОВІ UMAC-32

3.1 Розробка програмного проєкту, що реалізує роботу функції універсального гешування UMAC-32

Для спрощення розробки програмного проекту його слід виконати на структурній схемі пристрою для формування МАС-коду з використанням геш-функції UMAC-32. Структурна схема пристрою для формування МАС-коду приведена на рисунку 3.1.

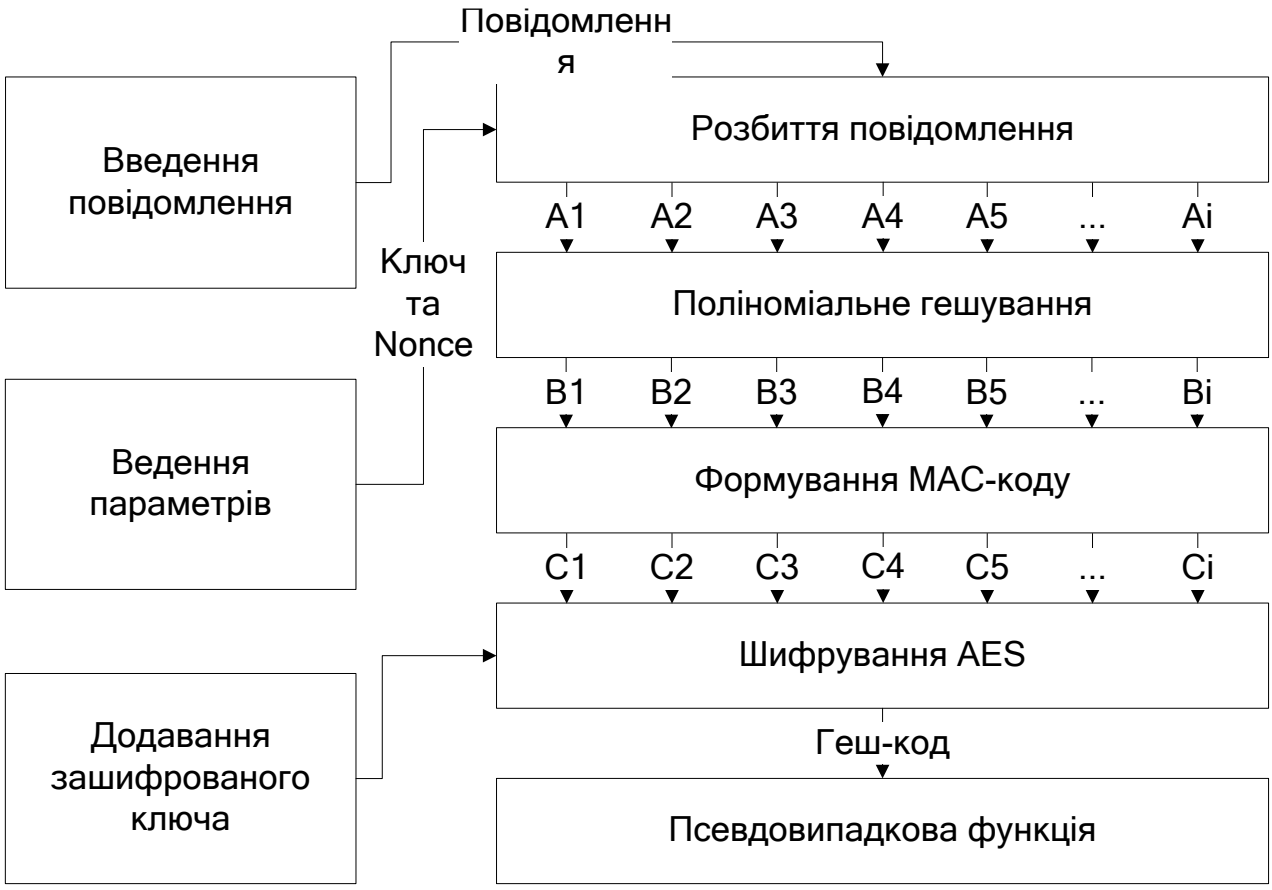

Рисунок 3.1 - Структурна схема пристрою для формування МАС-коду

Згідно представленої на рисунку 3.1 структурної схеми на вхід програмного продукту поступає вхідне повідомлення за допомогою пристрою

введення повідомлення та параметри геш-функції (ключ та Nonce) за допомогою пристрою введення параметрів. За допомогою пристрою розбиття повідомлення відбувається розділення повідомлення на блоки даних, які потім потрапляють до пристрою поліноміального гешування. З цього пристрою сформовані блоки даних потрапляють на вхід пристрою формування коду, після обробки в якому блок геш-коду (залишок) шифрується за допомогою пристрою шифрування у режимі блочно-симетричного шифру AES. Зашифрований геш-код потрапляє на пристрій виведення результату.

Після розробки структурної схеми слід обумовити вимоги до програмного проекту, що буде розроблено.

До програмного проекту сформовано наступні функціональні вимоги:

- програма повинна мати графічний інтерфейс;

- у програмі повинні бути передбачене формування, видалення і збереження сформованого геш-коду у файл;

- програма повинна генерувати необхідну кількість геш-кодів для подальшого використання в програмі NIST Statistical Test Suite;

Вимоги до продуктивності наступні: час запуску системи не більше 3 секунд, час обробки запиту на формування геш-коду – не більше 3 секунд, на генерування послідовності 10^6 біт – не більше 3 хвилин.

Програма розроблена на мові програмування С в середовищі C++ Builder 6.0 з використанням об’єктно-орієнтованої технології для реалізації графічного інтерфейсу програми. Повний текст програмно-реалізованої геш-функції знаходиться в дод. А.

Для запуску додатку необхідно запустити файл UmacApp.exe. Головне вікно додатку, що відкрилося, має вигляд, як показано на рисунку 3.2.

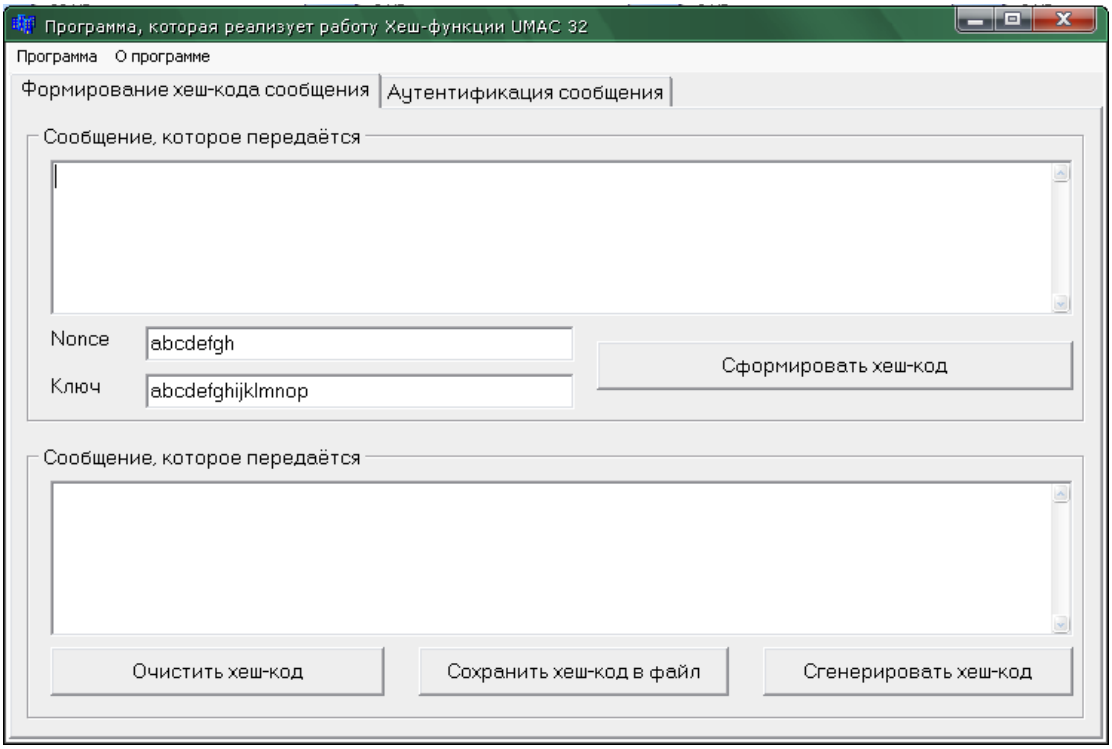

Рисунок 3.2 - Головне вікно додатку

Для отримання геш-коду вхідного повідомлення слід ввести дані в текстове поле і вказати ключ (key) і новизну (Nonce), як показано на рисунку 3.3.

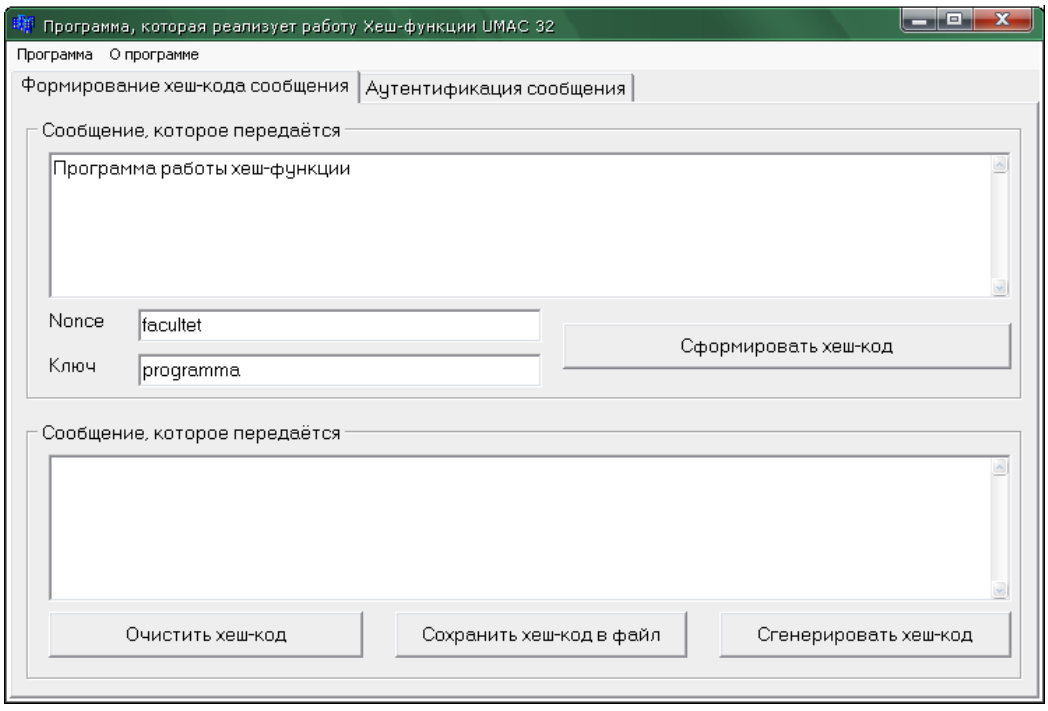

Рисунок 3.3 - Введення параметрів для отримання геш-коду

Потім необхідно натиснути кнопку “Сформировать хеш-код”. Отриманий геш-код повідомлення можна зберегти у файл із текстового поля у форматі bin, або очистити саме повідомлення і його геш-код в обох текстових полях. Результат отримання геш-коду показаний на рисунку 3.4.

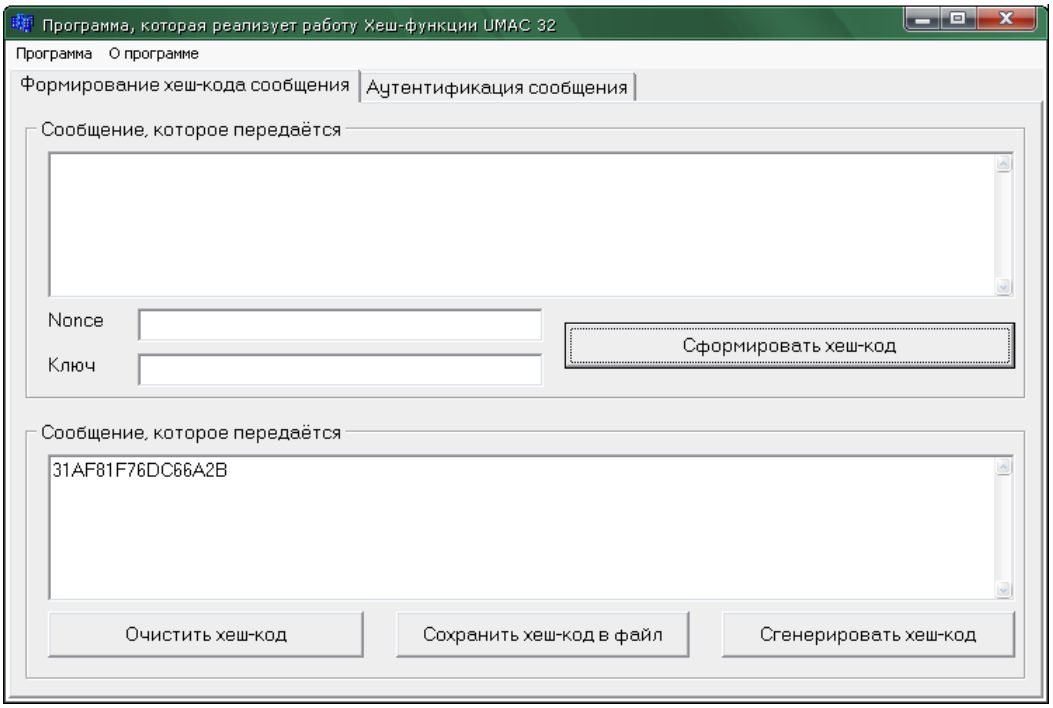

Рисунок 3.4 - Отриманий геш-код повідомлення

На другій вкладці програми можна відстежити швидкість верифікації переданих повідомлень за допомогою автентифікації. Для цього необхідно скористатися кнопкою “Начать аутентификацию” як показано на рисунку 3.5.

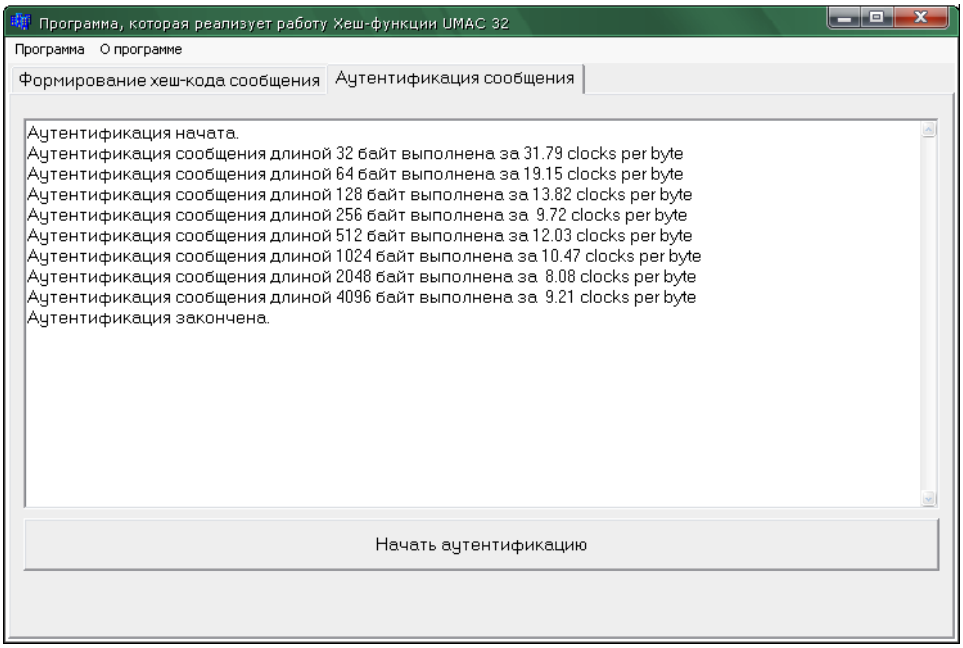

Рисунок 3.5 - Результат автентифікації повідомлень

На основі використання розробленого програмного проекту був проведений аналіз швидкості автентифікації MAC-кодів за допомогою UMAC-32 з різною довжиною дайджесту (профілю), який представлено в таблиці 3.1.

Таблиця 3.1 - Швидкість автентифікації повідомлень

Довжина повідомлення, байт								
Час автентифікації, cpb	32	64	128	256	512	1024	2048	4096
	32,81	19,07	12,7	9,31	7,73	6,91	6,87	6,65

Швидкість формування MAC-коду свідчить, що час його формування зменшується при збільшенні довжини дайджесту (профілю). Таким чином розроблений алгоритм дозволяє формувати 10^9 біт/сек MAC-коди від повідомлень довільної довжини. На рис. 3.6 представлений графік швидкості автентифікації MAC-кодів сформованих для повідомленнями довільної довжини.

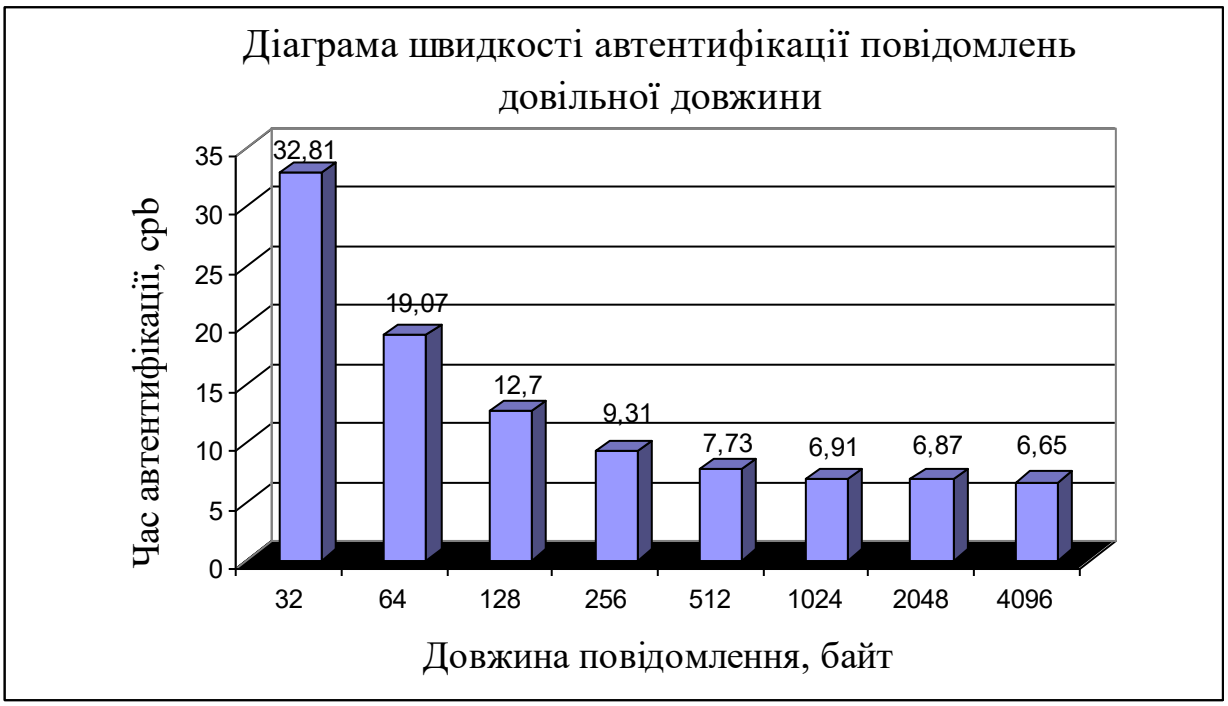

Рисунок 3.6 - Діаграма швидкості автентифікації повідомлень довільної довжини

Побудований графік дозволяє зробити висновок про те, що використана у програмному продукті геш-функція універсального гешування забезпечує достатню швидкість вироблення та автентифікацію MAC-коду для повідомлень довільної довжини.

3.2 Експериментальні дослідження статистичної безпеки функції універсального гешування UMAC-32

Дослідження безпеки ключових геш-функцій проводилися відповідно до методики тестування NIST SP 800-22, рекомендованої Національним інститутом по стандартизації і технологіям США.

Для проведення тестування були взяті наступні параметри:

– довжина тестованої послідовності $n = 10^6$ біт;

– кількість тестованих послідовностей $m = 100$;

– рівень значимості $\alpha = 0{,}01$.

Таким чином, обсяг вибірки, що тестується, становить:

$$N = 10^6 \times 100 = 10^8 \text{ біт};$$

Пакет NIST STS містить у собі 16 статистичних тестів. Але в залежності від початкових значень обчислюємо 189 значень імовірності *P,* які є результатом роботи окремих тестів, таким чином, статистичний портрет генератора становить 18900 значень ймовірностей P.

Результати тестування ключових та безключових геш-функцій приведені в таблиці 3.2.

Таблиця 3.2 - Результати тестування ключових та безключових геш-функцій

Статистичні дані	UMAC	MASH-1
Кількість тестів, у яких тестування пройшло 99% послідовностей	173	101
Кількість тестів, у яких тестування пройшло 96% послідовностей	198	47
Кількість тестів, у яких значення ймовірності $P \leq 0{,}01$		4
Кількість тестів, у яких значення ймовірності $P \leq 0{,}001$		16
Кількість тестів, у яких значення ймовірності $P \leq 0{,}05$	1	4
Припустиме значення частки проходження тесту для вибірки розміром 100 двійкових послідовностей.	0,96015	
Припустиме значення частки проходження тесту для вибірки розміром 71 двійкових послідовностей (Random-Excursion)	0,954575	

На рисунку 3.7 представлено статистичний портрет програмної реалізації ключової геш-функції MASH-1. Статистичний портрет представляє з себе діаграму ймовірностей проходження відповідних статистичних тестів.

Рисунок 3.7 - Статистичний портрет програмної реалізації геш-функції MASH-1

Статистичний портрет програмної реалізації безключової геш-функції MASH-1 відповідає запропонованим вимогам – по всім тестам позитивно пройшло більше 99% послідовностей, але один тип атаки на даний алгоритм дозволяє виявити колізію та скористатися нею.

Статистичний портрет програмної реалізації розробленої функції універсального гешування приведено на рисунку 3.9.

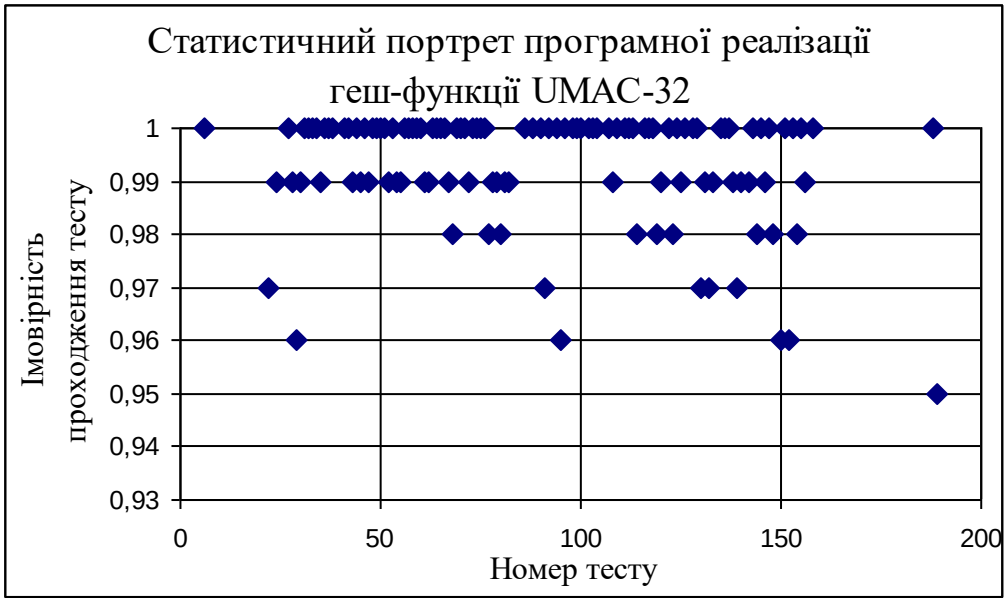

Рисунок3.9 - Статистичний портрет програмної реалізації UMAC-32

Результати статистичного портрету UMAC-32 наведені в дод. Б.

У таблиці 3.3 зведені результати тестування генераторів програмної реалізації ключових геш-функцій MASH-1, UMAC-32.

Таблиця 3.3 - Результати тестування генераторів

Генератор	Кількість тестів, у яких тестування пройшло більше 99% послідовностей	Кількість тестів, у яких тестування пройшло більше 96% послідовностей
MASH-1	101 (53%)	47 (24%)
UMAC 32	167 (88%)	189 (100%)

Генератори на UMAC і MASH -1 мають гарні статистичні властивості. За результатами досліджень, що основувались на стистичній безпеці видно, що ключові геш-функції забезпечують проходження тестів з більшою ймовірністю, чим алгоритми безключових геш-функцій.

3.3 Висновок до третього розділу

Аналіз геш-функцій показав, що за швидкістю формування геш-коду переможцем є алгоритм UMAC-32. Проведений аналіз алгоритмів ключових та безключових геш-функцій за стійкістю та швидкодією формування геш-коду в інформаційних системах показав, що в якості алгоритму забезпечення цілісності та автентичності інформації пропонується використовувати геш-функцію UMAC-32, яка дозволяє формувати геш-код від інформаційної довжини даних з максимальною швидкістю гешування, близькою до 10^9 біт/сек на процесорі Pentium 4 за дозволений час (час формування цифрового підпису на еліптичних кривих ДСТУ-4145).

Механізм забезпечення цілісності та автентичності інформації в інформаційних системах, побудований на геш-функції UMAC-32 дозволяє

вирішити протиріччя та забезпечити оперативне (своєчасне) формування MAC-коду для повідомлень довільної довжини.

Проведені дослідження показали, що алгоритм UMAC-32, побудований на основі універсальних класів геш-функцій, має максимальну швидкість гешування, проте істотним недоліком є втрата універсальності після накладення псевдовипадкової підкладки.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ХОРОНИ ПРАЦІ

4.1 Значення адаптації в трудовому процесі.

Праця людини безпосередньо пов'язана із виробничим середовищем. Працівник може нормально здійснювати трудову діяльність лише тоді, коли умови зовнішнього середовища відповідають оптимальним. Якщо вони стають несприятливими та на протидію їм організм людини включає спеціальний механізм, який зберігає постійність внутрішнього середовища, або змінює його в межах допустимого. Такий механізм називається адаптацією. Адаптація є важливим засобом попередження травмування, виникнення нещасних випадків у трудовому процесі і відіграє значну роль в охороні праці.

Адаптація - це динамічний процес пристосування організму та його органів до мінливих умов зовнішнього середовища.

Адаптація в трудовій діяльності поділяється на фізіологічну, психічну, соціальну та професійну:

- Фізіологічна адаптація - це сукупність фізіологічних реакцій, які є в основі пристосування організму до змін оточуючого середовища і направлені на збереження відносної постійності його внутрішнього середовища. Суть механізму адаптації полягає у змінах меж чутливості аналізаторів, розширенні діапазону фізіологічних резервів організму та зміні в певних межах параметрів фізіологічних функцій (підвищується стійкість організму до холоду, тепла, недостачі кисню, змін барометричного тиску та ін.) Фізіологічна адаптація до праці має активний характер і за сприятливих умов виробничого середовища та оптимальних навантажень веде до підвищення стійкості та продуктивності організму, збільшення його резервних можливостей, зменшення захворювань і травматизму.

- Психічна адаптація - це процес встановлення оптимальної відповідності особистості до оточуючого середовища в процесі діяльності. Психічна адаптація в процесі праці залежить від психічних властивостей

працівника, його психічного стану, психологічних реакцій на стреси, що виникають на роботі, кваліфікації та культури людини, особливостей професійної діяльності, конкурентних умов праці тощо.

- Соціальна адаптація - це пристосування працюючої людини до системи відносин у робочому колективі з його нормами, правилами, традиціями, ціннісними орієнтаціями. При несприятливому протіканні соціальної адаптації підвищується рівень стресу на роботі, наслідки якого позначаються на поведінці працівника та можуть призвести до між особових конфліктів, нещасних випадків.

- Професійна адаптація - це адаптація до трудової діяльності з усіма її складовими і адаптація до робочого місця, знарядь та засобів праці, об'єктів та предметів праці, особливостей технологічного процесу, головних параметрів роботи тощо. Професійна адаптація виражається у розвитку стійкого позитивного ставлення працівника до своєї професії, певного рівня оволодіння ним специфічними навичками та уміннями у формуванні необхідних для якісного виконання роботи властивостей.

У процесі адаптації працівник проходить наступні стадії:

- стадія ознайомлення, на якій працівник одержує інформацію про нову ситуацію в цілому, про критерії оцінки різних дій, про еталони, норми поведінки;

- стадія пристосування, на цьому етапі працівник переорієнтується, визнаючи головні елементи нової системи цінностей, але поки продовжує зберігати багато своїх установок;

- стадія асиміляції, коли здійснюється повне пристосування до середовища, ідентифікація з новою групою. Ідентифікація, коли особисті цілі працівника ототожнюються з цілями трудової організації, підприємства, фірми і т.д. По характері ідентифікації розрізняють три категорії працівників: байдужні, частково ідентифіковані і цілком ідентифіковані. Ядро будь-якої трудової організації складають цілком ідентифіковані працівники. І кінцеві результати такої трудової організації завжди високі.

Швидкість адаптації залежить від багатьох факторів. Нормальний термін адаптації для різних категорій працівників складає від 1 року до 3 років. Невміння ввійти в трудову організацію (колектив), адаптуватися в ній викликає явище виробничої і соціальної дезорганізації. Більшість соціально-трудових відносин виступає для людини у формі соціально-трудових процесів, у яких вона бере участь протягом трудового життя, починаючи зі свого першого робочого місця та участі в процесі виробничої адаптації.

Виробнича адаптація. У соціології праці останніми роками зростає увага до проблем адаптації людини у сфері праці, що є свідченням визнання тієї суттєвої ролі, яку відіграє процес адаптації в трудовій діяльності людини. Водночас розуміння сутності адаптації утруднене існуючими ще функціональними підходами до адаптації працівника з виокремленням лише деяких сторін його взаємодії з виробництвом. Такий підхід залишається типовим для багатьох соціологів, у тому числі й західних.

Необхідно підходити до дослідження адаптації особистості працівника, розглядаючи його як цілісну людину в різноманітності видів і форм виконуваної ним діяльності. З огляду на цілісне розуміння сутності працівника виробнича адаптація не обмежується професійною сферою, а охоплює сукупність соціально-трудових відносин, що зумовлює її структуру. До основних структурних елементів виробничої адаптації відносять професійну, організаційну, матерально-побутову, соціально-психологічну та адаптацію у сфері дозвілля. В основі механізму виробничої адаптації лежить адаптивна потреба індивіда, опосередкована взаємодією з потребою його в трудовій самореалізації. При цьому формується, з одного боку, орієнтаційний мотив поведінки, що спонукає індивіда до здобуття інформації про трудову ситуацію, розширення контактів із соціально-виробничим середовищем, оцінки характеру адаптивної ситуації. З другого боку, мотив опанування конкретної трудової діяльності і досягнення оптимальної взаємодії з виробництвом опосередковує зміст інформації і спрямованість особистісних контактів. В результаті складної полімотивації здійснюється виробнича адаптація, яка враховує можливості

реалізації на даному підприємстві життєвих цілей працівників. Успішне (або утруднене) проходження виробничої адаптації молодого працівника залежить від того, наскільки сприятливі (або несприятливі) умови склалися для задоволення його адаптивної потреби. З огляду на структуру останньої до таких умов належать певний рівень взаємної інформації між індивідом і виробництвом, взаємних контактів; порівняння життєвих цілей індивіда з завданнями даного підприємства, а також наявність на виробництві умов для успішної трудової діяльності індивіда.

Отже, кожен із розглянутих видів адаптації впливає на працездатність та здоров'я працівника, формує у нього певний рівень чутливості та стійкості до психоемоційних перевантажень, внаслідок розвитку яких може істотно змінитися надійність професійної діяльності. Наявність і повнота прояву сукупності умов, необхідних для проходження адаптації, будуть визначати як її ефективність, динамізм, так і межі самого процесу. Таким чином, адаптація працівника являє собою процес його взаємодії з соціально-виробничим середовищем з метою оволодіти новою трудовою ситуацією.

4.2 Організація служби охорони праці на підприємстві

Закон України «Про охорону праці» передбачає, що роботодавець зобов’язаний створити на робочому місці умови праці та забезпечити додержання вимог законодавства щодо прав працівників у галузі охорони праці. З цією метою роботодавець забезпечує функціонування системи управління охороною праці та несе безпосередню відповідальність за порушення вимог з охорони праці на підприємстві.

На підприємстві з кількістю працюючих менше 50 осіб функції служби охорони праці можуть виконувати в порядку сумісництва особи, які мають виробничий стаж не менше трьох років і пройшли навчання з охорони праці.

На підприємстві з кількістю працюючих 50 і більше осіб роботодавець створює службу охорони праці відповідно до типового

положення, що затверджується центральним органом виконавчої влади, що забезпечує формування державної політики у сфері охорони праці.

На підприємстві з кількістю працюючих менше 20 осіб для виконання функцій служби охорони праці можуть залучатися сторонні спеціалісти на договірних засадах, які мають відповідну підготовку.

Служба охорони праці підпорядковується безпосередньо роботодавцю. Керівники та спеціалісти служби охорони праці за своєю посадою і заробітною платою прирівнюються до керівників і спеціалістів основних виробничо-технічних служб.

Спеціалісти служби охорони праці мають право:

- видавати керівникам структурних підрозділів підприємства обов'язкові для виконання приписи щодо усунення наявних недоліків, одержувати від них необхідні відомості, документацію і пояснення з питань охорони праці;

- вимагати відсторонення від роботи осіб, які не пройшли передбачених законодавством медичного огляду, навчання, інструктажу, перевірки знань і не мають допуску до відповідних робіт або не виконують вимог нормативно-правових актів з охорони праці;

- зупиняти роботу виробництва, дільниці, машин, механізмів, устаткування та інших засобів виробництва у разі порушень, які створюють загрозу життю або здоров'ю працюючих;

- надсилати роботодавцю подання про притягнення до відповідальності працівників, які порушують вимоги щодо охорони праці;

- за поліпшення стану безпеки праці вносити пропозиції про заохочення працівників за активну працю.

Фінансування охорони праці здійснюється роботодавцем. Фінансування профілактичних заходів з охорони праці, виконання загальнодержавної, галузевих та регіональних програм поліпшення стану безпеки, гігієни праці та виробничого середовища, інших державних програм, спрямованих на запобігання нещасним випадкам та професійним захворюванням,

передбачається, поряд з іншими джерелами фінансування, визначеними законодавством, у державному і місцевих бюджетах.

На підприємствах, що утримуються за рахунок бюджету, розмір витрат на охорону праці встановлюється у колективному договорі з урахуванням фінансових можливостей підприємства, установи, організації.

Суми витрат з охорони праці, що належать до валових витрат юридичної чи фізичної особи, яка відповідно до законодавства використовує найману працю, визначаються згідно з переліком заходів та засобів з охорони праці, що затверджується Кабінетом Міністрів України.

Роботодавець зобов'язаний інформувати працівників або осіб, уповноважених на здійснення громадського контролю за дотриманням вимог нормативно-правових актів з охорони праці, та Фонд соціального страхування України про стан охорони праці, причину аварій, нещасних випадків і професійних захворювань і про заходи, яких вжито для їх усунення та для забезпечення на підприємстві умов і безпеки праці на рівні нормативних вимог.

Працівникам забезпечується доступ до інформації та документів, що містять результати атестації робочих місць, заплановані роботодавцем профілактичні заходи, результати розслідування, обліку та аналізу нещасних випадків і професійних захворювань і звіти з цих питань, а також до повідомлень, подань та приписів органів державного нагляду за охороною праці.

Органи державного управління охороною праці у встановленому порядку інформують населення України, працівників про реалізацію державної політики з охорони праці, виконання загальнодержавної, галузевих чи регіональних програм з цих питань, про рівень і причини аварійності, виробничого травматизму і професійних захворювань, про виконання своїх рішень щодо охорони життя та здоров'я працівників.

ВИСНОВКИ

1. Щоб зберегти автентичність, цілісність та конфіденційність використовують спеціальні механізми: шифрування (симетричне і несиметричне), електронний цифровий підпис, функції гешування. Альтернативою ЕЦП (електронний цифровий підпис) є код автентичності повідомлень (MAC-коди), які дозволяють інтегровано забезпечити конфіденційність і автентичність переданих повідомлень. Серед MAC-кодів особливе місце займають MAC-коди UMAC, побудовані на основі універсальних класів геш-функцій.

2. Аналіз методів побудови універсальних класів геш-функцій показав, що спеціальний код UMAC (код аутентифікації даних), базується на основі сімейства загальновживаних геш-функцій, а це забезпечує максимальну швидкодію утворення геш-коду будь-якого повідомлення. Методи універсального гешування не дозволяють забезпечити криптографічну стійкість до атак зловмисника. Всі функції універсального гешування застосовуються разом з алгоритмами шифрування, в результаті чого забезпечується стійкість, але втрачається властивість універсальності алгоритму, і як належить зменшується швидкодія алгоритму.

Практичне використання алгоритмів завадостійкого кодування при формуванні швидких алгоритмів формування MAC-кодів на першому слої геш функцій NH на основі універсальних класів в геш-функції Uhash дозволяє забезпечити оперативність формування геш-кодів повідомлень довільної довжини.

3. Проведений аналіз механізмів забезпечення автентичності, цілісності та конфіденційності показав, що зі збільшенням обсягів оброблюваних даних в комп’ютерних мережах і системах, а також бурхливим розвитком обчислювальної техніки висувають нові вимоги до спеціальних механізмів за швидкодією та надійністю даних, які обробляються, що призводить до необхідності розвитку інтегрованих механізмів безпеки, таким чином

застосування алгоритмів UMAC з усуненням виявлених недоліків є актуальною науково-технічною задачею, яка дозволяє забезпечити максимальну швидкодію оброблюваних даних (109 біт/с) і криптостійкість отриманих геш-кодів, не нижче криптостійкості використовуваного блоково-симетричного шифру AES.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1.1. Informacionnaja tehnologija. Kriptograficheskaja zashhita informacii. Funkcija heshirovanija : GOST 34.311-95. − [Dejstvujushhij s 1998-04-16] − Kiev. Gosstandart Ukrainy, 1998. – 1 s. – (Nacional'nyj standart Ukrainy).

2.2. Ukrainskij resurs po bezopasnosti [Elektronnij resurs]. – Rezhim dostupa: http://kiev-security.org.ua.

3.3. DSTU 7564–2014. Informacijni tehnologiï. Kriptografichnij zahist informaciï. Funkcija reshuvannja [Tekst]. – K. : Derzhstandart Ukraïni, 2014. – 39 s.

4.4. Avolio F. M. Zashhita informacii na predprijatii // Seti i sistemy svjazi. / F. M. Avolio, G. Shipli. – 2000. − №8 – S. 91−99.

5.5. Vervejko V. N. Funkcii heshirovanija: klassifikacija, harakteristika i sravnitel'nyj analiz / Vervejko V. N., Pushkarev A. I., Cepurit T. V. – HNURE : Har'kov, 2002. – 68 c.

6.6. Gorbenko I.D. Analiz kanalov ujazvimosti sistemy RSA // Bezopasnost' informacii. / Gorbenko I.D., Dolgov V.I., Potij A.V. – 1995.-№2. – S. 22–26.

7.7. Diffi U. Zashhishhennost' i imitostojkost' // Vvedenie v kriptografiju / U. Diffi, M. Hellman.– 1979. – №3 − S. 79–109.

8.8. Dolgov V.I. O nekotoryh podhodah k postroeniju bezuslovno stojkih kodov autentifikacii korotkih soobshhenij // Upravlenie i svjaz'. / V.I. Dolgov, V.N. Fedorchenko. – 1996. – №4– S. 47-51.

9.9. Domarev V.V. Zashhita informacii i bezopasnost' komp'juternyh sistem. – Kiev : Izdatel'stvo "DiaSoft", 1999. – 480 s.

10.10. Євseєv S. P. Mehanizmy obespechenija autentichnosti bankovskih dannyh vo vnutriplatezhnyh sistemah komercheskogo banka : zbirnik naukovih statej HNEU. / Євseєv S. P., Chevardin V. E., Radkovskij S. A. – Harkiv : HNEU, 2008. – Vip. 6. – 40–44 s.Задірака В. К. Методи захисту банківської інформації. / Задірака В. К., Олесюк О. С., Недашковський Н. О. – Київ : Вища школа, 1999. – 264 с.

11. Євсеєв С.П. Остапов С.Е., Король О.Г. Кібербезпека: сучасні технології захисту Навчальний посібник для студентів вищих навчальних закладів. Львів: “Новий Світ- 2000”, 2019. – 678.

12. Євсеєв С.П., Йохов О.Ю., Король О.Г. Гешування даних в інформаційних системах. Монографія Харків: Вид. ХНЕУ, 2013. – 312 с.

13. 13. Кузнєцов А. А. Аналіз механізмів забезпечення безпеки банківської інформації у внутрішньоплатіжних системах комерційного банку: Матеріали з міжнародної науково-практичної конференції «Безпека та захист інформації в інформаційних та телекомунікаційних системах»: зб. наук. статей "Управління розвитком". / Кузнєцов А. А., Король О. Г., Ткачов О. М. − ХНЄУ. № 6 - Х.: 2008. - 28 - 35 с.

14. Потій А.В. Статистичне тестування генераторів випадкових і псевдовипадкових чисел з використанням набору статистичних тестів NIST STS. – www.kiev-security.org.ua

15. Р. В. Грищук, В. В. Охрімчук, “Напрямки підвищення захищеності комп’ютерних систем та мереж від кібератак”, II Міжнар. наук.-практ. конф. “Актуальні питання забезпечення кібербезпеки та захисту інформації” (Закарпатська область, Міжгірський район, село Верхнє Студене, 24-27 лют. 2016 р.). – К. : Видавництво Європейського університету, 2016 с. 60 – 61.

16. S. Evseev, and B. Tomashevsky, “Two-factor authentication methods threats analysis”, Радіоелектроніка, інформатика, управління, Вип. 1(32), с. 52 – 60, 2015.

17. Самбурська Т. Ю. Аналіз методів хешування інформації для забезпечення цілісності й автентичності в комп'ютерних системах і мережа. : Матеріали міжнародної науково-практичної конференції «Актуальні проблеми науки і освіти молоді: теорія, практика, сучасні рішення» 16 квітня 2009 р. : зб. наук. статей «Управління розвитком». ХНЕУ. № 4 – Х. : 2009. – 73 – 77 с.

18. . В. Грищук, “Атаки на інформацію в інформаційно-комунікаційних системах”, Сучасна спеціальна техніка, №1(24), с.61 – 66. 2011.

19. Термінологія в галузі захисту інформації в комп'ютерних системах від несанкціонованого доступу. НД СТЗІ 1.1-003-99. − Чинний від 28.04.1999. − К. : Держстандарт України, 1999. — 24 с.

20. Чмора А. Л. Современная прикладная криптография. – Москва, 2002. – 508 с.

21. Шеннон К.Э. Теория связи в секретных системах. Работы по теории информации и кибернетике / К.Э. Шеннон. – М. : ИЛ., 1963. − 333-402 с.

22. Шипли Г. Основы безопасности ИТ // Сети и системы связи. / Г. Шипли. – М., 2003 – №4 – С. 78–82.

23. Bierbrauer J. On families of hash function via geometric codes and concatenation / J. Bierbrauer, T. Johansson, G. Kabatianskii // Advances in Cryptology – CRYPTO 93, Lecture Notes in Computer Science. – 1994 – № 773 – P. 331–342.

24. Carter J. L. Universal classes of hash functions / J.L.Carter, M.N.Wegman // Computer and System Scince. – 1979 – №18 – P. 143–154.

25. Chor B. A Knapsack-type public-key cryptosystem based on arithmetic in finite fields / B.Chor /Advances in Cryptology. – NY: Springer–Verlag, 1985 – 54p.

26. Diffi W. The first Ten Years of Public-Key Cryptography/ W. Diffi/ Computer Since. – 1988 – №5 – P. 21.

27. ISO 7498-2:1989 – Information technology – Open System Interconnection- Basic reference model-Part 2: Security architecture –32 p.

28. ISO/IEC 10181-(1-7):1996 – Information technology – Open System Interconnection – Security framework for open systems. –56 p.

29. Krawczyk H. LFSB-based Hashing and Authenticator./ H. Krawczyk/ Proceedings of CRYPTO Notes in Computer Science. – 1994 – №80 – P. 129 – 139.

30. McElliece R.J. A public-key cryptosystem based on algebraic coding theory/ R.J. McElliece - NY: Springer-Verlag, 1978. – 116 p.

31. Pieprzyk J.P. On public-key cryptosystems built using polynomial rings. In Advanced in Cryptography-Eurocrypt/ J.P.Pieprzyk./ – NY: Springer-Verlag, 1985 – 80 p.

32. Preneel B., Biryukov A., «New European Schemes for Signature, Integrity and Encryption» Final report of European project number IST-1999-12324, NESSIE, April 2004. – P. 487– 623

33. Simmons G.J. Authentication theory/coding theory in Cryptology/ G.J.Simmons/ Computer Science. – 1985 – №96 – P. 411–431.

34. Simons G.J. An impersonation-proof identity verification scheme/ G.J.Simons. / Computer Science. – 1988 – №87 – P. 211–215.

35. Smid M.E. The Past and Future / M.E.Smid, D.K.Branstad. / Computer Science. – 1988 – №76 – P. 122–132.

36. Wegman M. N. New hash functions and their use in authentication and set equality / M. N.Wegman, J. L. Carter. /Computer and System Scince. – 1981 – № 22 – P. 265–279.

Додаток А
Лістинг коду програми

```
#include <vcl.h>
#pragma hdrstop
#include "FormUmac.h"
#include "umac.h"
#include "rijndael-alg-fst.h"
#include <stdio.h>
#include <time.h>
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
#define HZ (2000e6)
#define UMAC_OUTPUT_LEN     8      /* 64 bytes   */
typedef unsigned char      UINT8;  /* 1 byte   */
typedef unsigned short     UINT16; /* 2 byte   */
typedef unsigned int       UINT32; /* 4 byte   */
typedef unsigned long long UINT64; /* 8 bytes  */
typedef unsigned long      UWORD;  /* Register */
AnsiString result;
static double run_cpb_test(umac_ctx_t ctx, int nbytes, char *data_ptr,
                           int data_len)
{
    clock_t ticks;
    double secs;
    char nonce[8] = {0};
    char tag[UMAC_OUTPUT_LEN+1] = {0}; /* extra char for null terminator */
    unsigned long total_mbs;
    unsigned long iters_per_tag, remaining;
    unsigned long tag_iters, i, j;
    if (nbytes <= 16)
        total_mbs = 5;
    if (nbytes <= 32)
        total_mbs = 30;
    else if (nbytes <= 64)
        total_mbs = 400;
    else if (nbytes <= 256)
        total_mbs = 800;
    else if (nbytes <= 1024)
        total_mbs = 1600;
    else
        total_mbs = 2500;
    tag_iters = (total_mbs * 1024 * 1024) / (nbytes) + 1;
    if (nbytes <= data_len) {

        i = tag_iters;
        umac(ctx, data_ptr, nbytes, tag, nonce);
        ticks = clock();
        do {
            umac(ctx, data_ptr, nbytes, tag, nonce);
            nonce[7] += 1;
```

```
  } while (--i);
        ticks = clock() - ticks;

    } else {
        i = tag_iters;
        iters_per_tag = nbytes / data_len;
        remaining = nbytes % data_len;
        umac_update(ctx, data_ptr, data_len);
        umac_final(ctx, tag, nonce);
        ticks = clock();
        do {
            j = iters_per_tag;
            do {
                umac_update(ctx, data_ptr, data_len);
            } while (--j);
            if (remaining)
                umac_update(ctx, data_ptr, remaining);
            umac_final(ctx, tag, nonce);
            nonce[7] += 1;
        } while (--i);
        ticks = clock() - ticks;

    }
    secs = (double)ticks / CLOCKS_PER_SEC;
    return (secs * (HZ/(tag_iters*nbytes)));
}
void pbuf(void *buf, int n)
{
        result = "";
 UINT8 *cp = (UINT8 *)buf;
 if (n <= 0 || n >= 30)
 n = 30;
        char buffer[10];
        for (int i = 0; i < n; i++)
        {
                sprintf(buffer, "%02X", (unsigned char)cp[i]);
                result += buffer;
        }
}
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Exit1Click(TObject *Sender)
{
        Close();
}
```

```
//---------------------------------------------------------------------------
```



```
void __fastcall TForm1::Button2Click(TObject *Sender)
{
        if (Memo2->Lines->Text == "")
        {
MessageBox(NULL, "Хеш-код должен быть сформированным", "Внимание",
MB_ICONWARNING);
        }
        else
        if (SaveDialog1->Execute())
        Memo2->Lines->SaveToFile(SaveDialog1->FileName);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        if (Memo1->Lines->Text == "")
        {
MessageBox(NULL, "Сообщение не может быть пустым", "Внимание", MB_ICONWARNING);
        }
        else
        {
        umac_ctx_t ctx;
 char *data_ptr;
 char data_ptr_tmp[32*1024];
 int data_len;
 char nonce[8];
       char key[128];
 char tag[21] = {0};
 int bytes_over_boundary;
 sprintf(data_ptr_tmp, "%s", Memo1->Text.c_str());
       sprintf(nonce, "%s", Edit1->Text.c_str());
       sprintf(key, "%s", Edit2->Text.c_str());
 data_len = strlen(data_ptr_tmp);
 data_len += 32 - (data_len % 32);
 data_ptr = (char *)malloc(data_len + 48);
 bytes_over_boundary = (ptrdiff_t)data_ptr & (16 - 1);
 if (bytes_over_boundary != 0)
  data_ptr += (16 - bytes_over_boundary);
 memset(data_ptr, '\0', data_len);
 memcpy(data_ptr, data_ptr_tmp, strlen(data_ptr_tmp));
 ctx = umac_new(key);
 umac(ctx, data_ptr, data_len, tag, nonce);
 umac_reset(ctx);
 pbuf(tag, UMAC_OUTPUT_LEN);
       Memo2->Lines->Add(result);
       }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button3Click(TObject *Sender)
{
```

```
        Memo1->Lines->Clear();
        Memo2->Lines->Clear();
}

//---------------------------------------------------------------------------
void __fastcall TForm1::N1Click(TObject *Sender)
{
        MessageBox(NULL, "Программу разработала студентка фак-та ЭИ 4-6 Ляпина
О.В.", "О программе ...", MB_ICONINFORMATION);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::N2Click(TObject *Sender)
{
 if (SaveDialog1->Execute())
        Memo1->Lines->LoadFromFile(SaveDialog1->FileName);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button4Click(TObject *Sender)
{
    umac_ctx_t ctx;
    char *data_ptr;
    int data_len = 4096;
    double cpb;
    int bytes_over_boundary, i;
    int length_pts[] = {32,64,128,256,512,512*2,512*4,512*8};
    char buffer[4096];
    /* Allocate memory and align to 16-byte multiple */
    data_ptr = (char *)malloc(data_len + 16);
    bytes_over_boundary = (ptrdiff_t)data_ptr & (16 - 1);
    if (bytes_over_boundary != 0)
        data_ptr += (16 - bytes_over_boundary);
    for (i = 0; i < data_len; i++)
        data_ptr[i] = (i*i) % 128;
    ctx = umac_new("abcdefghijklmnopqrstuvwxyz");
        Memo3->Lines->Add("Аутентификация начата.");
        for (i = 0; i < (int)(sizeof(length_pts)/sizeof(int)); i++) {
            cpb = run_cpb_test(ctx, length_pts[i], data_ptr, data_len);
                sprintf(buffer, "Аутентификация  сообщения длиной %d байт
выполнена за %5.2f clocks per byte", length_pts[i], cpb);
                Memo3->Lines->Add(buffer);
        }
        Memo3->Lines->Add("Аутентификация завершена.");
    umac_delete(ctx);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::N4Click(TObject *Sender)
{
        Memo1->Lines->Clear();
        Memo2->Lines->Clear();
}
```

Додаток Б

Лістинг статистичного портрету програмної реалізації геш-функції UMAC-32

```
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------
------------------------------------------------------------------------------
 C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
------------------------------------------------------------------------------
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.0000 *  frequency
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.0000 *  block-frequency
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.0000 *  cumulative-sums
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.0000 *  cumulative-sums
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.0000 *  runs
  2  86   2   0   0   0   2   3   5   0  0.000000 * 1.0000    longest-run
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.0000 *  rank
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.0000 *  fft
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.0000 *  nonperiodic-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.0000 *  nonperiodic-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.0000 *  nonperiodic-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.0000 *  nonperiodic-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.0000 *  nonperiodic-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.0000 *  nonperiodic-templates
 90   5   1   2   0   0   0   1   1   0  0.000000 * 0.2600 *  nonperiodic-templates
 99   1   0   0   0   0   0   0   0   0  0.000000 * 0.1300 *  nonperiodic-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.0000 *  nonperiodic-templates
 71  12   6   4   2   1   3   1   0   0  0.000000 * 0.4900 *  nonperiodic-templates
 91   6   2   0   0   1   0   0   0   0  0.000000 * 0.2400 *  nonperiodic-templates
 43   8   7   9   4   9   7   4   6   3  0.000000 * 0.8100 *  nonperiodic-templates
 19  18   9  11   7  11   4   8   9   4  0.005358   0.9100 *  nonperiodic-templates
 12   8   3   9   8   9  13  14  10  14  0.319084   0.9700    nonperiodic-templates
 30   6  15   7  11   4   6   8   7   6  0.000000 * 0.9000 *  nonperiodic-templates
  9   9   5   8  11  18   8  13   7  12  0.202268   0.9900    nonperiodic-templates
 95   2   3   0   0   0   0   0   0   0  0.000000 * 0.1400 *  nonperiodic-templates
 44  17   9   7   5   6   6   2   1   3  0.000000 * 0.7900 *  nonperiodic-templates
  4   8  12  15   9  13  15   6  10   8  0.191687   1.0000    nonperiodic-templates
  7  13   5   6  15  11  15   9   8  11  0.236810   0.9900    nonperiodic-templates
 19  15   6  12  11   6   9   5  10   7  0.037566   0.9600    nonperiodic-templates
 16   5   6  10   8  11  11  10   7  16  0.171867   0.9900    nonperiodic-templates
  1   7   5  17  11  16  17  15   7   4  0.000199   1.0000    nonperiodic-templates
  9  19  10  18  13  10  11   5   3   2  0.000555   1.0000    nonperiodic-templates
 11  11  14  18  14  10  13   8   1   0  0.000600   1.0000    nonperiodic-templates
  7  19  12  17  14  10  12   6   1   2  0.000170   1.0000    nonperiodic-templates
  7  10  10  13  13  17  14  11   1   4  0.012650   0.9900    nonperiodic-templates
  6  13  14  28   9  13   4   9   3   1  0.000000 * 1.0000    nonperiodic-templates
  7  16  19  15  11  12  10   6   2   2  0.000439   1.0000    nonperiodic-templates
 13  17  11  19  10  14   3  11   2   0  0.000026 * 1.0000    nonperiodic-templates
 92   4   2   0   1   0   0   0   1   0  0.000000 * 0.3100 *  nonperiodic-templates
 68  11   5   4   4   1   3   2   2   0  0.000000 * 0.6800 *  nonperiodic-templates
  4  12   7   9  11  13  10  12   9  13  0.595549   1.0000    nonperiodic-templates
  4  15  14  14  12  12  12  13   3   1  0.003712   1.0000    nonperiodic-templates
 52  24  17   2   1   2   1   0   1   0  0.000000 * 0.9900    nonperiodic-templates
 40  35  14   6   1   4   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
 11  13  19  19  12   4  10   8   2   2  0.000076 * 0.9900    nonperiodic-templates
 13  27  23  20   8   6   2   1   0   0  0.000000 * 1.0000    nonperiodic-templates
 19  26  18  13  11   2   3   5   3   0  0.000000 * 0.9900    nonperiodic-templates
```

```
  2   9  23  20  17  11   6   6   4   2  0.000000 * 1.0000    nonperiodic-templates
 44  25  17   9   1   1   2   1   0   0  0.000000 * 1.0000    nonperiodic-templates
 47  22  16   7   4   2   1   0   1   0  0.000000 * 1.0000    nonperiodic-templates
 81  13   5   1   0   0   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
 85   9   3   1   1   0   1   0   0   0  0.000000 * 0.9900    nonperiodic-templates
 83  16   1   0   0   0   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
 70  22   8   0   0   0   0   0   0   0  0.000000 * 0.9900    nonperiodic-templates
 83  10   6   1   0   0   0   0   0   0  0.000000 * 0.9900    nonperiodic-templates
 83  14   2   1   0   0   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
 79  19   2   0   0   0   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
 80  15   3   2   0   0   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
 84   7   6   1   2   0   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
 83  14   2   1   0   0   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
 11   6  11   9  14   8  17   8   9   7  0.334538   0.9900    nonperiodic-templates
  7   5   9   8   4  10  21  19  11   6  0.000555   0.9900    nonperiodic-templates
  6   8  13  14  11  12  14   9   4   9  0.319084   1.0000    nonperiodic-templates
 12  34  13  19   7   2   4   8   0   1  0.000000 * 1.0000    nonperiodic-templates
 92   8   0   0   0   0   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
 97   2   0   1   0   0   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.9900    nonperiodic-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.9800    nonperiodic-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
 84  15   1   0   0   0   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
 85   9   6   0   0   0   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.9900    nonperiodic-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
 70  18   8   4   0   0   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
 84   9   4   2   0   1   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
 99   1   0   0   0   0   0   0   0   0  0.000000 * 0.9800    nonperiodic-templates
 98   2   0   0   0   0   0   0   0   0  0.000000 * 0.9900    nonperiodic-templates
 99   1   0   0   0   0   0   0   0   0  0.000000 * 0.9900    nonperiodic-templates
 99   1   0   0   0   0   0   0   0   0  0.000000 * 0.9800    nonperiodic-templates
 99   1   0   0   0   0   0   0   0   0  0.000000 * 0.9900    nonperiodic-templates
 99   1   0   0   0   0   0   0   0   0  0.000000 * 0.9900    nonperiodic-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.0000 *  nonperiodic-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.0000 *  nonperiodic-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.0000 *  nonperiodic-templates
  3   3  14  16  15  11  16  13   5   4  0.000883   1.0000    nonperiodic-templates
 78   7   4   4   4   1   0   0   2   0  0.000000 * 0.5700 *  nonperiodic-templates
  3   5  11   7   8  16  21  16   9   4  0.000216   1.0000    nonperiodic-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.0000 *  nonperiodic-templates
  7  14  18  10  16  13  11   6   4   1  0.001509   1.0000    nonperiodic-templates
 15   5   6   5   9  16   9  10  13  12  0.115387   0.9700    nonperiodic-templates
 19  24  21  19   6   3   5   2   1   0  0.000000 * 1.0000    nonperiodic-templates
 74   4   7   4   4   2   0   4   0   1  0.000000 * 0.6600 *  nonperiodic-templates
 44  32  13   8   2   1   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
 14  10  10   8   7  16  12   8   5  10  0.366918   0.9600    nonperiodic-templates
 14  26  18  16   9   5   4   7   0   1  0.000000 * 1.0000    nonperiodic-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.0000 *  nonperiodic-templates
 50  20  17   9   3   1   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
  5   3   7  16  12   7  14  14  10  12  0.051942   1.0000    nonperiodic-templates
 15  25  20  24   4   5   4   2   1   0  0.000000 * 1.0000    nonperiodic-templates
 81   0   6   4   6   1   1   0   1   0  0.000000 * 0.4500 *  nonperiodic-templates
 77  18   3   2   0   0   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
  2  10  12  19  23   7  13   8   2   4  0.000001 * 1.0000    nonperiodic-templates
```

```
 45  26  12  10   4   1   1   1   0   0  0.000000 * 1.0000    nonperiodic-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.0000 *  nonperiodic-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.0000 *  nonperiodic-templates
 18  27  16  19   3   8   5   4   0   0  0.000000 * 1.0000    nonperiodic-templates
 17   9  12   8   5  13   8  11   4  13  0.115387   0.9900    nonperiodic-templates
 98   2   0   0   0   0   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
 56   9   5   4   6   6   1   4   4   5  0.000000 * 0.8000 *  nonperiodic-templates
 38  26  17  12   1   2   3   0   1   0  0.000000 * 1.0000    nonperiodic-templates
 81  11   7   1   0   0   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
 10  14  15  12  12  11  11  11   2   2  0.035174   1.0000    nonperiodic-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.9800    nonperiodic-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.0000 *  nonperiodic-templates
 40  35  13   6   5   1   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
 39  28  20   4   4   1   2   1   1   0  0.000000 * 1.0000    nonperiodic-templates
  3   8  17  12  15  12  12  11   5   5  0.025193   1.0000    nonperiodic-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.9800    nonperiodic-templates
 84  15   0   1   0   0   0   0   0   0  0.000000 * 0.9900    nonperiodic-templates
 57  17   9   4   4   1   4   2   0   2  0.000000 * 0.6900 *  nonperiodic-templates
 99   1   0   0   0   0   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
 69  26   4   1   0   0   0   0   0   0  0.000000 * 0.9800    nonperiodic-templates
  2   7  13  12  17  17   6  14   4   8  0.002374   1.0000    nonperiodic-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.9900    nonperiodic-templates
 81  12   4   3   0   0   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.0000 *  nonperiodic-templates
 79  15   5   1   0   0   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
 38  27  14  13   5   1   1   1   0   0  0.000000 * 1.0000    nonperiodic-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.9700    nonperiodic-templates
  4  14   9  22  17  11  13   6   4   0  0.000005 * 0.9900    nonperiodic-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.9700    nonperiodic-templates
 79  12   5   2   2   0   0   0   0   0  0.000000 * 0.9900    nonperiodic-templates
 24  16  11  12   4   7   7   6   7   6  0.000123   0.9200 *  nonperiodic-templates
 77  15   4   3   0   1   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
 84  12   4   0   0   0   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
 99   1   0   0   0   0   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
  4   7  17  18  15  12  10  13   1   3  0.000157   0.9900    nonperiodic-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.9700    nonperiodic-templates
 82  15   3   0   0   0   0   0   0   0  0.000000 * 0.9900    nonperiodic-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.0000 *  nonperiodic-templates
 78  13   8   0   0   1   0   0   0   0  0.000000 * 0.9900    nonperiodic-templates
 78  15   7   0   0   0   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
 99   1   0   0   0   0   0   0   0   0  0.000000 * 0.9800    nonperiodic-templates
  4  12  22  14  20   9   6   8   3   2  0.000002 * 1.0000    nonperiodic-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.9900    nonperiodic-templates
 77  19   3   1   0   0   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
 99   1   0   0   0   0   0   0   0   0  0.000000 * 0.9800    nonperiodic-templates
 36  19   5   8   7   3   5   9   4   4  0.000000 * 0.9100 *  nonperiodic-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.9600    nonperiodic-templates
 80  14   3   3   0   0   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
 99   1   0   0   0   0   0   0   0   0  0.000000 * 0.9600    nonperiodic-templates
 11  16  16  16   8   9  11  12   1   0  0.000439   1.0000    nonperiodic-templates
 99   1   0   0   0   0   0   0   0   0  0.000000 * 0.9800    nonperiodic-templates
 87   8   4   1   0   0   0   0   0   0  0.000000 * 1.0000    nonperiodic-templates
 99   1   0   0   0   0   0   0   0   0  0.000000 * 0.9900    nonperiodic-templates
 85   8   5   0   2   0   0   0   0   0  0.000000 * 0.3900 *  overlapping-templates
100   0   0   0   0   0   0   0   0   0  0.000000 * 1.0000    universal
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.0000 *  apen
```

```
  0   0   0   0   0   0   0   0   0   0  1.000000   -1.#IND *  random-excursions
  0   0   0   0   0   0   0   0   0   0  1.000000   -1.#IND *  random-excursions
  0   0   0   0   0   0   0   0   0   0  1.000000   -1.#IND *  random-excursions
  0   0   0   0   0   0   0   0   0   0  1.000000   -1.#IND *  random-excursions
  0   0   0   0   0   0   0   0   0   0  1.000000   -1.#IND *  random-excursions
  0   0   0   0   0   0   0   0   0   0  1.000000   -1.#IND *  random-excursions
  0   0   0   0   0   0   0   0   0   0  1.000000   -1.#IND *  random-excursions
  0   0   0   0   0   0   0   0   0   0  1.000000   -1.#IND *  random-excursions
  0   0   0   0   0   0   0   0   0   0  1.000000   -1.#IND *  random-excursions-variant
  0   0   0   0   0   0   0   0   0   0  1.000000   -1.#IND *  random-excursions-variant
  0   0   0   0   0   0   0   0   0   0  1.000000   -1.#IND *  random-excursions-variant
  0   0   0   0   0   0   0   0   0   0  1.000000   -1.#IND *  random-excursions-variant
  0   0   0   0   0   0   0   0   0   0  1.000000   -1.#IND *  random-excursions-variant
  0   0   0   0   0   0   0   0   0   0  1.000000   -1.#IND *  random-excursions-variant
  0   0   0   0   0   0   0   0   0   0  1.000000   -1.#IND *  random-excursions-variant
  0   0   0   0   0   0   0   0   0   0  1.000000   -1.#IND *  random-excursions-variant
  0   0   0   0   0   0   0   0   0   0  1.000000   -1.#IND *  random-excursions-variant
  0   0   0   0   0   0   0   0   0   0  1.000000   -1.#IND *  random-excursions-variant
  0   0   0   0   0   0   0   0   0   0  1.000000   -1.#IND *  random-excursions-variant
  0   0   0   0   0   0   0   0   0   0  1.000000   -1.#IND *  random-excursions-variant
  0   0   0   0   0   0   0   0   0   0  1.000000   -1.#IND *  random-excursions-variant
  0   0   0   0   0   0   0   0   0   0  1.000000   -1.#IND *  random-excursions-variant
  0   0   0   0   0   0   0   0   0   0  1.000000   -1.#IND *  random-excursions-variant
  0   0   0   0   0   0   0   0   0   0  1.000000   -1.#IND *  random-excursions-variant
  0   0   0   0   0   0   0   0   0   0  1.000000   -1.#IND *  random-excursions-variant
  0   0   0   0   0   0   0   0   0   0  1.000000   -1.#IND *  random-excursions-variant
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.0000 *  serial
100   0   0   0   0   0   0   0   0   0  0.000000 * 0.0000 *  serial
  0   0   0   0   0   0   0   0   0 100  0.000000 * 1.0000    lempel-ziv
 11  10  10   8   9  11   7   7  16  11  0.719747   0.9500 *  linear-complexity

The minimum pass rate for each statistical test with the exception of the random
excursion (variant) test is approximately = 0.960150 for a sample size = 100
binary sequences.

The minimum pass rate for the random excursion (variant) test is approximately
-1.#INF00 for a sample size = 0 binary sequences.
For further guidelines construct a probability table using the MAPLE program
provided in the addendum section of the documentation.
```