НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
Навчально-науковий фізико-технічний інститут
Кафедра математичних методів захисту інформації

«На правах рукопису»
УДК (впишіть правильний УДК!)

«До захисту допущено»
Завідувач кафедри
_____ Сергій ЯКОВЛЄВ
«__» _____ 2024 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою
«Математичні методи криптографічного захисту інформації»

зі спеціальності: 113 Прикладна математика
на тему: **«ДОСЛІДЖЕННЯ СТІЙКОСТІ СХЕМ ГЕШУВАННЯ ARGON2D ТА ARGON2I ДО АТАК ПОВНОГО ПЕРЕБОРУ З УРАХУВАННЯМ АРХІТЕКТУРИ ПРОЦЕСОРА»**

Виконав:
студент II курсу, групи ФІ-04
Бєш Радомир Андрійович _____

Керівник:
доцент кафедри ММЗІ, к.т.н., доцент
Кучинська Наталія Вікторівна _____

Рецензент:
посада, степінь, звання
Прізвище Ім'я По-батькові _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

Київ — 2024

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
Навчально-науковий фізико-технічний інститут
Кафедра математичних методів захисту інформації

Рівень вищої освіти — перший (бакалаврський)
Спеціальність — 113 Прикладна математика,
ОПП «Математичні методи криптографічного захисту інформації»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій ЯКОВЛЄВ
«__» _____ 2024 р.

ЗАВДАННЯ
на дипломну роботу

Студент: Бєш Радомир Андрійович

1. Тема роботи: «*ДОСЛІДЖЕННЯ СТІЙКОСТІ СХЕМ ГЕШУВАННЯ ARGON2D ТА ARGON2I ДО АТАК ПОВНОГО ПЕРЕБОРУ З УРАХУВАННЯМ АРХІТЕКТУРИ ПРОЦЕСОРА*», науковий керівник дисертації: доцент кафедри ММЗІ, к.т.н., доцент Кучинська Наталія Вікторівна,

затверджені наказом по університету №__ від «__» _____ 2024 р.

2. Термін подання студентом роботи: «__» _____ 2024 р.

3. Об'єкт дослідження: процес безпечного зберігання паролів з використанням функцій гешування

4. Предмет дослідження: схема гешування «Argon2» та її версії

5. Перелік завдань:

1) огляд опублікованих джерел за тематикою роботи (сучасні функції гешування паролів, їх структурні особливості, функції формування ключів, функції з використанням великої кількості пам'яті, структура схеми гешування «Argon2» та її версій);

2) аналіз необхідних теоретичних/практичних відомостей щодо структури та вразливостей функції гешування argon2 до атака повного

перебору;

3) проведені експерименті до атак повного перебору на схему гешування «Argon2» із використанням спеціального ПЗ, доведена стійкість схеми до атак повного перебору з використанням процесорів різних потужностей;

4) робота над розробкою програмної реалізації скриптів та програмного коду для візуалізації отриманих результатів;

5) робота з часовими оцінками.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: *Презентація доповіді.)*

7. Орієнтовний перелік публікацій: *планується доповідь на всеукраїнській конференції.)*

8. Дата видачі завдання: 10 вересня 2023 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання	Примітка
1	Узгодження теми роботи із науковим керівником	01-15 вересня 2023 р.	Виконано
2	Огляд опублікованих джерел за тематикою дослідження	Вересень-жовтень 2023 р.	Виконано
3	...	...	Виконано

Студент _____ Радомир Бєш

Керівник _____ Наталія КУЧИНСЬКА

РЕФЕРАТ

Кваліфікаційна робота містить: 70 стор., 10 рисунки, 7 таблиць, 36 джерел.

Аналіз стійкості сучасної схеми гешування «Argon2» та її версій «Argon2d» та «Argon2i» до атак повного перебору з урахуванням потужностей процесорів, використовуючи спеціальне програмне забеспечення. Схема гешування «Argon2» та її версії. Процес безпечного зберігання паролів з використанням функцій гешування.

КРИПТОГРАФІЧНА ГЕШ-ФУНКЦІЯ, СХЕМА ГЕШУВАННЯ «ARGON2D» ТА «ARGON2I», «PBKDF», MHF, АТАКА ПОВНОГО ПЕРЕБОРУ

ABSTRACT

The qualification work contains: 70 pages, 10 figures, 7 tables, 36 sources.

The analysis of the resistance of the modern hashing scheme "Argon2"and its versions "Argon2d"and "Argon2i"to brute-force attacks, taking into account the computational power of processors using special software. The hashing scheme "Argon2"and its versions. The process of secure password storage using hashing functions.

CRYPTOGRAPHIC HASH FUNCTION, HASHING SCHEME "ARGON2D"AND "ARGON2I <PBKDF», MHF, BRUTE-FORCE ATTACK

ЗМІСТ

ВСТУП

Актуальність дослідження. У сфері криптографічної безпеки важливість надійних та ефективних геш-функцій неможливо переоцінити. Геш-функції відіграють вирішальну роль у різних програмах, таких як гешування пароля та перевірка цілісності даних. В епоху, відзначену зростанням кіберзагроз і швидким розвитком технологій, потреба в надійних криптографічних заходах має першорядне значення. Паролі часто використовуються для захисту цифрових активів, ці слова завжди стають мішенню злочинців у спробі отримати несанкціонований доступ. Традиційні геш-функції, які колись вважалися безпечними, тепер піддаються все більш складним атакам, використовуючи як обчислювальну потужність, так і інноваційні методи. Оскільки кіберзагрози розвиваються, так само повинен розвиватися наш криптографічний захист.

Вивчення стійкості сучасних геш-функцій в умовах різних варіантів атаки стає важливим для оцінки їх постійної ефективності. Порушення пароля, витік даних та випадки несанкціонованого доступу стали звичайним явищем, підкреслюючи критичну потребу в геш-функціях, які можуть протистояти невпинному прагненню супротивників.

У 2014 році був оголошен конкурс Password Hashing Competition про створення нової функції гешування паролів. Новий алгоритм мав вимоги до обсягу використовуваної пам'яті, кількості проходів через блоки пам'яті та стійкості до криптоаналізу. Конкурс завершився 20 листопада 2015 року, а переможцем став «Argon2». Крім цього, особливе визнання отримали чотири алгоритми: «Catena», «Lyra2», «Makwa» і «yescrypt».

Метою дослідження є аналіз стійкості сучасної схеми гешування «Argon2» та її версій «Argon2d» та «Argon2i» до атак повного перебору з урахуванням потужностей процесорів, використовуючи спеціальне програмне забеспечення.

1) Проаналізувати теоритичні відомості за тематикою дослідження

2) Сформувати теоретичне підґрунтя, яке необхідне для розуміння основних принципів до структури сучасних геш-функцій

3) Описати схему гешуваняня «Argon2» та провести аналіз стійкості схеми до атак повного перебору та до інших атак.

4) Виконати експерименти для доведення практичної значущості стійкості до атак повного перебору «Argon2» в сучасному світі.

Об'єктом дослідження є процес безпечного зберігання паролів з використанням функцій гешування.

Предметом дослідження є схема гешування «Argon2» та її версії.

Методи дослідження: методи комп'ютерного та статистичного моделювання, комбінаторний аналіз, теорія складності алгоритмів.

Наукова новизна дослідження полягає в аналізі стійкості геш-функцій «Argon2», з точку зору швидкості підбору пароля. Даний аналіз спрямований на виявлення унікальних особливостей та застосування цих алгоритмів у високоризикових областях, таких як зберігання та перевірка паролів.

Практичне значення цього дослідження має на меті принести нові знання та висвітлити важливі аспекти використання сучасної схеми гешування «Argon2» у сферах, де забезпечення безпеки та ефективності є ключовими факторами.

1 ОСНОВНІ ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО СУЧАСНІ ГЕШ-ФУНКЦІЇ

В данному розділі буде надано теоретичні знання про криптографічні геш-функції. Буде розглянуто сучасні геш-функції та схеми гешування та їх застосування. Також згадаємо конкурс гешування паролів (The Password Hashing Competition, PHC). Дамо визначення функції форумвання ключа (Key Derivation Function, KDF) і функції з використанням великої кількості пам'яті (Memory-hard function, MHF).

1.1 Криптографічна геш-функція

Геш-функція [1] — це функція, яка приймає вхідні дані довільної довжини і видає вихідні дані фіксованої коротшої довжини, ефективно стискаючи вхідні дані. Ці функції використовуються в багатьох сферах, таких як ефективне отримання даних. *Криптографічні геш-функції [1]* є підмножиною функцій гешування і повинні відповідати певним властивостям, а саме стійкості до прообразу, стійкості до другого прообразу та стійкості до колізій. У цій дипломній роботі буде розглядаються лише криптографічні функції гешування.

Означення 1.1. *Криптографічна геш-функція* H приймає на вхід ключ K і рядок $x \in \{0,1\}^*$ і видає рядок $H_k(x) \times K \in \{0,1\}^{\ell(n)}$ (де K — множина параметрів безпеки (ключів)).
Якщо H_k визначено лише для вхідних даних $x \in \{0,1\}^{\ell'(n)}$ і $\ell'(n) > \ell(n)$, тоді ми кажемо, що H є функцією гешування з фіксованою довжиною для вхідних даних довжини ℓ'. У цьому випадку ми також називаємо H функцією стиснення.

Гешування паролів [1] — це процес, у якому пароль передається у функцію гешування. Це фактично стандартний метод збереження паролів

в операційних системах і додатках. У разі, якщо зловмисник отримує доступ до таких гешів паролів, практично неможливо відновити оригінальний пароль. Тому функції гешування також використовуються в процесі перевірки пароля, під час якої введений пароль гешується і порівнюється зі збереженим гешем.

Функції формування ключів (key derivation functions) [2] базуються на функціях гешування. Їх основна мета - взяти вхідні дані і видати вихідні дані, які можна використовувати як криптографічний ключ. Вхідні дані зазвичай є паролем або іншим матеріалом, таким як біометричний зразок, перетворений у бінарну форму. Ці матеріали самі по собі можуть використовуватися як криптографічні ключі, але часто не мають властивостей хорошого криптографічного ключа, таких як достатня ентропія або довжина.

1.1.1 Властивості криптографічних геш-функцій

Криптографічна геш-функція називається *важкооборотною (one-way hash function) [1]* , якщо вона відповідає таким вимогам:

– **Складність пошуку прообраза (preimage resistance) [1]** Пошук прообразу полягає в знаходженні такого вхідного значення x, для якого геш від x буде дорівнювати заданому значенню y:

$$H(x_1) = y. \tag{1.1}$$

Імовірність знаходження прообразу для геш-функції розміром n бітів:

$$P_{\text{preimage}} \approx \frac{1}{2^n}. \tag{1.2}$$

– **Складність пошуку другого прообраза (second preimage resistance) [1]** Пошук другого прообразу полягає в знаходженні другого вхідного значення x_2, щоб значення гешу співпадало з гешем вхідного

значення x_1:

$$H(x_1) = H(x_2), \tag{1.3}$$

де $x_1 \neq x_2$. Імовірність знаходження другого прообразу для геш-функції довжиною n бітів становить:

$$P_{\text{second\_preimage}} \approx \frac{1}{2^n}. \tag{1.4}$$

– **Складність пошуку колізій (collision resistance) [1]** Пошук колізії полягає в тому, щоб знайти значення геш-функції для двох різних входів x_1 та x_2, які б дорівнювали одне одному:

$$H(x_1) = H(x_2), \tag{1.5}$$

де $x_1 \neq x_2$. Імовірність знаходження колізії для геш-функції розміром n бітів становить:

$$P_{\text{collision}} \approx \frac{1}{2^{n/2}}. \tag{1.6}$$

1.2 Галузі використання сучасних геш-функцій

1) **Збереження паролів:**

– Геш-функції використовуються для зберігання паролів у безпечних формах. Замість збереження самого паролю, системи зберігають його хеш-значення, що робить важким відновлення оригінального паролю навіть у випадку витоку даних.

2) **Цифровий підпис:**

– Геш-функції використовуються для створення цифрових підписів. Вони генерують хеш-значення для повідомлення, а приватний ключ використовується для створення цифрового підпису, який можна перевірити за допомогою відкритого ключа.

3) **Перевірка цілісності даних:**

– Геш-функції використовуються для створення геш-сум файлів або

повідомлень. Це дозволяє перевірити цілісність даних: якщо геш-значення співпадає, то дані не були змінені.

4) **Блокчейн і криптовалюти:**

– Багато криптовалют та технологій блокчейн використовують геш-функції для створення геш-значень блоків та транзакцій, забезпечуючи надійність інформації та непереборне гешування.

1.3 Функції формування ключів (Key Derivation Function, KDF)

Функція формування ключів (KDF) [2] — це функція, яка утворює один або більше секретних ключів на основі секретного значення (майстер-ключ, пароль) за допомогою псевдовипадкової функції. Функція генерації ключів може використовуватися для генерації ключа необхідної довжини або заданого формату. В якості псевдовипадкової функції для формування ключа використовуються криптографічні геш-функції. Широко використовувані алгоритми формування ключа: «bcrypt», «PBKDF2», «scrypt».

1.3.1 Алгоритм «PBKDF2»

«PBKDF2» [3] є функцією виведення ключів на основі пароля, визначеною в RFC 8018 [5]. Цей RFC детально описує два випадки використання «PBKDF2»: схему шифрування на основі пароля та схему аутентифікації повідомлень на основі пароля. Функція вимагає чотири вхідних параметри: парольна фраза (password),це будь-які дані, які є джерелом для подальшого процесу виведення ключа, криптографічна сіль (salt), кількість ітерацій (iteration count) та довжина ключа, який потрібно вивести. Крім того, функція вимагає псевдовипадкову функцію, яка використовується в процесі виведення ключа. Метою параметра

(iteration count) є захист від атак повного перебору та словникових атак, які виконуються на «PBKDF2». Кількість ітерацій подовжує час, необхідний для виведення одного ключа. Технічно, кількість ітерацій означає кількість послідовних запусків вибраної псевдовипадкової функції для кожного блоку виведеного ключа. Взагалі, кількість ітерацій слід вибирати якомога більшою, враховуючи той факт, що час обробки повинен бути прийнятним для кінцевого користувача. [4] Згідно рекомендацій, мінімальна кількість ітерацій повинна становити 1,000 ітерацій, а для критичних систем безпеки рекомендується кількість 10,000,000 ітерацій. Функція описується за допомогою наступного алгоритму:

Загальний вид визову функції «PBKDF2»:

$$DK = PBKDF2(PRF, P, S, c, dkLen), \tag{1.7}$$

де

– PRF- псевдовипадкова функція,

– P- пароль,

– S- сіль (salt),

– c- кількість ітерацій виражена додатнім цілим числом,

– $dkLen$- довжина виведеного ключа,

– DK- сгенерований ключ , довжиною $dkLen$ вираженою додатнім цілим числом.

Коротко опишемо як працює цей алгоритм:

на початку функція «PBKDF2» приймає вхідні параметри , а саме пароль P, сіль S, кількість ітерацій C, бажану довжину ключа $dkLen$ та псевдовипадкову функцію PRF:

input: P, S, C, PRF

Далі перевіряється, чи бажана довжина ключа не перевищує допустиму межу.

if $dkLen > (2^{32} - 1) \times hLen$, де $hLen$- допустима межа. Якщо

перевищує, алгоритм завершується.

Далі обчислюється кількість блоків L, необхідних для досягнення бажаної довжини ключа, та кількість байтів у останньому блоці r. Кожен блок має довжину $hLen$ байтів (довжина виходу PRF), за винятком останнього блоку, який може бути коротшим:

$$L = \left\lceil \frac{dkLen}{hLen} \right\rceil$$
$$r = dkLen - (L - 1) \times hLen$$

Для кожного блоку i (від 1 до L) використовується функція F для обчислення блоку T_i:

$$T_1 = F(P, S, c, 1)$$
$$T_2 = F(P, S, c, 2)$$
$$\vdots$$
$$T_i = F(P, S, c, i)$$
$$\vdots$$
$$T_L = F(P, S, c, L)$$

Функція F виконує C ітерацій функції PRF, починаючи з комбінації паролю, солі та індексу блоку. Результати кожної ітерації об'єднуються за допомогою операції $\oplus$.

$$F(P, S, c, i) = U_1 \oplus U_2 \oplus \cdots \oplus U_c$$
$$U_1 = PRF(P, S \parallel \mathrm{INT}(i))$$
$$U_2 = PRF(P, U_1)$$
$$\vdots$$
$$U_c = PRF(P, U_{c-1})$$

Далі всі блоки $T_1, T_2, ..., T_L$ об'єднуються для отримання кінцевого

виведеного ключа DK. Від останнього блоку береться r байтів.

$$DK = T_1 \parallel T_2 \parallel \cdots \parallel T_l\langle 0 \dots r-1 \rangle$$

У Виводі отримуємо згенерований ключ:

output: DK

«Argon2» використовується як стандартна функція «PBKDF» в LUKS(Linux Unified Key Setup) версії 2.

1.4 Геш-функція «BLAKE2b»

«BLAKE2b» — це криптографічна геш-функція, створена Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, та Christian Winnerlein, яка є наступною версією «BLAKE», одного з фіналістів конкурсу SHA-3 [6]. Вона розроблена для високої швидкості та безпеки і оптимізована для 64-бітових платформ.

Короткий опис, як працює алгоритм «BLAKE2b»:

На початку ініціалізації, визначаються константи (ініціалізаційні вектори, константи перестановок та інші параметри). Створюється внутрішній стан V, який складається з 8 слів (по 64 біти кожне).

$$V = (V_0, V_1, V_2, V_3, V_4, V_5, V_6, V_7)$$

Внутрішній стан ініціалізується шляхом копіювання ініціалізаційних векторів (IV) у перші 8 слів внутрішнього стану. Параметри гешування (довжина вихідного геша, ключ, довжина ключа, персоналізація та інші параметри) додаються до внутрішнього стану. Повідомлення розбивається на блоки по 128 байтів, і кожен блок обробляється окремо. Нехай M - це повідомлення, тоді воно розбивається на m блоків:

$$M = (M_1, M_2, M_3..., M_m)$$

Кожен блок обробляється 12 раундами перестановок, використовуючи допоміжні змінні (векторні регістри), які взаємодіють за допомогою функцій стискнення G. Функція G складається з серії арифметичних операцій ('+', '$\oplus$' , '$\ggg$' (циклічний зсув вліво)), що виконується на парах слів.

$$G(a,b,c,d) \rightarrow \begin{cases} a = (a+b+x) \mod 2^{64} \\ d = (d \oplus a) \ggg 32 \\ c = (c+d) \mod 2^{64} \\ b = (b \oplus c) \ggg 24 \\ a = (a+b+y) \mod 2^{64} \\ d = (d \oplus a) \ggg 16 \\ c = (c+d) \mod 2^{64} \\ b = (b \oplus c) \ggg 63 \end{cases}$$

Після обробки всіх блоків повідомлення виконується фіналізація внутрішнього стану. Останні зміни робляться для забезпечення того, що всі частини повідомлення були належним чином враховані. Внутрішній стан перетворюється у вихідний геш H, який відрізається до потрібної довжини (наприклад, 256 біт або 512 біт).

$$H = (V_0, V_1, V_2, V_3, V_4, V_5, V_6, V_7)$$

«BLAKE2b» оптимізований для 64-бітових платформ і забезпечує високу швидкість обчислення та високий рівень криптографічної безпеки, включаючи стійкість до колізій та інших атак. Він підтримує параметри персоналізації, секретні ключі, сіль та інші налаштування, і є відкритим стандартом, вільним для використання у будь-яких додатках, детальніше можна ознайомитись тут...

На рисунку 1.1 зображено на скільки «BLAKE2b» працює швидше

ніж інші функції гешування.

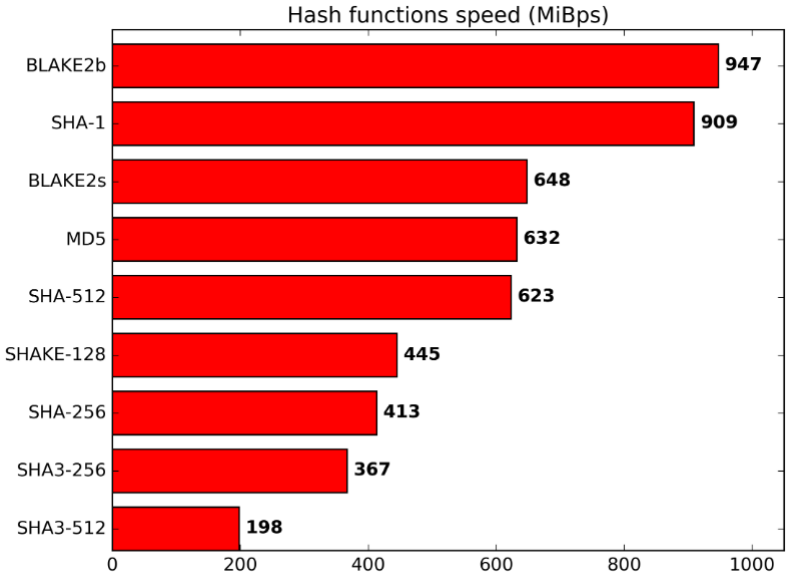


Рисунок 1.1 – Швидкість гешування MB в секунду

Розуміння цього алгоритму дуже важлива частина тому, що саме цей алгоритм використовується у схемі гешування «Argon2», де потрібна висока швидкість і безпека гешування.

1.5 Функції з використанням великої кількості пам'яті (Memory-hard function, MHF)

У криптографії *функція, що вимагає значної кількості пам'яті MHF [7]* — це функція, яка використовує значний обсяг пам'яті для ефективної оцінки. Вона відрізняється від функції, пов'язаної з пам'яттю *memory-bound function [8]*, що тягне за собою витрати шляхом уповільнення обчислень через затримку пам'яті. MHF мають підвищені вимоги до пам'яті, що значно зменшує перевагу обчислювальної ефективності користувацького обладнання над апаратними засобами загального призначення порівняно з не MHF.

MHF призначений для використання великих обсягів пам'яті

комп'ютера, що робить паралельні обчислення менш ефективними. Щоб використовувати менше пам'яті для оцінки функції, існує значний час. Оскільки кожне обчислення MHF потребує великого обсягу пам'яті, кількість обчислень функцій, які можна виконати одночасно, обмежена обсягом доступної пам'яті. Це знижує ефективність спеціалізованого апаратного забезпечення наприклад, інтегральних схем (ASIC) для конкретних програм і графічних процесорів (GPU), які використовують розпаралелювання під час обчислення MHF для великих вхідних даних (таких як геші паролів або криптовалюти).

Дамо формальні визначення з історника [9]:

Означення 1.2. *Алгоритм, що потребує значних ресурсів пам'яті, є таким, що використовує* $S(n)$ *обсяг пам'яті та* $T(n)$ *операцій, де* $S(n) = \Omega(T(n)^{1-\epsilon})$.

Таким чином, алгоритм, що потребує значних ресурсів пам'яті, — це алгоритм, який асимптотично використовує майже стільки ж місць у пам'яті, скільки й операцій; його також можна розглядати як алгоритм, який використовує найбільшу кількість пам'яті для заданої кількості операцій.

Означення 1.3. *Функція, що потребує значних ресурсів пам'яті(memory-hard function)* — це відображення $f : X \to Y$ таке, що для будь-якого $x \in X$, $f(x)$ може бути обчислена за $O(\text{poly}(N))$ кроків, використовуючи N обсяг пам'яті, але супротивник, який має тільки $N - 1$ обсяг пам'яті, тоді імовірність вгадати $f(x)$ не більше ніж $\frac{1}{|Y|}$.

MHFs можна розділити на дві різні групи:

1) залежні від даних функції пам'яті «data-dependent memory-hard functions»(dMHFs),

2) та незалежні від даних функції пам'яті «data-independent memory-hard functions»(iMHFs).

Схема гешування «Argon2» використовує як (dMHFs) так і (iMHFs). Таким чином «Argon2» має дві версії «Argon2в» та «Argon2i». Докладніше ці версії

будуть описані у другому розділі.

1.6 Історія схем гешування паролів

Паролі залишаються основною формою автентифікації на різних веб-сервісах. Паролі зазвичай зберігаються у гешованій формі в базі даних сервера. Ці бази даних часто захоплюються зловмисниками, які потім застосовують атаки за словником, оскільки паролі зазвичай мають низьку ентропію. Починаючи з кінця 70-х років, паролі гешуються разом з випадковим значенням солі, щоб запобігти виявленню однакових паролів у різних користувачів і сервісів. Обчислення геш-функції, яке ставало все швидшим завдяки закону Мура, було викликано багаторазово, щоб збільшити вартість підбору пароля для зловмисника.

Тим часом, зломщики паролів перейшли на нові архітектури, такі як FPGA, багатоядерні графічні процесори (GPU) і спеціалізовані модулі ASIC, де амортизаційна вартість багаторазово повторюваної геш-функції набагато нижча. Швидко стало зрозуміло, що ці нові середовища чудові, коли обчислення майже не потребують пам'яті, але вони зазнають труднощів при роботі з великим обсягом пам'яті. Захисники відповіли, розробивши функції, що потребують значних ресурсів пам'яті (MHFs), які вимагають великої кількості пам'яті для обчислення. Прикладом є схема гешування «Argon2», яку детальніше буде розглянуто у другому розділі.

Проблеми існуючих схем полягають в тому, що проектування функції, що потребує значних ресурсів пам'яті, виявилося складною проблемою. З початку 80-х років відомо, що багато криптографічних проблем, які, здавалося б, вимагають великої кількості пам'яті, насправді дозволяють здійснити компроміс між часом і пам'яттю (time-memory trade-off), де зловмисник може здійснити компроміс пам'яті на час і виконати свою роботу на швидкому обладнанні з малою пам'яттю. У застосуванні до схем гешування паролів це означає, що зломщики паролів все ще можуть бути реалізовані на спеціалізованому обладнанні, навіть

якщо це вимагає деяких додаткових витрат. Наприклад, функція scrypt дозволяє здійснити простий компроміс, де добуток часу на площу залишається майже постійним.

Іншою проблемою існуючих схем є їх складність. Та ж функція scrypt викликає стек підпроцедур, раціональність дизайну яких не була повністю обґрунтована (наприклад, scrypt викликає SMix, який викликає ROMix, який викликає BlockMix, який викликає Salsa20/8 тощо). Це важко аналізувати, і, більше того, важко досягти впевненості. Нарешті, схема не є гнучкою у розділенні витрат на час і пам'ять. У той же час історія криптографічних конкурсів продемонструвала, що найбезпечніші проекти поєднуються з простотою, де кожен елемент добре обґрунтований, а криптоаналітик має якомога менше точок входу.

1.6.1 Конкурс гешування паролів (The Password Hashing Competition, PHC)

У 2014 році було оголошено конкурс The Password Hashing Competition (PHC) [10] про створення нової схеми гешування паролів. Конкурс Password Hashing Competition також підкреслив наступні проблеми:

– Чи повинно адресування пам'яті (функції індексування) бути незалежним від введення або залежним від введення, або гібридним? Перший тип схем, де місце зчитування пам'яті відоме заздалегідь, одразу стає вразливим до атак компромісу між часом і пам'яттю, оскільки зловмисник може попередньо обчислити відсутній блок до моменту, коли він буде потрібен.Зі свого боку, схеми, залежні від введення, вразливі до атак побічних каналів, оскільки інформація про час дозволяє набагато швидше шукати паролі.

– Чи краще заповнити більше пам'яті, але постраждати від компромісів між часом і пам'яттю, чи зробити більше проходів через

пам'ять для підвищення надійності? Це питання було досить важко вирішити через відсутність універсальних інструментів для аналізу компромісів та відсутність єдиної метрики для вимірювання витрат зловмисника.

– Як слід обчислювати адреси, незалежні від введення? Кілька варіантів, що здавалися безпечними, були атаковані.

– Яким має бути розмір одного блоку пам'яті? Зчитування менших випадкових блоків відбувається повільніше (у циклах на байт) через принцип просторової локалізації кешу ЦП. У свою чергу, більші блоки важко обробляти через обмежену кількість довгих регістрів.

– Якщо блок великий, як вибрати внутрішню функцію стиснення? Чи повинна вона бути криптографічно безпечною чи більш легкою, забезпечуючи лише базове змішування вхідних даних? Багато кандидатів просто запропонували ітеративну конструкцію та виступили проти криптографічно сильних трансформацій.

– Які операції слід використовувати функції стиснення для досягнення максимальної вигоди зловмисника за фіксованих обчислювальних потужностей на настільному комп'ютері?

– Як використовувати кілька ядер сучасних ЦП, коли вони доступні? Паралелізація викликів функції гешування без будь-яких вказівок на ізоляцію може призвести до простих атак із компромісом між часом і пам'яттю.

Так у 2015 році «Argon2» [11] оголосили переможцем конкурсу (PHC), схема гешування була розроблена Алексом Бірюковим (англ. Alex Biryukov), Даніелем Дину (англ. Daniel Dinu) и Дмитрієм Ховратовичем (англ. Dmitry Khovratovich) з университету Люксембурга в 2015 році. Також судді звернути увагу на «yescrypt» [12] або його багатий набір функцій та простий спосіб оновлення «scrypt».

Розробники запропонували нову схему гешування під назвою «Argon2». «Argon2» представляє найсучасніші досягнення в розробці функцій, що потребують значних ресурсів пам'яті. Це оптимізована та

проста в застосуванні схема. Вона спрямована на максимальне заповнення пам'яті та ефективне використання багатьох обчислювальних одиниць, при цьому забезпечуючи захист від атак із компромісом між часом і пам'яттю. «Argon2» оптимізована для архітектури x86 та використовує кеш і організацію пам'яті сучасних процесорів Intel та AMD. «Argon2» має два варіанти: «Argon2d» та «Argon2i», які далі будуть детальніше розгялнуті у другому розділі. «Argon2d» є швидшим і використовує доступ до пам'яті, залежний від даних, що робить його придатним для криптовалют та застосувань без загроз з боку атак побічних каналів. «Argon2i» використовує доступ до пам'яті, незалежний від даних, що є кращим для гешування паролів і витягу ключів на основі паролів. «Argon2i» повільніший, оскільки робить більше проходів через пам'ять для захисту від атак із компромісом між часом і пам'яттю.

Висновки до розділу 1

В цьому розділі було розгялнуто такі важливі теми як криптографічна геш-функція, функція формування ключів, ознайомились з алгоритмами «PBKDF2» та геш-функцією «BLAKE2b». Ця інформація знадобиться для ліпшого розуміння, як працює схема гешування «Argon2», яка детально буде описана у другому розділі.

2 ДЕТАЛЬНИЙ ОПИС СХЕМИ ГЕШУВАННЯ «ARGON2»

У другому розділі детально опишемо схему гешування «Argon2». Розгялнемо дві версії «Argon2d» та «Argon2i». Покажемо механізми захисту від атак повного перебору та атак коспромісу між часом і пам'ятю.

2.1 Схема гешування «Argon2»

«Argon2» є сучасною криптографічною схемою гешування, розробленою для забезпечення високої безпеки та ефективності. Вона була представлена у 2015 році і здобула перемогу в конкурсі Password Hashing Competition (PHC). «Argon2» має два основні варіанти: «Argon2d» та «Argon2i» [12] та об'єднаний варіант «Argon2id». «Argon2d» використовує доступ до пам'яті, залежний від даних, що робить його стійким до атак із компромісом між часом і пам'яттю, але вразливим до атак по побічних каналів. Він підходить для криптовалют та серверних застосувань. «Argon2i», навпаки, використовує доступ до пам'яті, незалежний від даних, що робить його кращим для гешування паролів та витягу ключів на основі паролів, оскільки він забезпечує більшу стійкість до атак по побічних каналів. «Argon2» вирізняється простим дизайном, високою швидкістю заповнення пам'яті та ефективним використанням багатьох обчислювальних одиниць, що робить його оптимальним вибором для сучасних систем безпеки.

2.1.1 Вхідні параметри «Argon2»

«Argon2» має два типи вхідних даних: основні вхідні дані та додаткові вхідні дані, або параметри. Основні вхідні дані — це повідомлення P і

випадкове значення S , які є паролем і сіллю відповідно для гешування паролів.

Первинні вхідні дані:

– повідомлення P може мати будь-яку довжину від 0 до $2^{32}-1$ байтів;

– сіль S може мати будь-яку довжину від 8 до $2^{32}-1$ байтів (рекоменедується довжина солі 16 байтів для гешування пароля).

Вторинні вхідні дані:

– ступінь паралелізму p визначає, скільки незалежних (але синхронізованих) обчислювань може виконуватися (цей параметр може приймати будь-яке ціле значення від 1 до $2^{24}-1$);

– довжина тега τ може бути будь-яким цілим числом байтів від 4 до $2^{32}-1$;

– розмір пам'яті m може бути будь-яким цілим числом у кілобайтах від $8p$ до $2^{32}-1$ (фактична кількість блоків — це m', що дорівнює m, округленому вниз до найближчого кратного $4p$);

– кількість ітерацій t (використовується для налаштування часу виконання незалежно від розміру пам'яті) може бути будь-яким цілим числом від 1 до $2^{32}-1$;

– номер версії v є одним байтом $0x19$ (в даній роботі розглянуто 19 версію);

– секретне значення K (виступає ключем, якщо це необхідно, але за замовчуванням не припускається використання будь-якого ключа, використання ключа потрібно для для функції формування ключів «PBKDF») може мати будь-яку довжину від 0 до $2^{32}-1$ байтів;

– асоційовані дані X можуть мати будь-яку довжину від 0 до $2^{32}-1$ байтів;

– тип y «Argon2»: 0 для «Argon2d», 1 для «Argon2i», 2 для «Argon2id».

«Argon2» використовує внутрішню функцію стиснення G з двома вхідними даними по 1024 байтів та вихідними даними на 1024 байтів, а також внутрішню геш-функцію H. Тут H є геш-функцією «BLAKE2b» [6], а G базується на її внутрішній перестановці. Режим роботи «Argon2» є досить простим, коли не використовується паралелізм, функція G ітерується m разів. На кроці i з пам'яті береться блок з індексом $\phi(i) < i$, як зображено на рисунку 2.1, де $\phi(i)$ визначається або попереднім блоком у «Argon2d», або є фіксованим значенням у «Argon2i».

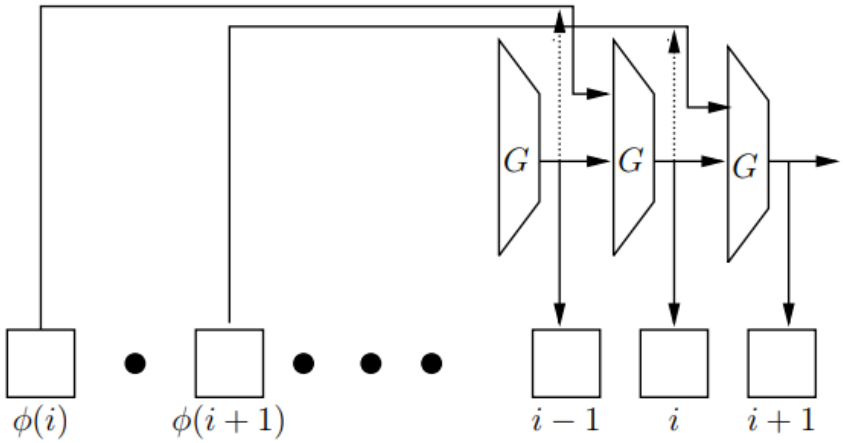


Рисунок 2.1 – Режим роботи «Argon2» без паралелізму

2.1.2 Заповнення блоків пам'яті

«Argon2» слідує концепції *"extract-then-expand"* тобто *витягнути-розширити*. Концепція *"extract-then-expand"* використовується в криптографії, особливо в контексті функцій формування ключів (key derivation functions, KDF). Вона складається з двох основних етапів, розглянутих нижче.

1) **Extract (Витягування).** На цьому етапі початковий матеріал (зазвичай це пароль або інший секретний ключ) обробляється для

витягнення криптографічно сильного ключа з невідомого або низькоентропійного вхідного матеріалу. Мета полягає в тому, щоб зменшити вплив недосконалостей або слабкостей в початковому матеріалі, створивши єдиний ключ, який має високу ентропію та стійкість до атак.

2) **Expand (Розширення).** На цьому етапі витягнутий ключ використовується для генерації одного або більше криптографічних ключів необхідної довжини. Мета полягає в тому, щоб створити достатню кількість ключового матеріалу для використання в різних криптографічних протоколах, зберігаючи при цьому високу безпеку і стійкість.

Спочатку «Argon2» витягує ентропію з повідомлення і випадкового значення, гешуючи їх. Всі інші параметри також додаються до вхідних даних. Вхідні дані змінної довжини P, S, K, X доповнюються їх довжинами:

$$H_0 = H(p,\tau,m,t,v,y,\langle P\rangle,P,\langle S\rangle,S,\langle K\rangle,K,\langle X\rangle,X).$$

Тут H_0 є початковим значенням геш-функції довжиною 64 байт, а параметри $p, \tau, m, t, v, y, \langle P\rangle, \langle S\rangle, \langle K\rangle, \langle X\rangle$ розглядаються як 32-бітні цілі числа у форматі little-endian, тобто у порядку від меншого до більшого.

Опишемо вхідні дані:

– p: степінь паралелізма (degree of parallelism) — це кількість незалежних потоків (lanes), які можуть бути виконані одночасно. Значення p визначає, як пам'ять буде розподілена для паралельного оброблювання.

– τ: довжина тега (tag length) — це довжина вихідного геш-значення, яке потрібно отримати. Наприклад, це може бути 32 байта або 64 байта і т.д.

– m: розмір пам'яті (memory size) — це кількість пам'яті, яке буде використовуватись в процесі гешування. Зазвичай вимірюється у

кілобайтах (kB) або мегабайтах (MB).

– t: кількість ітерацій (number of iterations) — це кількість разів, які алгоритм гешування буде повторювати процес заповнення та обробки пам'яті. Більша кількість ітерацій збільшує час обчислення і ускладнює атаку на геш.

– v: номер версії (version number) — це версія алгоритма «Argon2». Наприклад, поточна версія яка буде розгялнута в дослідженні буде вказана як $0x19$.

– y : тип «Argon2»: 0 для «Argon2d», 1 для «Argon2i», 2 для «Argon2id».

– $\langle P \rangle$: довжина пароля (length of the password) — довжина пароля, який буде гешуватися.

– P: пароль або парольна фраза (password) — сам пароль, який гешується.

– $\langle S \rangle$: довжина солі (length of the salt) — довжина випадкової солі, яке додається до пароля для збільшення ентропії.

– S: сіль (salt) — випадкове значення, що додається до пароля для запобігання атак за словником і попередньо обчислених таблиць (rainbow tables).

– $\langle K \rangle$: довжина секретного значення (length of the secret) — довжина секретного ключа.

– K: секретне значення (secret) — додатковий ключ, який може використовуватися для підвищення безпеки гешування.

– $\langle X \rangle$: довжина асоційованих даних (length of the associated data) — довжина додаткових даних, які можуть бути пов'язані з процесом гешування.

– X: асоційовані дані (associated data) — додаткові дані, які можуть бути включені в процес гешування для підвищення безпеки або для інших цілей.

«Argon2» заповнює пам'ять блоками $m' = \lfloor \frac{m}{4p} \rfloor \cdot 4p$ по 1024 байт. Для налаштовуваного паралелізму з p потоками пам'ять розглядається у

вигляді матриці $B[i][j]$ блоків із p рядками і $q = m'/p$ стовпцями. Позначаємо блок, створений на проході t, як $B^t[i][j], t > 0$. Блоки обчислюються наступним чином:

$$B^1[i][0] = H'(H_0 || \underbrace{0}_{\text{4 bytes}} || \underbrace{i}_{\text{4 bytes}}), \quad 0 \leqslant i < p;$$
$$B^1[i][1] = H'(H_0 || \underbrace{1}_{\text{4 bytes}} || \underbrace{i}_{\text{4 bytes}}), \quad 0 \leqslant i < p;$$
$$B^1[i][j] = G(B^1[i][j-1], B^1[i'][j']), \quad 0 \leqslant i < p, 2 \leqslant j < q,$$

де індекс блоку $[i'][j']$ визначається по-різному для «Argon2d» і «Argon2i», G — це функція стиснення, а H' — це геш-функція змінної довжини, побудована на основі H. Обидві функції G та H' будуть повністю визначені в подальшому тексті.

Якщо $t > 1$, повторюємо процедуру, але замість перезапису старих блоків ми застосовуємо операцію $\oplus$ до нових блоків із старими.

$$B^t[i][0] = G(B^{t-1}[i][q-1], B[i'][j']) \oplus B^{t-1}[i][0];$$
$$B^t[i][j] = G(B^t[i][j-1], B[i'][j']) \oplus B^{t-1}[i][j].$$

Тут блок $B[i'][j']$ може бути або $B^t[i'][j']$ для $j' < j$, або $B^{t-1}[i'][j']$ для $j > j'$.

Після виконання t ітерацій над пам'яттю ми обчислюємо фінальний блок B_{final} як результат операції $\oplus$ останньої колонки:

$$B_{\text{final}} = B^T[0][q-1] \oplus B^T[1][q-1] \oplus \cdots \oplus B^T[p-1][q-1].$$

Потім ми застосовуємо H' до B_{final}, щоб отримати вихідний тег (τ).

$$H'(B_{\text{final}}) \to \text{Tag}.$$

Геш-функція змінної довжини H' (Variable-length hash function). Нехай H_x буде геш-функцією з довжиною виходу x байтів (у нашому випадку «Argon2» використовує геш-функцію H_x — це «BLAKE2b» [6], яка підтримує довжину $1 \leqslant x \leqslant 64$). Визначимо H' наступним чином. Нехай V_i буде блоком довжиною 64 байтів, A_i — це перші 32 байта, а $\tau < 2^{32}$ буде 32-бітною довжиною тега (у форматі «little-endian») в байтах. Тоді:

$$
\begin{aligned}
&\text{if } \tau \leqslant 64 \\
&\qquad H'(X) \stackrel{\text{def}}{=} H_\tau(\tau||X). \\
&\text{else} \\
&\qquad r = \lceil \tau/32 \rceil - 2; \\
&\qquad V_1 \leftarrow H_{64}(\tau||X); \\
&\qquad V_2 \leftarrow H_{64}(V_1); \\
&\qquad \dots \\
&\qquad V_r \leftarrow H_{64}(V_{r-1}), \\
&\qquad V_{r+1} \leftarrow H_{\tau-32r}(V_r). \\
&\qquad H'(X) \stackrel{\text{def}}{=} A_1||A_2|| \dots A_r||V_{r+1}.
\end{aligned}
$$

На рисунку 2.2 показано, як геш-функція H' обчислюється шляхом послідовного застосування H_{64} до вхідного значення X і конкатенації результатів обчислень для отримання кінцевого гешу змінної довжини.

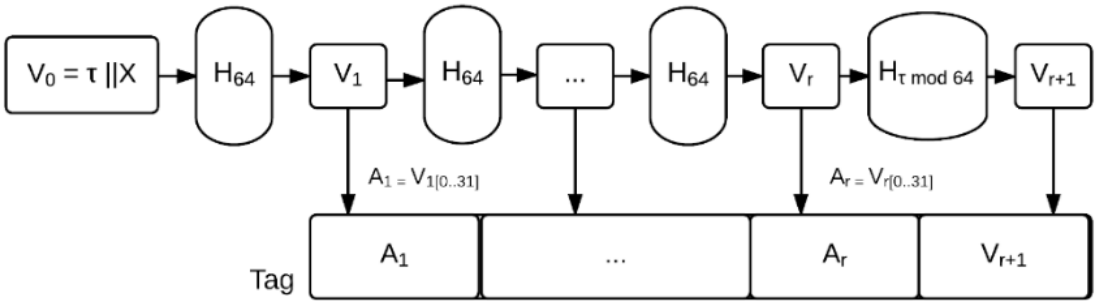


Рисунок 2.2 – Геш-функція змінної довжини H' в «Argon2».

Алгоритм забезпечує, що вихідний геш $H'(X)$ має потрібну довжину τ. Якщо $\tau \leqslant 64$, то використовується проста функція H_τ. Якщо $\tau > 64$, то хеш обчислюється через кілька ітерацій з використанням функції H_{64} і завершується функцією $H_{\tau-32r}$, де r залежить від τ. На рисунку 2.3 зображено один прохід «Argon2» з 4 стовпичками та p рядками.

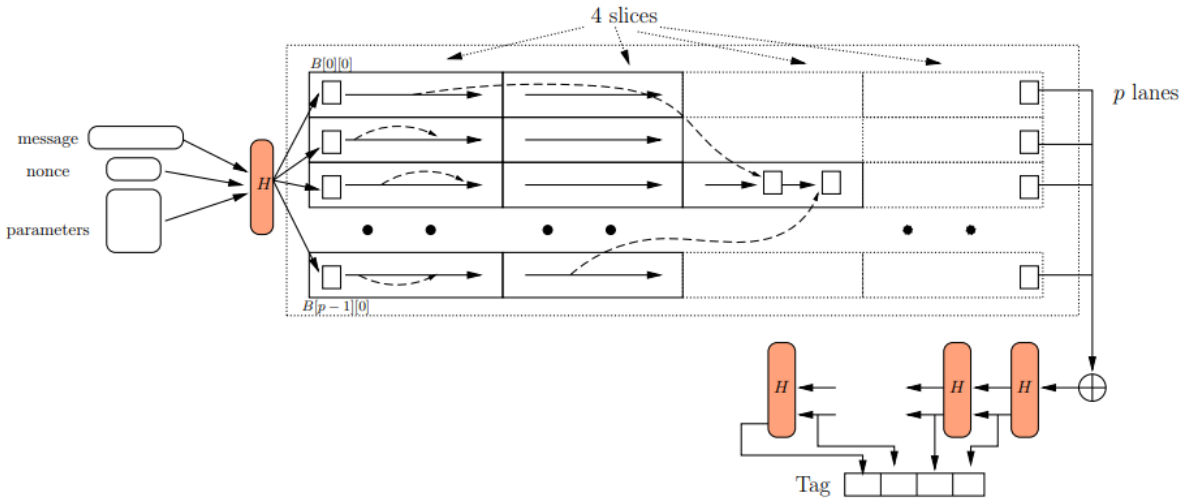


Рисунок 2.3 – Один прохід «Argon2» з 4 стовпчиками та p рядками.

2.1.3 Як відбувається індексація

Щоб забезпечити паралельне обчислення блоків, матрицю пам'яті додатково розділяється на $S = 4$ вертикальні стовпчики *слайси*. Перетин стовпчика і рядка є *сегментом* довжиною q/S. Сегменти одного і того ж слайсу обчислюються паралельно і не можуть посилатися на блоки один одного. Всі інші блоки можуть бути задіяні, тепер детально пояснимо цю процедуру.

Отримання двох 32-бітних значень [11]. В «Argon2d» обирається перші 32 біти блоку $B[i][j-1]$ і позначається це значення як J_1. Потім береться наступні 32 біти $B[i][j-1]$ і позначається це значення

як J_2. У «Argon2i» використовується G^2 — 2-раундова функцію стиснення G — в режимі лічильника, де перший вхідний блок є нульовим, а другий вхідний блок описується як:

$$(\underbrace{r}_{8\text{ bytes}} || \underbrace{l}_{8\text{ bytes}} || \underbrace{s}_{8\text{ bytes}} || \underbrace{m'}_{8\text{ bytes}} || \underbrace{t}_{8\text{ bytes}} || \underbrace{x}_{8\text{ bytes}} || \underbrace{i}_{8\text{ bytes}} || \underbrace{0}_{968\text{ bytes}}),$$

де

– r — номер проходу;

– l — номер лінії;

– s — номер слайсу;

– m' — загальна кількість блоків пам'яті;

– t — загальна кількість проходів;

– x — тип функції Argon (дорівнює 1 для «Argon2i» та 0 для «Argon2d»);

– i — лічильник, який починається в кожному сегменті з 1.

Всі числа подані в форматі «little-endian». Далі збільшуємо лічильник, щоб кожне застосування G^2 давало 128 64-бітних значень $J_1||J_2$.

Процес прив'язки обчислених значень J_1, J_2 до індексу блоку пам'яті до якого звертаються або *mapping (англ.)*. Значення $l = J_2 \bmod p$ визначає індекс лінії, з якої буде взятий блок. Якщо працюємо з першим слайсом і першим проходом ($r = s = 0$), то l встановлюється на поточний індекс лінії.

В алгоритмі Argon2 важливо визначити, які блоки пам'яті можуть бути використані для обчислення поточного блоку пам'яті $B[i][j]$. Це визначається за певними правилами, залежно від того, чи є поточна лінія l тією ж самою, в якій знаходиться поточний блок. Визначимо набір індексів як $\mathcal{R}$, які можуть бути використані для даного блоку пам'яті $B[i][j]$ відповідно до наступних правил:

1) Якщо l є поточною лінією, тоді $\mathcal{R}$ включає всі блоки, обчислені в цій лінії, які ще не були перезаписані, за винятком $B[i][j-1]$.

2) Якщо l не є поточною лінією, тоді $\mathcal{R}$ включає всі блоки в останніх $S - 1 = 3$ сегментах, обчислених і завершених у лінії l. Якщо $B[i][j]$ є першим блоком сегмента, тоді останній блок з $\mathcal{R}$ виключається.

Обираємо блок з множини $|\mathcal{R}|$ з нерівномірним розподілом $[0..|\mathcal{R}|)$:

$$J_1 \in [0..2^{32}) \to |\mathcal{R}| \left(1 - \frac{(J_1)^2}{2^{64}}\right).$$

Нерівномірний розподіл досягається за допомогою фактора $1 - \frac{(J_1)^2}{2^{64}}$, що робить деякі блоки більш імовірними для вибору, ніж інші. Щоб уникнути не цілих обчислень, використовується наступне цілочисельне наближення:

$$\begin{aligned} x &= (J_1)^2/2^{32}; \\ y &= (|\mathcal{R}| * x)/2^{32}; \\ z &= |\mathcal{R}| - 1 - y. \end{aligned}$$

Перераховуючи блоки в $\mathcal{R}$ в порядку їх створення, вибираємо z-й блок з них як опорний блок.

2.1.4 Функція стиснення G

Функція стиснення G побудована на основі геш-функції «BLAKE2b» [6] раундової функції $\mathcal{P}$ (повністю визначена тут...).$\mathcal{P}$ працює з вхідним значенням розміром 128 байт, яке можна розглядати як 8 або 16-байтних регістрів:

$$\mathcal{P}(A_0,A_1,\ldots,A_7) = (B_0,B_1,\ldots,B_7).$$

Функція стиснення $G(X,Y)$ працює з двома блоками розміром 1024 байтів X та Y. Спочатку обчислюється $R = X \oplus Y$. Потім R розглядається як матриця 8×8 з 16-байтних регістрів $R_0, R_1, \ldots, R_{63}$. Потім $\mathcal{P}$ спочатку

застосовується по рядках, а потім по стовпцях, щоб отримати Z:

$$(Q_0,Q_1,\ldots,Q_7) \leftarrow \mathcal{P}(R_0,R_1,\ldots,R_7);$$
$$(Q_8,Q_9,\ldots,Q_{15}) \leftarrow \mathcal{P}(R_8,R_9,\ldots,R_{15});$$
$$\ldots$$
$$(Q_{56},Q_{57},\ldots,Q_{63}) \leftarrow \mathcal{P}(R_{56},R_{57},\ldots,R_{63});$$

$$(Z_0,Z_8,Z_{16},\ldots,Z_{56}) \leftarrow \mathcal{P}(Q_0,Q_8,Q_{16},\ldots,Q_{56});$$
$$(Z_1,Z_9,Z_{17},\ldots,Z_{57}) \leftarrow \mathcal{P}(Q_1,Q_9,Q_{17},\ldots,Q_{57});$$
$$\ldots$$
$$(Z_7,Z_{15},Z_{23},\ldots,Z_{63}) \leftarrow \mathcal{P}(Q_7,Q_{15},Q_{23},\ldots,Q_{63}).$$

Нарешті, G виводить $Z \oplus R$ (схематично зображено на рисунку 2.4.):

$$G: \quad (X,Y) \;\rightarrow\; R = X \oplus Y \;\xrightarrow{\mathcal{P}}\; Q \;\xrightarrow{\mathcal{P}}\; Z \;\rightarrow\; Z \oplus R.$$

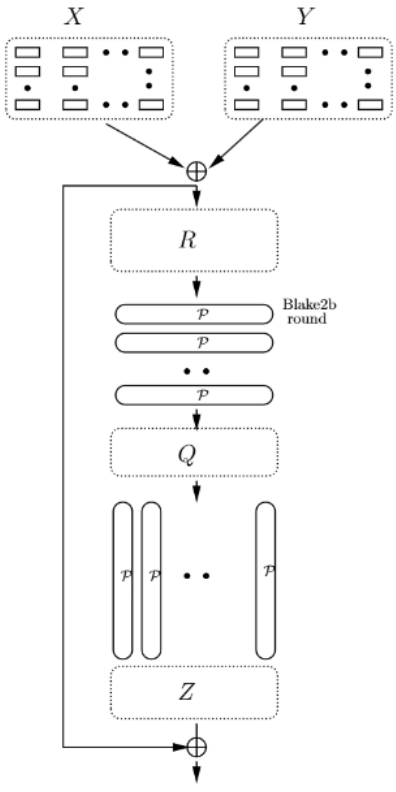


Рисунок 2.4 – Функція стиснення G в «Argon2».

2.2 Атака на ітеративну функцію стиснення

Розглянемо наступну структуру функції стиснення $F(X,Y)$, де X і Y є вхідними блоками:

– Вхідні блоки розміром t діляться на коротші підблоки довжиною t' (наприклад, 128 біт) X_0, X_1, $X_2, \ldots$ та Y_0, Y_1, $Y_2, \ldots$.

– Вихідний блок Z обчислюється по підблоках:

$$Z_0 = G(X_0,Y_0);$$
$$Z_i = G(X_i,Y_i,Z_{i-1}),; i > 0.$$

Ця схема нагадує *дуплексний режим автентифікованого шифрування (duplex authenticated encryption mode [14])* — це криптографічна схема, яка дозволяє одночасно здійснювати шифрування і автентифікацію даних. Він є розвитком стандартного блочного шифрування та забезпечує більш високий рівень безпеки для передачі даних, який є безпечним за певних припущень щодо G. Однак вона абсолютно небезпечна проти противників, які використовують атаки із компромісом, як показано нижче.

Припустимо, що супротивник обчислює $Z = F(X,Y)$, але Y не зберігає. Припустимо, що Y є деревоподібною функцією від збережених елементів глибиною D. Супротивник починає з обчислення Z_0, що вимагає тільки Y_0. У свою чергу, $Y_0 = G(X_0', Y_0')$ для деяких X',Y'. Таким чином, супротивник обчислює дерево тієї ж глибини D, але з функцією G замість F. Тоді Z_1 є деревоподібною функцією глибиною $D + 1$, Z_2 глибиною $D + 2$ і так далі. В цілому, перевизначення займає $(D + s)L_G$ часу, де s — кількість підблоків, а L_G — затримка функції G. Це слід порівняти з повною реалізацією, яка займає час sL_G. Таким чином, якщо пам'ять зменшується у факторі q, то добуток час-площа змінюється як

$$AT_{new} = \frac{D(q) + s}{sq} AT,$$

де A (Area) — це площа, яка використовується для збереження пам'яті. В контексті криптографічних функцій, площа може відображати кількість апаратних ресурсів, необхідних для збереження всіх проміжних даних. T (Time) — час, необхідний для виконання обчислень. В даному контексті, це час, який потрібен для виконання криптографічної функції. Рівняння описує компроміс між часом і пам'яттю (time-area tradeoff), коли супротивник зменшує обсяг використовуваної пам'яті у факторі q, збільшуючи при цьому час обчислень. Таким чином, AT є добутком часу та площі у первісній реалізації алгоритму, а AT_{new}— отриманий добуток часу та площі у новій реалізації, де використовується менше пам'яті, але обчислення займають більше часу.

Отже, якщо

$$D(q) \leqslant s(q-1), \tag{2.1}$$

то супротивник виграє.

Іншими словами, якщо супротивник може зменшити обсяг пам'яті у q разів таким чином, що глибина дерева обчислень $D(q)$ залишається в межах допустимого значення, атака буде успішною.

Можна зробити висновок, що ітеративна функція F є небезпечним варіантом, оскільки супротивник може зберігати всі проміжні значення і успішно здійснювати атаку. Зазначимо, що ця атака також застосовувалась до інших кандидатів (PHC) з ітеративною функцією стиснення.

2.3 Криптографічна стійкість «Argon2»

Тепер розглянемо стійкість до знаходження прообразу та колізій «Argon2». Вхідні дані змінної довжини доповнюються їхньою довжиною, що повинно забезпечити відсутність однакових вхідних рядків. Доповнення вхідних даних їхньою довжиною створює унікальність для кожного вхідного рядка, що ускладнює спроби підбору однакових гешів для різних наборів вхідних даних. Використання криптографічної

геш-функції, яка є стійкою до колізій, гарантує, що навіть найменші зміни у вхідних даних призведуть до значних змін у геші (*лавинний ефект*). Це ускладнює процес знаходження вхідних даних, які дають однаковий хеш, тобто ускладнює виконання атак повного перебору.

Стійкість до колізій. На початку введені дані обробляються геш-функцією H, яка визначена як функція «BLAKE2b» і вважається криптографічно безпечною. Схема гешування «Argon2» за твердженням ...(тут силка). є внутрішньо стійкою до колізій, навіть якщо компресійна функція G не є стійкою до колізій або пошуку прообразу. Оскільки супротивник не має контролю над вхідними даними функції G, колізії є дуже малоймовірними.

Розглянемо, наприклад, що супротивник знайшов колізію в функції G для двох довільних значень x_1, x_2 таких, що $G(x_1) = G(x_2)$. Для того щоб перетворити це на колізію для всієї схеми, супротивнику доведеться знайти відповідний прообраз для $H(X)$ (де X - це деяке значення, визначене шляхом *'зворотного відстеження'* m_1 або m_2 через алгоритм), метод *'зворотного відстеження'* означає, що значення X було знайдене за допомогою методу, який називається зворотним відстеженням або зворотною трасировкою. У контексті криптографічних атак це означає, що супротивник намагається знайти попередні значення в алгоритмі, які привели б до отримання конкретного результату, що є неможливим, враховуючи, що H є криптографічно безпечною геш-функцією [16] . Крім того, вживаються заходи для уникнення колізій блоків, ініціалізуючи стартові блоки пам'яті за допомогою лічильника і забезпечуючи, щоб перший блок у будь-якій лінії не міг посилатися (через функцію індексування ϕ) на останній блок попередньої лінії.

Стійкість до пошуку прообразу. Як зазначено вище, внутрішня функція стиснення G не претендує на стійкість до пошуку прообразу, проте попередня обробка введених користувачем даних криптографічно безпечною геш-функцією H робить загальну схему стійкою до пошуку прообразу. Наприклад, припустимо, що супротивник здатний знайти x

таке, що $G(x) = h$ для будь-якого довільного h. Щоб продовжити цю атаку пошуку прообразу на загальну схему, супротивник зрештою повинен знайти прообраз n таке, що $H(n) = x'$, що є неможливим, враховуючи, що H є криптографічно безпечною [16].

Стійкість до пошуку другого прообразу: Подібно до зазначених властивостей, розглянемо, що супротивник здатний здійснити атаку на знаходження другого прообразу на функцію стиснення G, зумівши знайти деяке x_2, таке що $G(x_1) = G(x_2)$, $x_1 \neq x_2$ для будь-яких x_1. Щоб продовжити цю атаку на загальну схему, супротивник зрештою повинен буде знайти n таке, що $H(n) = x_2'$ (тобто знайти прообраз для H).

На момент написання цієї роботи , немає згадок про практичні криптографічні слабкості як і у функції «BLAKE2b», так і в конструкції «Argon2» в цілому. Той факт, що «Argon2» не використовує конструкцію *Merkle-Damgård*[1] у своєму дизайні, також усуває загрозу атак (наприклад, атак на розширення довжини), специфічних для цієї конструкції. Крім того, доступний криптоаналіз, пов'язаний з «BLAKE2», показує, що навіть атаки, націлені на скорочені версії, залишаються суто теоретичними. Крім того, у фінальному звіті «NIST» [17] за результатами конкурсу «SHA-3» зазначається, що геш-функція «BLAKE2b» пропонує «дуже великий запас безпеки» , що, враховуючи ретельний аналіз, пов'язаний з процесом «SHA-3», є добрим знаком.

Отже, здається доцільним зробити висновок, що, крім забезпечення традиційних властивостей безпеки, пов'язаних з криптографічними геш-функціями, наразі практичні криптографічні слабкості в схемі «Argon2» відсутні.

2.4 Захист від атак з використанням таблиць пошуку та атак компромісу між часом і пам'ятю.

Коли використовуються одноразові значення солі в «Argon2» (належним чином), однакові повідомлення будуть створювати різні теги для різних одноразових значень. Атаки компромісу між часом і пам'ятю (надалі буде позначати як *АКЧП*) вимагатимуть попереднього обчислення таблиць пошуку для всіх можливих одноразових значень солі, що, враховуючи рекомендовану довжину солі у 16 байтів, означає попереднє обчислення (і зберігання) 2^{128} таблиць пошуку, кожна з яких буде займати (щонайменше) кілька гігабайтів. Це, звичайно, передбачає правильне генерування одноразових значень солі (бажано з використанням «CSPRNG» [15]). Якщо генерування солі відбувається таким чином, що більшість гешів «Argon2» створюється з використанням одного й того ж (передбачуваного) одноразового значення, то АКЧП можуть стати більш здійсненними. Це стосується АКЧП (таких як *rainbow tables*), які націлені на схему гешування паролів загалом, а не на ті, що прагнуть зменшити обмеження стійкості по пам'яті, як зазначено в підрозділі нижче.

2.4.1 Стійкість від атак з використанням центрального процесора (CPU) та стійкість до атак з використанням великої кількістю пам'яті

При захисті від оптимізованих спеціальних програмних зламувальників слід зазначити, що хоча стійкість від атак з використанням центрального процесора та стійкість від атак з використанням великої кількістю пам'яті можуть бути важливими для схеми «Argon2», основна безпека, яку вона забезпечує, все ще є функцією

її часу та пам'яті. Таким чином, залишається за користувачем визначити їхнє використання (та відповідну модель загроз) і відповідним чином налаштувати параметри. Стійкість від атак з використанням ЦП та стійкість до атак з використанням великої кількістю пам'яті лише гарантують, що схема відповідає заявленим вимогам безпеки її параметрів (тобто немає "швидких шляхів"для зламу паролів).

Стійкість від атак з використанням центрального процесора (CPU). Схема «Argon2» оптимізована для архітектури x86 (покладаючись на кеш та організацію пам'яті сучасних процесорів Intel та AMD) і спеціально розроблена так, щоб бути несприятливою для реалізацій на GPU/FPGA/ASIC. GPU, FPGA, і ASIC — це різні типи апаратних засобів, які використовуються для обробки даних і виконання обчислень, кожен з яких має свої особливості та застосування.

1) GPU (Graphics Processing Unit): GPU мають велику кількість ядер (тисячі), що дозволяє їм виконувати паралельні обчислення дуже ефективно. Це робить їх також корисними для задач загального призначення, таких як машинне навчання, обробка великих даних та криптографія.

2) FPGA (Field-Programmable Gate Array): FPGA дозволяють створювати специфічні апаратні прискорювачі для різних задач, маючи високу гнучкість у програмуванні та конфігурації. Вони часто використовуються в розробці спеціалізованих обчислювальних пристроїв, таких як цифрові сигнальні процесори, мережеві пристрої та криптографічні модулі.

3) ASIC (Application-Specific Integrated Circuit): ASIC забезпечують високу продуктивність та енергоефективність для конкретних задач, оскільки вони оптимізовані для цих задач на рівні апаратного забезпечення. Проте вони не піддаються змінам після виготовлення, що робить їх менш гнучкими у порівнянні з FPGA. Приклад застосування: виготовлення процесорів для майнінгу криптовалют, спеціалізованих контролерів.

Схеми гешування паролів, однак, стикаються з оптимізованими реалізаціями для конкретних CPU, які можуть використовувати цю вбудовану спорідненість платформи. Ефективність таких реалізацій обмежується як обмеженнями пам'яті (доступною для роботи CPU), так і кількістю доступних ядер CPU архітектури x86 за умови відсутності (достатньо ефективної) АКЧП. Розробники «Argon2» передбачили підтримку паралелізму для багато-ядерних CPU (щоб дозволити збільшення пропускної здатності на призначеній платформі захисту), але як було зазначено перед цим, «Argon2» підтримує ступінь паралелізму p таким чином, що запобігає ефективним АКЧП, обмежуючи внутрішній паралелізм сегментами (взаємодія між lane/slice). В описі схеми «Argon2» було вибрано розмір сегменту $S = 4$, оскільки він накладає низькі синхронізаційні вимоги, водночас значно збільшуючи часові штрафи для оптимізації проти АКЧП. Враховуючи, що «Argon2» передбачає використання більше 60 ядер на сучасних процесорах Intel x86 з SIMD інструкціями. *SIMD (Single Instruction, Multiple Data)* - це метод обробки даних, який дозволяє одній інструкції працювати одночасно з кількома даними. SIMD використовується для паралельної обробки масивів даних, що значно підвищує продуктивність у задачах, які можуть бути паралелізовані, таких як обробка відео, аудіо, графіки, наукові розрахунки тощо. У 3 розділі будуть наведині дві інструкції SIMD, це SSE2 (Streaming SIMD Extensions 2), та AVX (Advanced Vector Extensions). Все це разом з додатковими інвестиціями в пам'ять (навіть з урахуванням обмежень з боку гешування паролів) робить «Argon2» спеціально стійким до АКЧП і дуже ефективним у плані оптимізації використання ресурсів CPU.

Складність пов'язана з пам'ятю (Memory-Hardness) При захисті від зловмисників з апаратною підтримкою їх можна умовно розділити на тих, хто має обмежений бюджет і вибирає реалізації на GPU та FPGA, та тих, хто має значні бюджети і вибирає більш ефективні (але також і більш дорогі) реалізації на ASIC. З різних причин (таких як

площа, яку займає пам'ять, висока затримка пам'яті в GPU, вищі витрати на виробництво пам'яті тощо) правильно спроектовані схеми з інтенсивним використанням пам'яті обмежують ефективність реалізацій на апаратній підтримці таким чином, що вони не будуть ні дешевшими, ні швидшими. У (статі розробників) зазначено, що будь-яка схема, яка використовує більше кількох сотень МБ оперативної пам'яті, майже напевно неефективна для реалізацій на GPU або FPGA.

Розробники «Argon2» стверджують [18] ,що для обох варіантів схеми «Argon2d» та «Argon2i», при використанні стандартної кількості ітерацій, супротивник із обладнанням ASIC не може зменшити добуток "площа-час"AT за умови, що пам'ять зменшується в 4 або більше разів без високих штрафів, що застосовуються при більшій кількості ітерацій через пам'ять. Зокрема, існує атака з використанням рангового індексування яка розглянута тут [19], яка спрямована як на залежні від даних (dMHFs), так і на незалежні від даних (iMHFs) варіанти функції індексування ϕ в «Argon2». Ця атака показує, що «Argon2d» є більш вразливим до АКЧП через залежність від даних функції індексування, тому для нього використовується 3 ітерації, а не 1, за замовчуванням. Розробники «Argon2» виявили, що зменшення пам'яті в 3 рази вже означатиме, що площа, зайнята ядрами обчислення «BLAKE2b», перевищує вимоги до пам'яті в 1 ГБ, що робить її верхньою межею ефективності АКЧП незалежно від варіанту схеми.

На додаток до атак, обговорених у вище, останні роботи `Corrigan-Gibbs та ін.` [20] стверджують, що АКЧП можуть зменшити пам'ять у 5 разів у випадку одноразового проходу «Argon2i» та у 2.72 для рекомендованого варіанту з кількома проходами без додаткових штрафів за час. Ці атаки спираються на згаданий вище факт, що для «Argon2i» адреси блоків відомі заздалегідь, і таким чином блоки, які не використовуються в даний момент, можна вилучити (до їх перезапису на пізнішій стадії), що дозволяє запустити алгоритм із меншими вимогами до пам'яті без відповідного збільшення часу. У дискусії на PHC mailing

list [21]. була запропонована ідея XOR через блоки пам'яті (як у схемах *Lyra2* [22] і *Gambit*) [23], яка забезпечує запобігання знаходженню блоків у стані "які не використовується"протягом певного часу.

2.5 Загальні висновки

Сімейство схем гешування паролів «Argon2» є дуже універсальним і може функціонувати як схема гешування паролів або функція формування ключів за різних умов і на різних апаратних налаштуваннях користувача. Однак ця універсальність також має свої недоліки. Хоча в специфікаціях вказані деякі рекомендовані параметри (щодо використання пам'яті, варіанта схеми та ступеня паралелізму) для різних сценаріїв використання, ці рекомендації є досить загальними, і як апаратні налаштування (доступний процесор і ОЗП), так і сценарії використання (та відповідна модель загроз) можуть відрізнятися.

Це може спокусити розробника програмного забезпечення, який не знайомий з деталями схеми гешування паролів і прагне оптимізувати продуктивність, встановити значення параметрів (щодо використання пам'яті та кількості ітерацій), які пропонують недостатній захист в контексті їх моделі загроз.

Іншими словами, універсальність, яку пропонує схема, може надати деяким користувачам *«достатньо мотузки, щоб повісити себе»*. Замість того, щоб зменшувати цю універсальність (що могло б негативно вплинути на широке впровадження схеми), було б доцільніше провести дослідження, оцінюючи різні сценарії використання з різними моделями загроз і різними апаратними налаштуваннями, та виділити рекомендовані параметри для всіх цих випадків як довідник для менш обізнаних у криптографії розробників.

Нарешті, слід зазначити, що схеми гешування паролів загалом не є повним вирішенням проблем, властивих автентифікації за допомогою паролів. Природа паролів, *«легко запам'ятовується, важко вгадати»*,

як правило, призводить до того, що люди обирають паролі, які вони думають, що важко вгадати, але це не так. Важливо розуміти, що паролі не генеруються за критеріями, які зазвичай застосовуються до криптографічних ключів: оскільки їх обирають люди, їх ключовий простір та ентропія набагато нижчі, і вони часто включають передбачувані елементи, пов'язані з людьми, які їх обирають (імена, дати народження, улюблені числа тощо), що призводить до відносно малих ключових просторів для атаки. Для вирішення цієї проблеми було запропоновано різні рішення, такі як використання парольних фраз [24]. або метод створення парольних фраз *«diceware»*паролів [25]. (де паролі складаються з конкатенації кількох випадкових слів, обраних з грального кістки) та подібні схеми. До рішень на зразок багатофакторної автентифікації (MFA), де для автентифікації потрібні кілька токенів (наприклад, пароль, поєднаний з апаратним токеном або кодом, надісланим по каналу поза мережі. .Недоліком багатьох з цих рішень є те, що вони або залишають відповідальність за безпечну генерацію паролів на кінцевих користувачах або тягнуть за собою додаткові витрати на розробку та інфраструктуру.

Паролі або фрази, навіть якщо вони складаються з конкатенації кількох (індивідуально легких для запам'ятовування) слів, недостатні проти хитрих зловмисників. Вони можуть генерувати кандидати на паролі на основі зібраної особистої інформації [24]. або використовувати розумні, адаптивні стратегії або техніки зламу паролів для оптимізації процесу зламу через оптимізований порядок вгадування [26], зменшення простору паролів [27] або виявлення патернів, присутніх у найслабших паролях у базі даних, та екстраполяції їх на інші паролі за допомогою масок [28] або тематичних словників [29]. Крім того, навіть якщо люди обирають складний пароль, повсюдне використання паролів повторно [30]. також залишається загрозою. Використання одного і того ж складного пароля у багатьох місцях означає, що компрометація облікових даних одного сервісу напряму веде до компрометації інших (можливо

високобезпечних) сервісів. Щоб погіршити ситуацію, навіть якщо служба автентифікації використовує криптографічно безпечну схему гешування паролів, зловмисник може легко обійти це, підлаштовуючи функціонал логування і записуючи текстові версії паролів перед їх передачею до гешуючої функції. У такому сценарії ні складні паролі, ні безпечні схеми гешування паролів не надають жодного захисту, а тенденція до повторного використання паролів призводить до впливу на інші сервіси також.

Висновки до розділу 2

Хоча злам паролів був би набагато менш успішним за наявності безпечної схеми гешування паролів, як «Argon2», не слід вважати це повністю вирішенням всіх проблем, пов'язаних з паролями в цілому (особливо перед обличчям обізнаних та добре оснащених зловминсиків). Де це можливо, схеми аутентифікації слід доповнювати деякою формою багатофакторної аутентифікації, щоб принаймні ускладнити зусилля атакуючого у досягненні несанкціонованої аутентифікації.

3 ПРОВЕДЕННЯ ЕКСПЕРЕМЕТАЛЬНОГО ДОСЛІДЖЕННЯ

В цьому розділі будуть наведені результати тестів експериментального дослідження.

3.1 Вибір параметрів для дослідження (Експеримент №1)

Першим етапом дослдіження буде вибір параметрів схеми гешування «Argon2» обох версій «Argon2d» та «Argon2i». Експеримент буде проведено на двох комп'ютерах з різними процесорами різних потужностей. Перший комп'ютер на базі Intel, Lenovo Legion 5 зі встановленим процесором i5-10300H з базовою частотою 2.5GHz та максимальною 4.5GHz. Другий комп'ютер HP Pavilion 15 з процесором AMD Ryzen 5 5500u with Radeon Graphics з базовою частотою 2.1GHz та максимальною 4GHz.

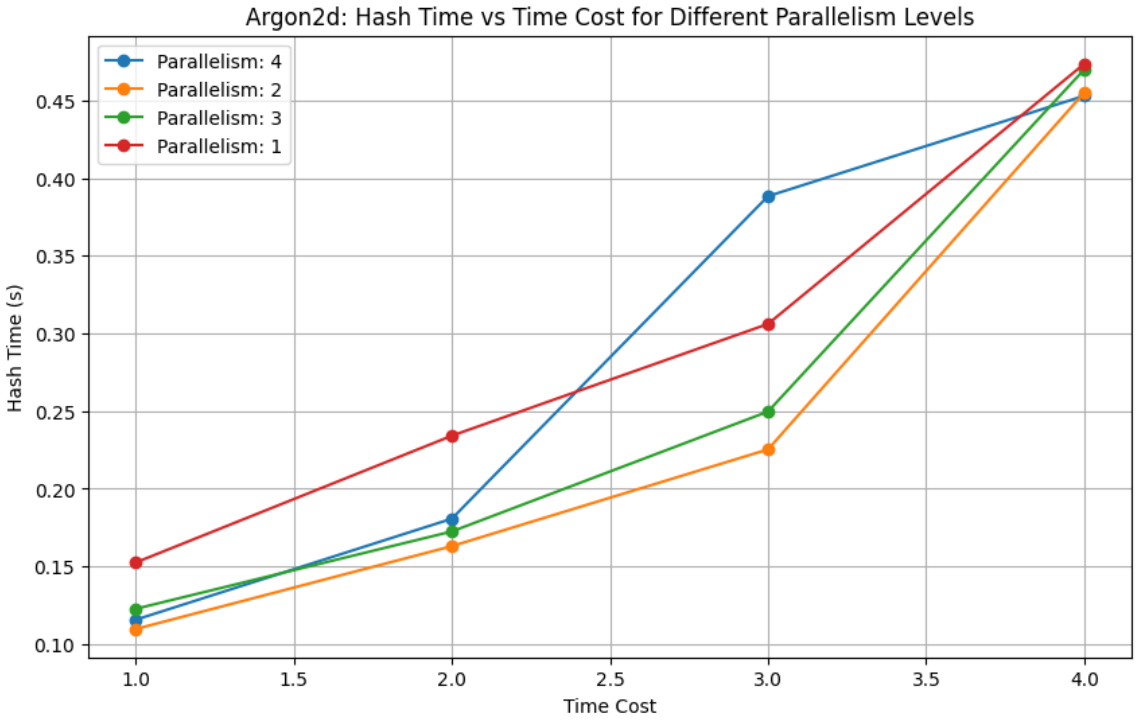
Метою даного експерименту є визначення оптимальних параметрів гешування для алгоритмів «Argon2d» та «Argon2i» при фіксованих витратах на пам'ять, рекомендовано $m = 2^{16}$ кілобайт. Експеримент спрямован на знаходження параметрів t(кількість ітерацій) та p (паралелізм), які забезпечують мінімальний час гешування при макисмальних затратах на обчислення.

Методологія експерименту включає вибір параметрів t, m, p, що будуть позначені далі у таблицях як $t =$ `Time cost`, $m =$ `Memory cost`, `p= Parallelism`, процес гешування та аналіз результатів. Пам'ять зафіксована на рівні 2^{16} кілобайт. Вибрані значення для `Time cost` (кількість ітерацій): 1, 2, 3, 4, а для `parallelism` (паралелізм): 1, 2, 3, 4. Використовується бібліотека `argon2` з реалізацією алгоритмів «Argon2d» та «Argon2i» на мові програмування Python3. Для кожної комбінації параметрів обчислюється час гешування пароля

Argon2hash1. Вимірюється час гешування для кожної комбінації параметрів. Проведено порівняння алгоритмів «Argon2d» та «Argon2i» для кожної конфігурації, результати дослідження збережені у вигляді таблиць та відсортовані за швидкістю гешування.

Таблиця 3.1 – Результати швидкості гешування Argon2d

Time Cost	**Memory Cost**	**Parallelism**	**Hash Time (s)**
1	65536	2	0.109632
1	65536	4	0.115619
1	65536	3	0.122648
1	65536	1	0.152478
2	65536	2	0.163026
2	65536	3	0.172481
2	65536	4	0.180700
2	65536	1	0.234146
3	65536	2	0.225340
3	65536	3	0.249689
3	65536	1	0.306136
3	65536	4	0.388421
4	65536	4	0.453068
4	65536	2	0.455075
4	65536	3	0.469998
4	65536	1	0.473622



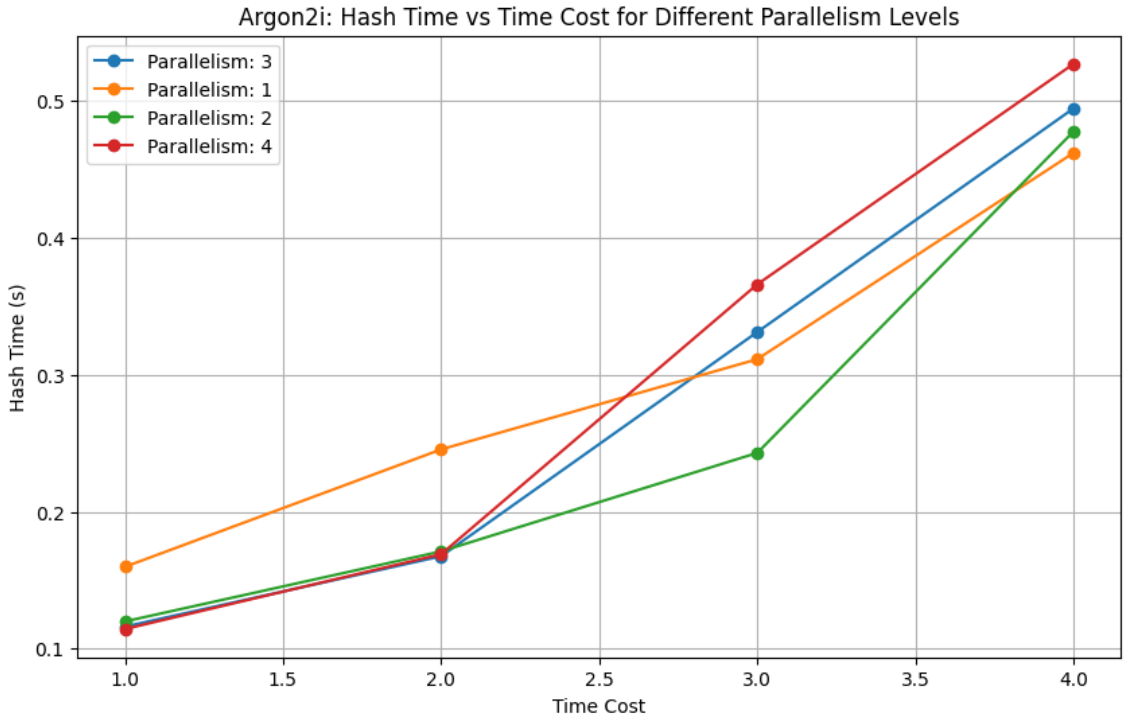
– **Залежність від `Time Cost` (t)** : На графіку видно, що зі збільшенням значення `Time Cost` (t) час гешування `(Hash Time)` зростає для всіх рівнів паралелізму. Це вказує на значну залежність часу гешування від ітерацій t.

– **Залежність від `Parallelism` (p)** : Лінії для різних рівнів паралелізму (1, 2, 3 і 4) розташовані досить близько одна до одної, особливо при Time Cost 1 і 2. Це свідчить про те , що збільшення паралелізму (p) не призводить до такого значного збільшення часу гешування порівняно зі збільшенням ітерацій.

Швидкість гешування більше залежить від кількості ітерацій, ніж від рівня паралелізму. Збільшення кількості ітерацій викликає більш значне збільшення часу гешування порівняно зі збільшенням рівня паралелізму.

Таблиця 3.2 – Результати швидкості гешування Argon2i

Time Cost	**Memory Cost**	**Parallelism**	**Hash Time (s)**
1	65536	4	0.114364
1	65536	3	0.116136
1	65536	2	0.119918
1	65536	1	0.159817
2	65536	3	0.167551
2	65536	4	0.169017
2	65536	2	0.171101
2	65536	1	0.245577
3	65536	2	0.243057
3	65536	1	0.311479
3	65536	3	0.331321
3	65536	4	0.366157
4	65536	4	0.462150
4	65536	2	0.477759
4	65536	3	0.494617
4	65536	1	0.527147



Порівнюючи два графіки швидкості гешування для «Argon2d» і «Argon2i», можна зробити такі висновки щодо параметрів:

– **Залежність від `Time Cost (t)`** : Для обох графіків видно, що збільшення ітерацій гешування призводить до збільшення часу гешування для всіх рівнів паралелізму. Це вказує на значну залежність часу гешування від параметра `Time Cost (t)`.

– **Залежність від `Parallelism (p)`** : У «Argon2d» паралелізм має більш помітний вплив на час гешування при низьких значеннях `Time Cost`. При збільшенні ітерацій вплив паралелізму згладжується.

Нормалізація даних:

Далі результати нормалізуються за часом та витратами, оцінюється загальний показник (Score) для кожної комбінації параметрів. Найкращі конфігурації параметрів визначаються за мінімальним часом гешування при максимальних затратах.

Таблиця 3.3 – Результати гешування з фіксованим значенням пам'яті (2^{16}) кб

Time Cost	Parallelism	Argon2 Type	Hash Time (s)	Score
3	3	Argon2d	0.391951	0.475000
4	3	Argon2d	0.422712	0.437500
2	2	Argon2i	0.340400	0.413693
4	1	Argon2i	0.334671	0.366706
2	4	Argon2d	0.334671	0.329206
4	2	Argon2d	0.334671	0.329206
1	2	Argon2d	0.257344	0.313655
4	4	Argon2i	0.334671	0.254206
1	4	Argon2i	0.256173	0.236716
1	1	Argon2d	0.164382	0.197212

Бачимо, що найкращий результат серед протестованих даних показав випадок при кількості ітерацій 3 та паралелізм 3 при фіксованому значенні пам'яті 2^{16} кб для «Argon2d» та при кількості ітерацій 2 та паралелізм 3 для «Argon2i». В насутпних експериментах дослідимо швидкість перебору паролів для «Argon2d» та «Argon2i» саме на параметрах: $t = 3, p = 3, m = 2^{16}$

3.2 Налаштування вірутального середовища та програмного забеспечення John the Ripper (Експеримент №2)

Розглянемо процес встановлення та використання віртуальної машини VirtualBox, а також встановлення операційної системи Kali Linux та інструменту для злому паролів John the Ripper. Інструкцію по встановленню віртуальної машини можна знайти тут [31]. На обох комп'ютерах встановлення остання версія `VirtualBox 7.0` для Windows 11. Інструкцію по встановленню ОС Kali Linux також можна знайти тут [32]. На обох комп'ютерах встановлення остання версія ОС Kali Linux, яка

є акутальною версію на червень 2024 року.

Наступнним кроком буде встановлення спеціального програмного забеспечення `John the Ripper`. Інструкцію по встановленню також можна знайти тут [33]. З метою обичислити середню швидкість перебору паролей було обрано пароль довжиною $P = 10$ символів, в якому є великі букви та цифри. Пароль: *Argon2hash1*. Геші пароля були отримані на сайті [34].

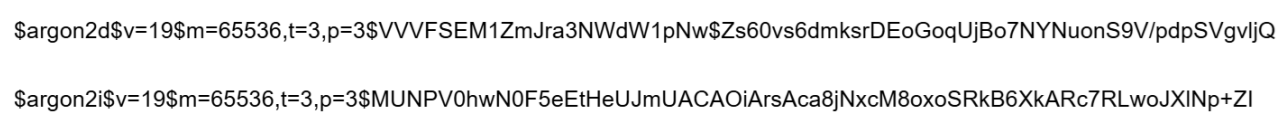


Рисунок 3.1 – Отримані геші для версій «Argon2d», «Argon2i»

Параметри для моделі атаки повного перебору за допомогою John the Ripper було обрано наступні:

В терміналі вводимо наступну команду:

```
john -incremental=ASCII -min-length=10 -max-length=10 test.txt
```

1) режим «incremental=ASCII» у John the Ripper — це один з режимів атаки, призначений для перебору паролів методом повного перебору з використанням усіх можливих комбінацій символів ASCII. Формат «incremental=ASCII» в John the Ripper використовує всі символи з набору ASCII. ASCII (American Standard Code for Information Interchange) — це стандартний набір символів, який включає в себе англійські букви (великі і малі), цифри, знаки пунктуації і деякі спеціальні символи. Загалом, це 128 символів (від 0 до 127), деякі з яких є непечатними контрольними символами. Щоб знайти загальну кількість можливих 10-символьних паролів, створених з символів ASCII, потрібно враховувати, що стандартний набір символів ASCII містить 128 символів. Кожна позиція у паролі може бути будь-яким із цих 128 символів. Відповідно, загальна кількість можливих комбінацій для 10-символьного

пароля буде:

$$|\Omega| = 128^{10}.$$

Обчислимо цю величину:

$$|\Omega| = 128^{10} = 2^{7\times 10} = 2^{70}.$$

Точне значення:

$$|\Omega| = 128^{10} = 1\ 180\ 591\ 620\ 717\ 411\ 303\ 424.$$

2) «min-length=10» та «max-length=10» це границі пошуку паролів довжиною 10 символів, це зроблено для того щоб John the Ripeer не починав перебирати паролі довжиною менші за 10 символів, а одразу приступав до пошуку 10-символьних паролів.

3) «Using default input encoding: UTF-8», використовується кодування UTF-8 для введення даних за замовчуванням.

4) інструкція SIMD: «SSE2». SSE2 (Streaming SIMD Extensions 2) — це набір інструкцій для процесорів, які призначені для покращення обчислювальних можливостей процесорів, особливо в задачах, пов'язаних з обробкою числових даних, таких як наукові розрахунки, мультимедіа та криптографія. SSE2 додає підтримку 128-бітових векторів, які можуть містити кілька елементів типу double (64-бітових) або кілька елементів типу int (32-бітових). Завдяки SIMD, процесори можуть виконувати одну і ту ж операцію над кількома даними одночасно, що значно підвищує продуктивність в задачах обробки масивів даних. SSE2 включає нові інструкції для арифметичних операцій, логічних операцій, зсувів та ін.

Все це зроблено для того, щоб умови були однакові і можна було порівняти отримані результати.

Мета експерименту №2: дослідити як кількість потоків та кількість загального кешу процесора впливають на швидкість перебору паролів. **Опишемо коротко експеримент:**

1) Запускаємо віртуальну машину починаючі з одним логічним

процесором.

2) В терміналі вводимо команду, яка була зазначена вище.

3) Протягом 4 хвилин заміряємо середню швидкість перебору паролів

4) Повторюємо процес 8 разів на кожному комп'ютері

5) Результати середньої швидкості перебору та максимальну частоту записуємо в таблиці

Таблиця 3.4 – Результати швидкості гешування та частоти ЦП для «Argon2d» на процесорі i5-10300H

Потоки	**Швидкість (паролі/с)**	**Частота ЦП (ГГц)**
1	7.481	4.19
2	15.07	4.20
3	18.15	4.19
4	20.63	4.18
5	21.48	4.19
6	22.44	4.18
7	22.24	4.18
8	21.94	4.18

Таблиця 3.5 – Результати швидкості гешування та частоти ЦП для «Argon2d» на процесорі AMD Ryzen 5 5500u

Потоки	**Швидкість (паролі/с)**	**Частота (ГГц)**
1	6.508	3.31
2	13.19	3.44
3	17.97	3.46
4	21.97	3.46
5	25.48	3.46
6	22.48	3.45
7	20.04	3.42
8	20.19	3.39

Таблиця 3.6 – Результати швидкості гешування та частоти ЦП для «Argon2i» на процесорі i5-10300H

Потоки	**Швидкість (паролі/с)**	**Частота ЦП (ГГц)**
1	7.4	4.24
2	12.81	4.20
3	18.88	4.23
4	20.88	4.22
5	22.09	4.20
6	22.50	4.18
7	22.89	4.20
8	20.10	4.20

Таблиця 3.7 – Результати швидкості гешування та частоти ЦП для «Argon2i» на процесорі AMD Ryzen 5 5500u

Потоки	**Швидкість (паролі/с)**	**Частота ЦП (ГГц)**
1	6.4	3.20
2	13.30	3.43
3	18.39	3.46
4	21.69	3.46
5	25.80	3.46
6	22.21	3.46
7	20.88	3.46
8	19.46	3.46

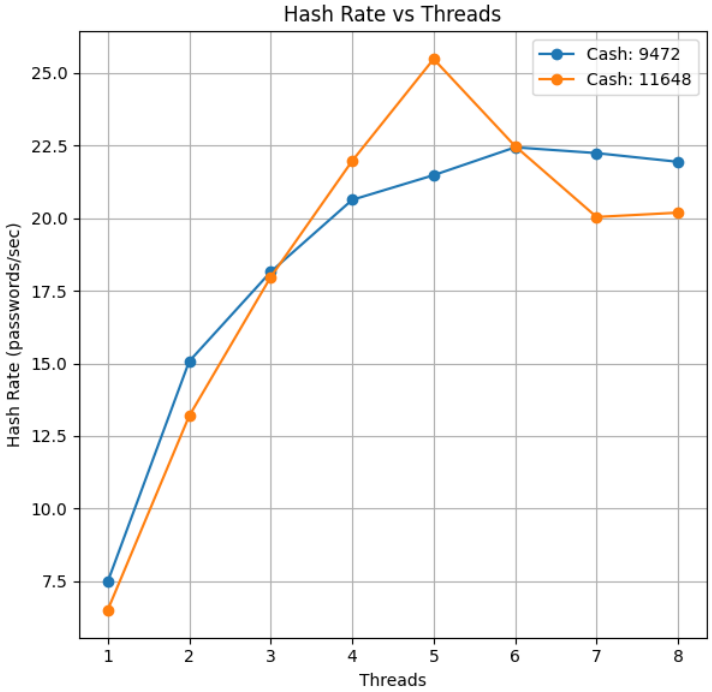


Рисунок 3.2 – Порівняння для версії «Argon2d»

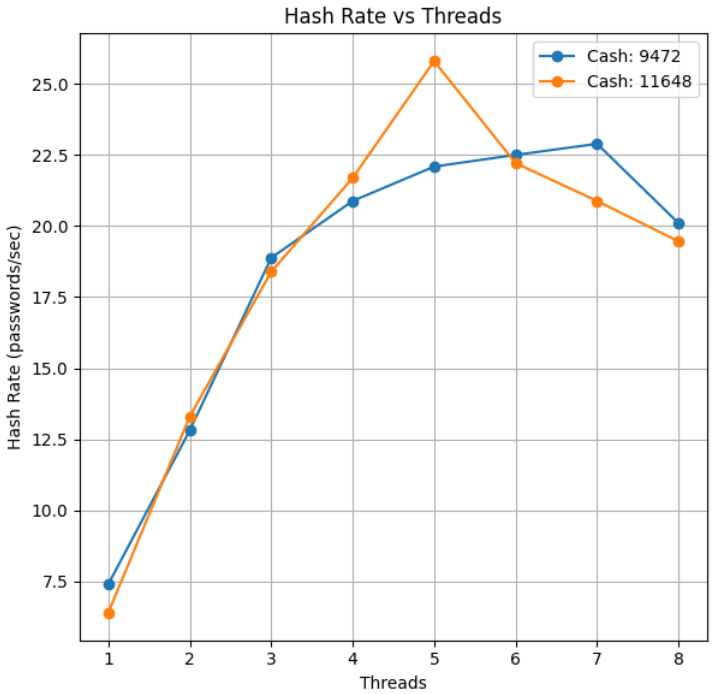


Рисунок 3.3 – Порівняння для версії «Argon2i»

Тут 9472 кб — це загальний кеш процесора i5-10300H, а 11648 кб — загальний кеш AMD Ryzen 5 5500u.

3.3 Теоретична оцінка вартості взлому

На основі попередніх результатів можна зробити припущення щодо можливостей зловмисника, який намагається взламати пароль повним перебором. Для цього Vojtěch Polášek [35] з університету Масарика факультету інформатики створив математичну модель ціноутворення, яка дозволяє оцінити витрати, пов'язані зі знаходженням правильного пароля. Ці витрати включають покупку пристроїв і витрати на електроенергію. Перераховані нижче змінні, визначені для формалізації моделі:

Можемо порахувати приблизно скільки часу нам знадобиться для знаходження пароля таким чином врахувавши, що швидкість перебору приблизно 22.89 паролів в секунду:

$$P = |\Omega| = 128^{10} = 2^{7\times 10} = 2^{70},$$

$$L = \frac{2^{70}(\text{p})}{\text{швидкість}(\text{p/s})}.$$

N – кількість машин, виражена цілим числом,

D – споживана потужність однієї машини, виражена в кіловатах,

E – ціна електроенергії, виражена в доларах за кіловат-годину,

F – очікувана кінцева вартість всієї атаки, виражена в доларах,

H – початкова вартість однієї машини (ЦП, ОЗП, аксесуари), виражена в доларах,

L – очікувана тривалість атаки, виражена в днях,

P – кількість паролів у вибраному просторі паролів, виражена цілим числом,

S – швидкість однієї машини, виражена як кількість секунд, витрачених на обчислення одного пароля.

Формула для знаходження загальної кількості машин:

$$N = \frac{\frac{S}{60\times60\times24} \times \frac{P}{2}}{L},$$

$$N = \frac{\frac{\frac{1}{22.89}}{60\times60\times24} \times \frac{2^{70}}{2}}{1344695212310} \approx 180.$$

Формула для оцінки кінцевої вартості всієї атаки:

$$F = (N \times H) + (N \times D \times E \times 24 \times L).$$

Припустимо , що вартість одного такого комп'ютера становить 1000\$, споживана потужність однієї машини 15 Вт, а ціна електроенергії становить 0.07\$ Кв за годину.

$$F = (180 \times 1000) + (180 \times 15 \times 0.07 \times 24 \times 1344695212310) \approx 6 \cdot 10^{15}\text{доларів.}$$

Ця сума є надто великою і робить такі атаки економічно невигідними для зловмисників. Витрати включають вартість апаратного забезпечення, електроенергії, охолодження, а також інші експлуатаційні витрати, які необхідні для підтримки обчислювальних ресурсів на необхідному рівні продуктивності протягом тривалого періоду часу.

Для більшості організацій та зловмисників такі витрати є непосильними і нераціональними. Інвестування в обладнання та ресурси для проведення подібної атаки не гарантує успіху і не виправдовує витрат. Замість цього, зловмисники можуть шукати менш витратні та більш доступні способи досягнення своїх цілей.

Використання алгоритмів гешування, таких як «Argon2», значно підвищує економічну доцільність безпеки паролів. Іншими словами, навіть якщо теоретично злам пароля можливий, практична реалізація такої атаки стає марною через колосальні фінансові витрати. Це створює додатковий рівень захисту, оскільки зловмисники будуть змушені шукати

менш затратні альтернативи.

Економічний бар'єр, створений високими витратами на злам, робить використання «Argon2» надзвичайно ефективним засобом захисту. Витрати в $6 \cdot 10^{15}$ доларів США не тільки перевищують можливості більшості зловмисників, але й роблять саму ідею зламу паролів економічно недоцільною. Це підкреслює важливість використання сучасних алгоритмів гешування, які можуть забезпечити високий рівень безпеки за рахунок створення фінансово непереборних перешкод для атакуючих.

Таким чином, алгоритм «Argon2» не лише забезпечує технічну стійкість до атак повного перебору, але й створює потужний економічний бар'єр, який робить спроби зламу паролів марними з фінансової точки зору.

3.4 Закон Мура і його вплив в сучасному світі

Закон Мура [36], який стверджує, що кількість транзисторів на інтегральній схемі подвоюється приблизно кожні два роки, має значний вплив на всі аспекти обчислювальної техніки, включаючи криптографію. Згідно з цим законом, обчислювальна потужність комп'ютерів зростає експоненційно, що призводить до значного збільшення можливостей для виконання складних обчислень.

У контексті гешування паролів, такий стрімкий розвиток обчислювальної потужності створює серйозні виклики для забезпечення безпеки. З одного боку, потужніші процесори можуть швидше виконувати обчислення, необхідні для атаки повного перебору, що збільшує ризик зламу паролів. З іншого боку, ці ж потужні процесори можна використовувати для розробки та впровадження більш складних і ефективних алгоритмів гешування, таких як «Argon2».

Схема «Argon2», будучи оптимізованою для використання великих обсягів пам'яті та багатоядерних процесорів, здатна ефективно

протистояти зростаючим обчислювальним можливостям зловмисників. Використання великих обсягів пам'яті ускладнює можливість паралельного виконання атак повного перебору, що робить злам паролів значно менш ефективним.

Закон Мура також підкреслює необхідність постійного оновлення і вдосконалення криптографічних методів. У міру зростання обчислювальної потужності, алгоритми гешування повинні адаптуватися, щоб залишатися на крок попереду зловмисників. «Argon2» є прикладом такої схеми гешування, яка враховує сучасні можливості обчислювальної техніки і використовує їх для підвищення рівня безпеки.

Таким чином, дослідження та впровадження алгоритмів гешування, таких як «Argon2», є важливим кроком у забезпеченні інформаційної безпеки в умовах швидкого технологічного прогресу. Необхідно постійно стежити за розвитком обчислювальних технологій і вчасно адаптувати криптографічні методи, щоб ефективно протистояти новим загрозам і забезпечувати надійний захист інформації.

Загалом, закон Мура підкреслює важливість інновацій в галузі криптографії. З огляду на швидкий розвиток обчислювальних потужностей, використання сучасних схем гешування, таких як «Argon2», забезпечує стійкість до багатьох атак і захист даних у цифровому майбутньому.

Висновки до розділу 3

З графіків видно, що збільшення кількості потоків до певного значення призводить до збільшення швидкості перебору. І це логічно, бо це пояснюється тим, що багатопоточність дозволяє процесору ефективніше використовувати свої обчислювальні ресурси. Однак, після досягнення певного максимуму (як показав експеримент №2: це 5-6 потоків), подальше збільшення кількості потоків не призводить до значного збільшення швидкості, а в деяких випадках навіть

спостерігається зниження. Це може бути пов'язано з перевантаженням процесора та збільшенням накладних витрат на управління потоками. Зіставлення графіків для процесорів з різним розміром кешу (9472 КБ і 11648 КБ) показує, що більший кеш забезпечує вищу швидкість перебору при меншій кількості потоків. Це пояснюється тим, що більший кеш дозволяє зберігати більше даних безпосередньо в пам'яті процесора, зменшуючи кількість звернень до оперативної пам'яті, що пришвидшує обчислення. Однак, при збільшенні кількості потоків, перевага більшого кешу стає менш очевидною, оскільки процесор починає відчувати більш накладні витрати на управління потоками. Оптимальна кількість потоків для досягнення максимальної швидкості перебору залежить від архітектури процесора та розміру кешу. Більший кеш сприяє вищій швидкості перебору, особливо при меншій кількості потоків, але його вплив зменшується при збільшенні кількості потоків через зростаючі накладні витрати на управління потоками. Для досягнення максимальної ефективності, важливо знайти баланс між кількістю потоків і розміром кешу, враховуючи архітектурні особливості конкретного процесора. Для процесора i5-10300H частота процесора коливалась незначно, залишаючись в діапазоні 4.18 - 4.24 ГГц при збільшенні кількості потоків від 1 до 8. В свою чергу для процесора AMD Ryzen 5 5500u частота процесора залишалась стабільною на рівні 3.46 ГГц при використанні 3-8 потоків, з незначними коливаннями на менших кількостях потоків (3.20 - 3.46 ГГц). Для процесора i5-10300H максимальна швидкість гешування досяглась при частоті 4.18 ГГц на 6 потоках, що свідчить про те, що процесор ефективно використовує свої ресурси при збільшенні кількості потоків до певного значення, після чого частота стабілізується. Для процесора AMD Ryzen 5 5500u максимальна швидкість гешування досяглась при частоті 3.46 ГГц на 5 потоках, що свідчить про те, що процесор також ефективно використовує свої ресурси, але досягнення стабільної частоти на рівні 3.46 ГГц при більшій кількості потоків вказує на оптимальне завантаження процесора. Якщо порівнювати ці процесори

то i5-10300H має більшу стабільність частоти вищу ніж AMD Ryzen 5 5500u при різних кількостях потоків, що дозволяє йому досягати трохи вищої швидкості гешування при оптимальних умовах. В свою чергу процесор AMD Ryzen 5 5500u демонструє не стабільну продуктивність протягом експерименту, що також дозволяє йому досягати високих швидкостей гешування. Оптимальна продуктивність досягається при певній кількості потоків, після чого подальше збільшення кількості потоків не призводить до значного зростання швидкості гешування, а іноді навіть може знижувати продуктивність через накладні витрати на управління потоками. Ці експерименти практично показують, що швидкість перебору паролів хоч і залежить від таких параметрів процесора, як кількість потоків, розмір загального кешу та частота, проте значних досягнень у швидкості вони не дають. Можливо більш потужніші процесори зробили би перебір всіх паролів при більшій швидкості, однак вона не була б набагато більшої. Ці експерименті доводять стікість схеми гешування «Argon2d» та «Argon2i» до атак повного перебору на комп'ютерах з різними архітектурами.

ВИСНОВКИ

В даній роботі проведено ґрунтовне дослідження стійкості схем гешування «Argon2d» та «Argon2i» до атак повного перебору з урахуванням архітектури процесора. Основною метою було оцінити ефективність та безпеку цих алгоритмів у сучасних умовах, де загроза зламу паролів та несанкціонованого доступу до даних є надзвичайно актуальною.

Після детального огляду сучасних геш-функцій та аналізу структури схеми гешування «Argon2», було проведено серію експериментів, які підтвердили високу стійкість обох варіантів «Argon2» до атак повного перебору. «Argon2d», завдяки своїй залежності від даних, може використовуватись у криптовалютних і серверних застосуваннях, де відсутні атаки по побічним каналам. З іншого боку, «Argon2i», незалежний від даних, демонструє відмінну стійкість до атак по побічних каналах, що робить його ідеальним для використання у системах зберігання паролів і формування ключів на основі паролів.

Результати дослідження показали, що «Argon2» є одним з найнадійніших і найефективніших алгоритмів гешування паролів на сьогоднішній день. Використання цього алгоритму рекомендується в системах, де безпека даних має критичне значення. «Argon2» забезпечує високий рівень захисту завдяки використанню великої кількості пам'яті та можливості налаштування параметрів для оптимальної роботи в різних середовищах.

Рекомендації по застосуванню включають використання «Argon2» для зберігання паролів у базах даних, де захист від атак повного перебору та побічних каналів є ключовим фактором. Варіант «Argon2d» рекомендується для середовищ, де швидкість і ефективність обробки мають вирішальне значення, таких як криптовалюти та серверні додатки. «Argon2i», завдяки своїй стійкості до атак побічних каналів, ідеально

підходить для захисту паролів і ключів у системах з високими вимогами до безпеки.

Результати цієї роботи підкреслюють важливість використання сучасних і надійних методів гешування для забезпечення безпеки інформації. Подальші дослідження в цій області можуть зосередитися на вдосконаленні алгоритмів та їх адаптації до нових загроз, що постійно з'являються у сфері інформаційної безпеки.

ПЕРЕЛІК ПОСИЛАНЬ

1. J. Katz and Y. Lindell, *Introduction to Modern Cryptography*, 2nd ed., CRC Press, 2014. Режим доступу: https://eclass.uniwa.gr/modules/document/file.php/CSCYB105/Reading%20Material/%5BJonathan_Katz%2C_Yehuda_Lindell%5D_Introduction_to_Mo%282nd%29.pdf

2. Режим доступу: https://www.tarsnap.com/scrypt/scrypt.pdf

3. B. Kaliski and A. Rush, ŞPKCS 5: Password-Based Cryptography SpeciĄcationŤ, RFC Editor, RFC 8018, Jan. 2017, p. 40. [Online]. Available: Режим доступу: https://tools.ietf.org/html/rfc8018(visitedon10/28/2018)

4. M. S. Turan, E. B. Barker, W. E. Burr, and L. Chen, ŞSP 800-132. Recommendation for Password-Based Key Derivation: Part 1: Storage ApplicationsŤ, Gaithersburg, MD, United States, Tech. Rep., 2010

5. Режим доступу: https://datatracker.ietf.org/doc/html/rfc8018

6. Blake2 Paper. Режим доступу: https://www.blake2.net/blake2.pdf

7. Режим доступупу: https://www.cs.utexas.edu/~dwu4/courses/sp22/static/projects/Kornerup.pdf

8. Режим доступу: https://www.iacr.org/archive/crypto2003/27290425/27290425.pdf

9. COLIN PERCIVAL. Stronger key derivation via sequential memory-hard functions. 01 2009

10. The Password Hashing Competition (PHC). Режим доступу: https://www.password-hashing.net/

11. Alex Biryukov, Daniel Dinu, Dmitry Khovratovich. Argon2: new generation of memory-hard functions for password hashing and other applications. European Symposium on Security and Privacy, 2016. Режим доступу: https://github.com/P-H-C/phc-winner-argon2

12. Режим доступу: https://www.openwall.com/yescrypt/

13. Режим доступу: https://www.ietf.org/archive/id/draft-irtf-cfrg-argon2-04.txt

14. Режим доступу: https://eprint.iacr.org/2011/499.pdf

15. Режим доступу: https://www.uobabylon.edu.iq/eprints/paper_1_17913_649.pdf

16. Режим доступу: J-P.Aumasson,BLAKE2:simpler,smaller, fastasMD5

17. Режим доступу: https://csrc.nist.gov/pubs/fips/202/final

18. A. Biryukov et al., Argon2: the memory-hard function for password hashing and other applications. Режим доступу: https://password-hashing.net/argon2-specs.pdf

19. A. Biryukov et al., Tradeoff Cryptanalysis of Memory-Hard Functions

20. H. Corrigan-Gibbs et al. Balloon Hashing: Provably Space-Hard Hash Functions with Data-Independent Access Patterns

21. Режим доступу: http://article.gmane.org/gmane.comp.security.phc/3608

22. M. A. Simplicio Jr. et al. Lyra2: Password Hashing Scheme with improved security against time-memory trade-offs

23. K. Pintér. Gambit: A sponge based, memory hard key derivation function

24. Режим доступу: https://www.schneier.com/blog/archives/2012/03/the_security_of_5.html

25. Режим доступу: http://www.postcogito.org/PublicationsInEnglish/improving-diceware-v100-final.pdf

26. M. Dürmuth et al., OMEN: Faster Password Guessing Using an Ordered Markov Enumerator

27. A. Narayanan et al., Fast Dictionary Attacks on Passwords using Time-Space Tradeoff

28. John the Ripper Official Website. Режим доступу: https://www.openwall.com/john/

29. Режим доступу: https://crackstation.net/buy-crackstation-wordlist-password-cracking-dictionary.htm

30. Режим доступу: https://www.lightbluetouchpaper.org/2011/02/09/

measuring-password-re-use-empirically/

31. Режим доступу: https://www.virtualbox.org/manual/ch01.html

32. Режим доступу: https://www.kali.org/docs/virtualization/install-virtualbox-guest-vm/

33. John the Ripper on GitHub. Режим доступу: https://github.com/openwall/john

34. Argon2 Online. Режим доступу: https://argon2.online/

35. Сайт університету Masaryk University.Режим доступу: https://research.redhat.com/blog/article-author/vojtech-polasek/

36. Режим доступу: https://www.alejandrobarros.com/wp-content/uploads/old/4363/Understanding-Moores-Law_Chapter-07.pdf

ДОДАТОК А ТЕКСТИ ПРОГРАМ

ДОДАТОК Б ВЕЛИКІ РИСУНКИ ТА ТАБЛИЦІ