УДК 621.391

АНАЛІЗ, ПОРІВНЯННЯ ТА ОСОБЛИВОСТІ АРХІТЕКТУРИ ФУНКЦІЇ ГЕШУВАННЯ BLAKE ПРОЕКТУ SHA-3

Є.Ю. КУТЯ, І.Д. ГОРБЕНКО

Наводяться порівняння функції гешування BLAKE відносно інших чотирьох фіналістів проекту SHA-3. Наведені результати аналізу архітектури геш-функції BLAKE оцінка результатів швидкодії реалізації алгоритму.

Ключові слова: алгоритм гешування, колізія, швидкодія, стійкість, функція стиску, початковий стан, однонаправленість, ітеративна структура.

ВСТУП

У зв'язку з розвитком інформаційно-телекомунікаційних систем, після останніх досягнень в сфері криптоаналізу геш-функцій, виникла проблема створення нових, більш захищених функцій гешування. Для прийняття нового стандарту функцій гешування було вирішено провести конкурс геш-функйій SHA-3 Competition, ініційований Національним інститутом стандартов і технологій США (NIST). На конкурс було представлено 64 конкурсанта. Після трьох раундів та декількох років аналізу та порівняння до фіналу вийшли 5 алгоритмів. Європейське криптографічне співтовариство проводило параллельний конкурс на базі проекту Ecrypt. Переможцем цього конкурсу стала геш функція BLAKE, створена командою зі Швейцарії. Задача аналізу та порівняння геш-функцій, які пройшли до фіналу конкурсу, та аналіз переваг архітектури та властивостей геш-функції BLAKE залишається актуальною на сьогоднішній день через те, що міжнародний конкурс ще не закінчився, і кожен за кандидатів може бути внесений до національного стандарту фукцій гешування США. Доцільно більш глибоко розглянути архітектуру саме алгоритму BLAKE, як найбільш ймовірного переможця, а значить, алгоритм, який буде реалізований та впроваджений у всі сучасні інформаційні системи.

1. АЛГОРИТМ BLAKE

В цьому розділі розглядається алгоритм BLAKE: історія, архітектура та криптографічний зміст.

1.1. Основа для створення

Функція BLAKE розроблена командою з чотирьох незалежних розробниців зі Швейцарії. Головним розробником вважається Жан-Філіппе Аумассон (Jean-Philippe Aumasson), видатний криптограф та дослідник комп'ютерних наук. Алгоритм BLAKE базується на "ChaCha", версії поточного шифру Salsa20, надійність якого вже була досліджена та описана Аумассоном. Хоча геш принципово відрізняється від двонаправленої природи шифрів, загальними залишаються операції перетворення, до того ж вони передюбачають однакові криптографічні вимоги. Початкова специфікація алгоритму була подана до NIST у 2008 році, але була вдостконалена протягом 2010 року[1].

1.2.Структура Алгоритму

Розробники запропонували чотири офіційні варіанти BLAKE, які відрізняються вихідним розміром геш-значення, згідно до вимог конкурсу. Офіційними вважаються варіанти з вихідним розміром геш-значення 224, 256, 384 та 512 біт (як зазначено у назві BLAKE-224, BLAKE-256 і т.д.). Розміри, що підтримуються, відповідають розмірам SHA-2 з метою підвищення сумісності. Представлені чотири версії також мають відмінності у розмірі оброблюваного повідомлення та максимальній вхідній довжині повідомлення.

BLAKE – це ітеративний алгоритм, що базується на добре-відомій та визнаній ітеративній структурі Hash Iterative Framework (HAIFA). Первісний геш, h^0, загружається з передвизначеного вектору ініціалізації. Розробники BLAKE вирішили використати вектори ініціалізації, що використовуються в SHA-2, з причини її сильних псевдо-випадкових властивостей. Далі вхідне повідомлення розбивається на N еквівалентнорівних блоків (з розширенням до розміру блоку за потребою) та проходить разом із нульовим геш-значенням крізь функцію стиску. Результатом є нове геш-значення. Далі, цей процес повторюється до тих пір поки все повідомлення не буде стиснуте.

Останній результат функції стиску приймається як фінальний геш повідомлення. Ця структура зображена на рис. 1. Головною перевагою структури HAIFA є те, що довге повідомлення може бути гешовано, нібито у потоці, без необхідності мати все повідомлення у доступній пам'яті водночас.

$$h^0 \leftarrow IV$$
$$\textbf{for } i = 0, \ldots, N-1$$
$$\qquad h^{i+1} \leftarrow \textbf{compress}(h^i, m^i, s, \ell^i)$$
$$\textbf{return } h^N$$

Рис. 1. Використання конструкції HAIFA у BLAKE

Функція стиску є головною особливістю структури HAIFA, але її детальна функціональність визначена специфікаціями конкретного алгоритму гешування. Функція стиску BLAKE

визначена у наступній секції. Важливо зазначети, що в функцію передається значення «таймеру»(так його назвали розробники), яке визначає кількість вже оброблених біт. Такий метод допомагає забеспечити те, що ідентичні блоки у ріних частинах вхідного повідомлення будуть мати різні геш-значення. Розробники BLAKE також вирішили включити змінну «сіль» (далі - *salt*) до функції стиску, що забезпечує більшу захищеність від коллізійних атак. Додавання *salt* не є вимогою висунутою NIST, тому це значення встановлюється в нуль, якщо не впроваджено або не визначено користувачем.

1.3. BLAKE-256

В цьому розділі будуть розглянуті деталі версії алгоритму BLAKE-256. Для спощення, три другі версії алгоритму будуть розглядатись тільки в цілях створення акценту на відмінностях між ними.

Функція BLAKE-256 виробляє 256-біт геш-значення, яке розуміється як вісім 32-бітних слів. Вона приймає на вхід повідомлення довжиною $0 \le l \le 264$ біт. Повідомлення доповнюється одиничним бітом, після цього доповнюється послідовністю з нульових бітів та має 64-бітне представлення довжини *l*. Число нульових бітів визначено таким чином, що загальна довжина потоку є поєднанням 512-бітних блоків. Це дозволяє використовувати 512-бітні блокі як вхідні в функцію стіску для обробки.

Функція стиску приймає на вхід чотири параметри: 256 біт поточного значення геш (h^i), 512-бітній блок вхідного повідоплення (m^i), 128-біт *salt* (*s*) та кумулятівній лічільнік 64-бітних повідомлень (l^i). Треба зазначити, що l^i не включає біти розширення (паддінгу), тобто 100-бітне повідомлення буде мати l^i=100 , а не 512. Якщо кінцевий блок складається тільки з бітів розширення, такий випадок вважається особливим та l^N передається як 0, для впевненності відмінності від попереднього значення ітерації.

Початковий геш h0 використаний з SHA-2, як вже зазначалось вище. Він називається Вектором Ініціалізації (IV) та зображений на рис. 2, як вісім шістнадцятирічних слів.

$$IV_0 = 6A09E667 \quad IV_1 = BB67AE85$$
$$IV_2 = 3C6EF372 \quad IV_3 = A54FF53A$$
$$IV_4 = 510E527F \quad IV_5 = 9B05688C$$
$$IV_6 = 1F83D9AB \quad IV_7 = 5BE0CD19$$

Рис. 2. Вектор ініціалізації, який використовується в BLAKE-256

$$\begin{pmatrix} h_0 & h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 & h_7 \\ s_0 \oplus c_0 & s_1 \oplus c_1 & s_2 \oplus c_2 & s_3 \oplus c_3 \\ t_0 \oplus c_4 & t_0 \oplus c_5 & t_1 \oplus c_6 & t_1 \oplus c_7 \end{pmatrix}$$

Рис. 3. Початковий стан функції стиску

Всередині функції стиску, 512-бітний стан *v* зберігається та оброблюється як матриця 4x4 32-бітних слів. Цей стан ініціалізується з поточного геш-значення, значення *salt*, значення таймеру *t* та 256-бітної констнанти *c*, як зображено на рис. 3 (значення l^i що передається всередину функції називається таймером, який відрізняється від значення довжини всередині функції стиску). Значення константи *c*, прямо взяті з шістнадцятирічного представлення π, обранного для такої ітераційної структури (рис. 4).

Після ініціалізації матриці стану, вона ітеративно проходить 14 раундів обробки. Розробники геш-функції BLAKE обрали модель захисту, яка використовує невелику кількість складних раундів, в той час як велика кількість дослідників віддають перевагу великій кількості менш складних раундів.

$$c_0 = 243F6A88 \quad c_1 = 85A308D3$$
$$c_2 = 13198A2E \quad c_3 = 03707344$$
$$c_4 = A4093822 \quad c_5 = 299F31D0$$
$$c_6 = 082EFA98 \quad c_7 = EC4E6C89$$
$$c_8 = 452821E6 \quad c_9 = 38D01377$$
$$c_{10} = BE5466CF \quad c_{11} = 34E90C6C$$
$$c_{12} = C0AC29B7 \quad c_{13} = C97C50DD$$
$$c_{14} = 3F84D5B5 \quad c_{15} = B5470917$$

Рис. 4. Константи *c*, алгоритму BLAKE, отримані з π

Обидві парадигми мають своїх прихильників, але чи є одна з парадигм більше захищена ніж друга, доведено не було. («Захищена» в цьому контексті означає складність інвертування, тобто визначення вхідних даних, використовуючи вихідні).

Кожний раунд складається з восьми перетворень стану, які обозначають як G0-G7. Ці перетворення відповідаються за змішування даних (змінення даних) та дифузію бітів (розсіювання існуючих даних) по всім даним.

Кожний з раундів перетворює тільки 4 з 16 слів стану, які обозначаються як *a*, *b*, *c* та *d*. Перетворення (включаючи додавання, інверсії, виключення або XOR-операції) представлені на рис. 5.

$$\begin{aligned} a &\leftarrow a + b + (m_{\sigma_r(2i)} \oplus c_{\sigma_r(2i+1)}) \\ d &\leftarrow (d \oplus a) \ggg 16 \\ c &\leftarrow c + d \\ b &\leftarrow (b \oplus c) \ggg 12 \\ a &\leftarrow a + b + (m_{\sigma_r(2i+1)} \oplus c_{\sigma_r(2i)}) \\ d &\leftarrow (d \oplus a) \ggg 8 \\ c &\leftarrow c + d \\ b &\leftarrow (b \oplus c) \ggg 7 \end{aligned}$$

Рис. 5. Головне перетворення G

Головна функція перетворення G відображена на рис. 6 у вигляді візуальної схеми перетворень. Індекс σ_r відповідає одному з десяти перетворень чисел з 0 до 15, індексовані номером раунду *r* за модулем 10. Ця функція має вирішуюче значення

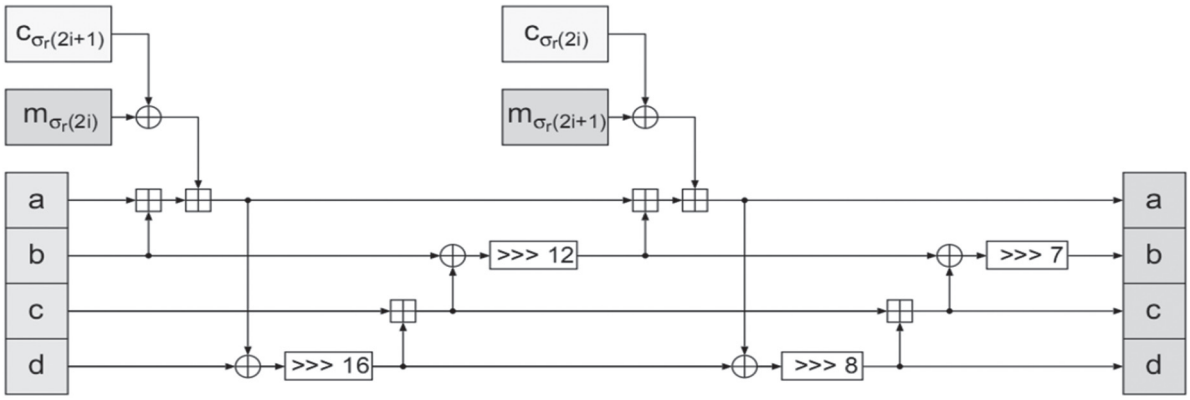

Рис. 6. Візуальне представлення функції G

для відповідного розсіювіання вхідних даних. Таблиця перестановок цього перетворення представлена на рис. 7.

Стани слова, по якому кожне перетворення функції G діє, були спеціально зформовані для підвищення ефективності. Перші чотири оперують над незалежними стовбцями та можуть виповнюватись параллельно. Останні чотирі оперують над незалежними діагональними значеннями, які також можуть бути виконані у паралельному режимі.

Таким чином раунд може бути зображений у вигляді двох великих операцій, ніж розглядати це як вісім послідовних операцій. Першаоперація названа як діагональний крок, а друга – вертикальний. Це істотно зменшує час обчислення одного раунду на 75% у аппаратній та деяких програмних реалізаціях. Таке розпараллелювання представлене на рис. 8, зображаючи, які слова обробляються кожною операцією.

σ_0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
σ_1	14	10	4	8	9	15	13	6	1	12	0	2	11	7	5	3
σ_2	11	8	12	0	5	2	15	13	10	14	3	6	7	1	9	4
σ_3	7	9	3	1	13	12	11	14	2	6	5	10	4	0	15	8
σ_4	9	0	5	7	2	4	10	15	14	1	11	12	6	8	3	13
σ_5	2	12	6	10	0	11	8	3	4	13	7	5	15	14	1	9
σ_6	12	5	1	15	14	13	4	10	0	7	6	3	9	2	8	11
σ_7	13	11	7	14	12	1	3	9	5	0	15	4	8	6	2	10
σ_8	6	15	14	9	11	3	0	8	12	2	13	7	1	4	10	5
σ_9	10	2	8	4	7	6	1	5	15	11	9	14	3	12	13	0

Рис. 7. Десять перетворень визначені для BLAKE

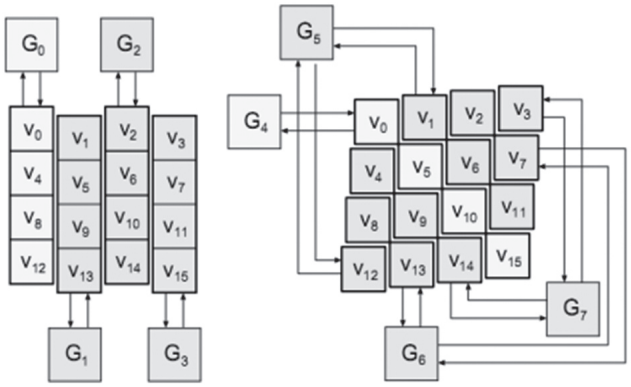

Рис. 8. Розпараллелені вертикальний та діагональний кроки

Після чотирнадцяти раундів перетворення функції G функція стиску виконує останній фінальний крок. Вона генерує нові 256-біт геш-значення, h^{i+1}, як великий XOR попереднього геш-значення h^i, значення *salt s* та 512-біт матриці стану *v* (рис. 9). Якщо кінцевий блок розширенного вхідного потоку був оброблений, тоді це являється результатом кінцевого виходу геш-функції BLAKE.

$$
\begin{aligned}
h'_0 &\leftarrow h_0 \oplus s_0 \oplus v_0 \oplus v_8 \\
h'_1 &\leftarrow h_1 \oplus s_1 \oplus v_1 \oplus v_9 \\
h'_2 &\leftarrow h_2 \oplus s_2 \oplus v_2 \oplus v_{10} \\
h'_3 &\leftarrow h_3 \oplus s_3 \oplus v_3 \oplus v_{11} \\
h'_4 &\leftarrow h_4 \oplus s_0 \oplus v_4 \oplus v_{12} \\
h'_5 &\leftarrow h_5 \oplus s_1 \oplus v_5 \oplus v_{13} \\
h'_6 &\leftarrow h_6 \oplus s_2 \oplus v_6 \oplus v_{14} \\
h'_7 &\leftarrow h_7 \oplus s_3 \oplus v_7 \oplus v_{15}
\end{aligned}
$$

Рис. 9. Фінальний крок стиску

1.4. Відмінності варіантів алгоритму

Другі три варіанти геш-функції дуже схожі на BLAKE-256. 512-бітна версія подвоює бітовий розмір більшості змінних: всі серединні геш-значення, матриця стиску, максимальний розмір повідомлення, сіль та значення лічильника. Вектор ініціалізації обраний з SHA-512, а константа c просто розширена більшою кількістю цифр з ϖ. Таблиця перестановок σ залишається без змін.

Єдиною нетрівіальною різниця в функції G: кількість bit-rotation визначені не просто подвоєнням попередньої кількості. Також в 512-бітній версії рекомендовано збідьшити кількість раундів з 14 до 16. Але фактично, цей параметр є настраюванним параметром користувача, як один з параметрів компромісу швидкості та захисту.

224- та 384-бітні версії використовують 256-бітну та 512-бітну версії алгоритму, потім урізають відповідно вихідне геш-значення до потрібного. Швидкодія цих реалізацій алгоритму ріна до тих що вже були розглянуті. Єдиною істотною різницею є вектори ініціалізації (використовуються SHA-224 та SHA-384 відповідно) та та виключення одиничних бітів («1»), під час розширення повідомлення до розміру вхідного блоку.

1.5. Надійність

Архітектура криптографічних геш-функцій передбачає балансування між рівнем захищеності

та швидкодією (а в аппаратних реалізаціях також затратами на електороенергію). В геш-функції BLAKE параметром, який дозволяє регулювати баланс захищеності та швидкодії виступає число раундів. З неофіційною метою значного підвищення рівня захисту відносно SHA-2, початкова кількість раундів SHA-2 була 10 раундів для 256-бітної версії та 14 раундів для 512-бітної версії. А вже в грудні 2010 кількість раундів була підвищена до 14 та 16 раундів відповідно, з метою забезпечення дуже високого рівня захищеності.

Під час розробки алгоритму, його автори приділили велику увагу питанню розсіювання даних, або лавинному ефекту. «Повне розсіювання» означає, що змінення одного біту вхідних даних (повідомлення, «солі» *salt* або вектору ініціалізації) може справити вплив на кожен біт виходу, причому досягається він за два раунди перетворень. Фунція стиску та функція G були спроектовані з метою мінімізації ймовірності локальних колізій, які з'являються коли два різних повідомлення призводять до однакового внутрішнього стану після деякого числа раундів. Крім того, значення *salt* (яке зазвичай тримається в секреті користувачем) передається до кожної функції стиску (а не тільки під час ініціалізації або фіналізації) для запобігання визначення значення *salt* із відомих пар повідомлення – геш-значення.

Розробники BLAKE також представили чотири зменшенні версії , тобто навмистно ослаблені версії для криптоаналітичних цілей.

Такими версіями виступають:

- BLOKE: всі *y* перетворення є проті від 0-15, по черзі
- FLAKE: функція стиску не містить *salt* або поточне геш-значення в її фінальному XOR-перетворенні
- BLAZE: функція G використовує нульові значення замість констант *c* отриманих з π
- BRAKE: дуже слабкий, містить всі три попередні слабкості

Методи криптоаналізу слабких версій перевіряються і на повній версії алгоритму. На грудень 2010 найкращою атакою, аку прийняли розробники геш-функції, була атака першого прообразу на 2.5 раундах, на вповній версії BLAKE із зменшеною кількістю раундів[2].

2. РЕАЛІЗАЦІЯ АЛГОРИТМУ

Так як внутрішня структура алгоритму BLAKE була успішно викладена та проаналізована, наступний розділ описує деякі методи практичної реалізації алгоритму. Цей розділ зфокусований на швидкодії, та не бере до уваги захищеність чи аппаратні вимоги.

2.1. ОФіційна реалізація BLAKE

Розробники функції гешування BLAKE запропонували декілька своїх реалізацій виконаних мовою програмування C. Разом із версіями простими для розуміння, які повністю відповідають опису алгоритму автори пропонують оптимізовані версії зі специфічними розмірами ключів та архітектурами вцілому. Офіційнийвеб-ресурс також містить посилання на одобрені та затверджені імплементації мовами Perl, PHP, Javascript та інші.

Коротко проаналізуємо спрощену 512-бітну версію (lightweight version). Так як компактна та призначена для одного варіанту вихідного розміру, вона вимагає мінімальних ресурсів програмного та аппаратного забеспечення та може працювати на багатьох платформах. Також в цю версію включено один Вбудований Тест (BIST), який гешує два вручну закодованих повідомлення та порівнює результат з вже відомими правильними геш-значеннями.

Геш-функція визивається як:

blake512_hash(digest, data, length), де

digest – це вказівник на 512-бітний вихідний буфер,

data – вказівник на вхідне повідомлення,

length – довжина вхідого повідомлення (у байтах).

Функція створює матрицю стану 4x4 та викликає три підпрограми : одна для ініціалізації станів, друга – для головного перетворення та третя для створення фінального стану повідомлення. Це відповідає основним крокам представлених в розділі з описом BLAKE-256.

Фунція blake512_update відповідає за розширення вхідного повідомлення до розміру оброблюваного блоку, розбиває повідомлення на 1024-бітні блоки, та ітеративно виконує функцію стиску. Функція стиску називається blake512_compress, вона має 16 раундів, кожний з яких містить вісім перетворень G, які використовують арифметичні макроси. Конкретно дана реалізація виконує всі вісім перетворень послідовно, вона не оптимізована для використання техніки розпаралелювання, яка була розглянута раніше.

Результат записується у digest та звичайно представляється, як 128 символьна строка у шістнадцятиричній системі. За замовчуванням, программа гешує повідомлення довжиною у 144 байти (1152 біта), але ця величина може бути змінена для тестування з різними довжинами. У код программи додається таймер для вимірювання швидкості гешування та фіксування результатів під час тестування.

Попередні результати тестування з використанням 64-бітного процесору Intel Core Duo представлені в табл. 1. Кількість тактів процесору на кожний вхідний байт для різних кількостей раундів.

Таблиця 1

Швидкодія / Розмір повідомлення

Назва геш-функції	Кількість раундів	Байти повідомленя			
		10	100	1000	10000
BLAKE-256	10	~1600	36.4	18.4	16.7
BLAKE-256	8	~1600	32.2	15.4	13.8
BLAKE-256	5	~1600	26.9	10.9	9.6
BLAKE-512	14	~1900	33.7	13.8	12.3
BLAKE-512	10	~1900	29.9	11.6	9.3
BLAKE-512	7	~1900	26.8	8.5	7.2

Результати ECRYPT тестування BLAKE-512 з використанням 64-бітного процесору Intel Core Duo представлені в табл. 2. Середні значення геш тактів на байт, для різних вхідних довжин.

Таблиця 2

Середні значення геш тактів на байт

Байти повідомлення	Q1	Median	Q2
8[1]	159.00	316.50	316.50
64[1]	19.69	38.81	39.00
576	9.12	9.19	9.19
1536	8.61	8.62	8.63
4096	8.02	8.03	8.05
довге[2]	7.65	7.69	7.74

[1]результати значно відрізняються
[2] приблизний асимптотичний ліміт

2.2. Комп'ютерна швидкодія

Специфікація BLAKE містить деякі тести швидкодії програмної реалізації на мікроконтроллерах та побутових (споживчих) процесорах CPU. Було відмічено, що 64-бітні версії значно швидчі за 32-бітні, кількість тактів на байт повідомлення зменшується, якщо росте розмір входу (так як розрахунок обислювальних витрат розподіляється на більшу кількість оброблених блоків), в той час як результати для маленьких довжин повідомлення (наприклад 10 байтів) є приблизними, з причини малого числа оброблених байтів які значно впливають на відношення такт/байт.

Відомий проект ECRYPT провів незалежне порівняння п'яти SHA-3 фіналістів на великій кількості комп'ютерних систем, та з різними довжинами повідомлень. Ці результати публічно представлені та доступні на веб-ресурсі проекту[3]. Таблиця 2 узагальнює ці результати для BLAKE-512 виконаному на 64-бітному процессорі Intel Core 2 Duo 2.4 GHz, для порівняння з результатами представленими у Таблиці 1. Результати також представлені на рис. 10, ілюструючи підвищення швидкодії (зменшення відношення тактів/байт) при збільшені розміру повідомлення.

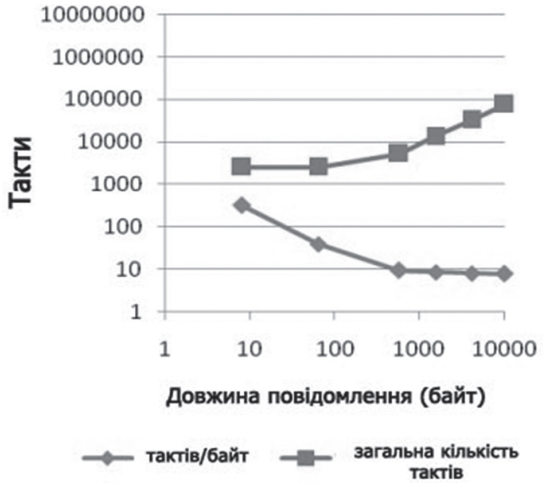

Рис. 10. Результати тестів ECRYPT для BLAKE-512 (Core 2 Duo)

Проект ECRYPT слугує як інформативне джерело порівняння геш-функцій фіналістів SHA-3: BLAKE, JH, Skein, Keccak та Grostl. За результатами даного проекту BLAKE є одним з найшвидших алгоритмів (можливо з причини невеликої кількості раундів), навіть швидший за Skein, та був обраний, як неофіційний переможець у власному конкурсі ECRYPT.

Кандидати SHA-3 також були проаналізовані на наявність «особливих» архітектур. Дослідники з Лабораторії Криптографічних алгоритмів протестували швидкодію всіх кандидатів другого раунду конкурсу SHA-3 використовуючи два спеціалізованих процессора: Cell Broadband Engine створений Sony, Toshiba та IBM, а також графічний процесор (GPU), розроблений NVIDIA. Обидва процесора є високоефективними, багатоядерними процесорами, які використовують модель Single Instruction Multiple Data (SIMD) та Single Instruction Multiple Threads (SIMT)[4]. GPU звичайно використовується з метою обробки важкого графічного контенту, в той час як Cell процессор використовується, наприклад, в самих передових PlayStation 3 ігрових приставках.

Ці тести використовують також різні типи метрики для вимірювання швидкодії. Кандидати були поділені на групи, AES натхненні та не AES натхненні (до яких і відноситься BLAKE). Тести останньої групи були сфокусовані на їх функціях стиску, більше ніж на весь алгоритм. Не враховуються час на налаштування системи, час фіналізації, або час копіювання даних, окрім копіювання ланцюга значень (таких як h^i для BLAKE). Ці досліди також сфокусовані на кількості разів, коли функція буде визвана, замість тактів. Такі дослідження важливі, тому що переможець SHA-3 буде реалізований на великій кількості сучасних спеціалізованих платформ.

ВИСНОВКИ

В даній статті була розглянута та проаналізована функція гешування BLAKE, один з фіналістів конкурсу на стандарт функції гешування SHA-3. Була розглянута історія створення, можливе застосування, архітектура та реалізація алгоритму. Його криптографічна структура та алгоритм були детально досліджені, зазначаючи на фактах які доводять захищеність та швидкодію. На практиці було реалізовано програмне забезпечення використовуючи за основу реалізацію представлену авторами функції, часткову реалізацію методів.

Майбутне алгоритму BLAKE та його потенційне світове впровадження буде відомо вже в поточному році, коли NIST оголосить, яка геш-функція стане новим стандартом SHA-3. Ці дослідження мають цінність для України, по-перше, тому що функція, яка війде до міжнародного стандарту функцій гешування буде впроваджена майже в усі сучасні інформаційні системи, по-друге, за досвідом конкурсу на стандарт симметричного шифру, навіть якщо BLAKE не буде

прийнятий, як стандарт, він буде реалізований в багатьох системах та пристроях.

Література

[1] J.-P. Aumasson, L. Henzen, W. Meier, and R. Phan, "SHA-3 proposal BLAKE," December 2010.

[2] L. Ji and X. Liangyu, "Attacks on round-reduced BLAKE," 2009.

[3] D. J. Bernstein and T. Lange, "List of SHA-3 candidates measured, indexed by machine," 2011, http://bench.cr.yp.to/results-sha3.html.

[4] J. W. Bos and D. Stefan, "Performance analysis of the SHA-3 candidates on exotic multi-core architectures," 2010.

[5] S. Neves, "ChaCha implementation," 2009, URL http://eden.dei.uc.pt/sneves/chacha/chacha.html.

[6] M. Knezevic, K. Kobayashi, J. Ikegami, S. Matsuo, A. Satoh, U. Kocabas, J. Fan, T. Katashita, T. Sugawara, K. Sakiyama, I. Verbauwhede, K. Ohta, N. Homma, and T. Aoki, "Fair and consistent hardware evaluation of fourteen round two SHA-3 candidates," April 2011.

Надійшла до редакції 13.03.2012

Кутя Євген Юрійович, магістрант кафедри БІТ ХНУРЕ. Область наукових інтересів: аналіз асиметричних криптосистем і функцій гешування, генератори ПВП, асиметричні криптопримітиви в групі точок еліптичних кривих.

Горбенко Іван Дмитрович, фото та відомості про автора див. на с. 190.

УДК 621.3.06

Анализ, сравнение и особенности архитектуры функции хеширования BLAKE проекта SHA-3 / Е.Ю. Кутя, И.Д. Горбенко // Прикладная радиоэлектроника: науч.-техн. журнал. – 2012. – Том 11. № 2. – С. 228–233.

Проводится анализ перспективной функции хеширования, претендента к принятию в качестве американского стандарта хеширования. Приводятся результаты оценки быстродействия алгоритма в зависимости от увеличения размера входного сообщения, а также возможности реализации параллельного вычисления значения хеш.

Ключевые слова: алгоритм хеширования, коллизия, быстродействие, стойкость, функция сжатия, начальное состояние, однонаправленность, итеративная структура.

Табл. 2. Ил. 10. Библиогр.: 6 наим.

UDC 621.3.06

Analysis, comparison and structure features of SHA-3 project's hash function BLAKE / E.Yu. Kutia, I.D. Gorbenko // Applied Radio Electronics: Sci. Journ. – 2012. Vol. 11. № 2. – P. 228–233.

The analysis of a perspective hash function is made which is an applicant for the acceptance as the American hash function standard. Algorithm performance evaluation results depending on input message size and possibilities of implementing parallel hash value computing are presented in this paper.

Keywords: hashing algorithm, collision, performance, attack resistance, compress function, initial state, inconvertibility, iterative structure.

Tab. 2. Fig. 5. Ref.: 5 items.