УДК 004.056
DOI https://doi.org/10.17721/ISTS.2020.4.48-52

О. І. Махович, orcid.org/0000-0002-4684-9881,
makhovych.oleksandr@univ.kiev.ua
Р. В. Миколайчук, orcid.org/0000-0001-5349-4487,
mykroman@ukr.net
Київський національний університет імені Тараса Шевченка, Київ, Україна
В. Г. Миколайчук, orcid.org/0000-0002-2532-5771,
mukwira@ukr.net
Державний університет телекомунікацій, Київ, Україна

РЕКОМЕНДАЦІЇ ЩОДО ВИБОРУ СПОСОБУ БЕЗПЕЧНОГО ЗБЕРІГАННЯ ПАРОЛІВ

Використання паролів залишається найпоширенішим способом автентифікації користувачів для різного роду інформаційних систем. У зв'язку із цим виникає задача забезпечення безпеки зберігання інформації, що стосується даних автентифікації користувачів, та її захисту від несанкціонованого доступу. На практиці набули широкого розповсюдження різноманітні алгоритми безпечного зберігання паролів. Взаємосуперечні вимоги до таких алгоритмів безпечного зберігання паролів, які з одного боку мають бути достатньо складними для протидії різноманітним атакам, а з іншого – простими для забезпечення швидкодії інформаційної системи – перебору, особливо враховуючи те, що обчислювальна потужність центральних і графічних процесорів постійно зростає. Тому виникає потреба мати можливість змінювати складність обчислення хеш-коду, а отже й обсяг обчислень і час так, щоб значно ускладнити здійснення атаки, але не спричиняти дискомфорту кінцевому користувачу через затримку перевірки достовірності пароля. Серед відомих способів безпечного зберігання паролів розглянуто шифрування паролів, використання хеш-функції у класичному варіанті, а також із додаванням солі та застосуванням ітерацій для обчислення хеш-коду. У роботі проведено порівняльний аналіз наведених способів, установлено їхні переваги й недоліки, окреслено доцільні галузі застосування кожного способу, розроблено відповідні рекомендації. Для проведення обчислювального експерименту використовувалися засоби платформи Microsoft .NET Core 3.1, що дало змогу встановити часові показники роботи алгоритму отримання хеш-коду залежно від установлених параметрів алгоритму. Отримані за результатами експерименту дані можуть бути використані для вибору способу безпечного зберігання паролів.

Ключові слова: *захист інформації; автентифікація користувача; шифрування; алгоритми обчислення хеш-коду; обчислювальний експеримент.*

1. ВСТУП

Використання паролів залишається найпоширенішим способом автентифікації користувачів. Для забезпечення безпеки зберігання інформації та її захисту від несанкціонованого доступу набули широкого розповсюдження алгоритми безпечного зберігання паролів [1–3]. Водночас для уникнення можливості компрометації інформаційної системи під впливом зовнішніх загроз, необхідно підвищувати складність зазначеного алгоритму. З іншого боку, надмірне ускладнення процедури автентифікації призводить до зниження швидкості роботи системи. У сукупності наведені взаємосуперечні вимоги до алгоритмів безпечного зберігання паролів складають актуальну наукову проблему, одному з елементів якої, а саме дослідженню способів безпечного зберігання паролів, присвячена ця стаття. Метою статті є обґрунтування рекомендацій щодо вибору способу безпечного зберігання паролів.

2. СПОСОБИ ЗБЕРІГАННЯ ПАРОЛІВ

Як зазначено раніше, важливим аспектом застосування парольної автентифікації є безпечне зберігання паролів. Для забезпечення зазначеної функції інформаційної системи використовують способи, що будуть розглянуті далі.

2.1. ШИФРУВАННЯ ПАРОЛІВ

Шифрування є звичним засобом забезпечення конфіденційності інформації. Цей процес передбачає застосування двох процесів:

зашифровування та розшифровування даних із використанням секретного ключа. Шифрування може бути застосоване для зберігання паролів, але при цьому можуть виникнути певні труднощі й задачі:

- організація керування ключами;
- забезпечення надійного зберігання секретного ключа;
- визначення обмеження доступу клієнтського додатку до секретного ключа для шифрування пароля перед його відправкою на сервер;
- організація безпечної передачі ключів.

***Таблиця* 1**

Хеш-функції, доступні в .NET Core

Алгоритм	Розмір хешу (байт)	Опис
MD5 *(**M**essage **D**igest **A**lgorithm)*	16	Не рекомендується для нових проєктів, оскільки нестійкий до колізій, але швидкий у роботі
SHA1 (***S**ecure **H**ashing **A**lgorithm*)	20	Використання в інтернеті не рекомендується з 2011 р.
SHA256 SHA384 SHA512 *(**S**ecure **H**ashing **A**lgorithm)*	32 48 64	Алгоритми із сім'ї "Безпечний алгоритм хешування другого покоління" із різними розмірами хеш-значень

Відсутність необхідності використовувати відкритий пароль та складність управління ключами шифрування ставить під сумнів оптимальність застосування шифрування для організації безпечного зберігання паролів та може породити загрози інформаційній безпеці. У випадку компрометації секретного ключа та таблиці паролів, зловмисник може отримати повний доступ до всієї інформації.

2.2. ХЕШУВАННЯ ПАРОЛІВ

Криптографічна хеш-функція H є однобічним перетворенням бітового рядка M довільної довжини на бітовий рядок (блок) фіксованої довжини h [1]. Вона має такі властивості [2]:

1) хеш-код повинен легко обчислюватися для довільного вхідного повідомлення;

2) результат роботи хеш-функції має залежати від усіх двійкових символів вихідного повідомлення, а також від їхнього взаємного розташування (незначна зміна вхідного повідомлення має призводити до тотальної зміни значення хеш-функції – так званий "лавинний ефект"). Значення хеш-коду не повинно давати витоку інформації навіть про окремі біти аргументу;

3) хеш-функція має бути стійкою до відновлення прообразу (односторонність) – відсутність практичної можливості за прийнятний час відтворити повідомлення M за його хеш-кодом h так, щоб $H(M) = h$;

4) хеш-функція має бути стійкою до виникнення колізій:

- *стійкість до колізій першого роду*: для заданого повідомлення M має бути неможливо за допомогою обчислень за прийнятний час підібрати інше повідомлення N, для якого $H(N) = H(M)$;
- *стійкість до колізій другого роду*: має бути неможливо за допомогою обчислень підібрати пару повідомлень (M, N) і $M \neq N$ таких, що мають однаковий хеш $H(M) = H(N)$.

На платформі Microsoft .NET Core є доступними для використання [3] реалізації декількох алгоритмів хешування (табл. 1). Обираючи алгоритм хешування, потрібно враховувати два важливих фактори:

1) стійкість до колізій;

2) стійкість до знаходження прообразу.

Використання хеш-функцій є поширеним для підтримки цілісності даних, проте їхнє застосування для зберігання паролів може породжувати певні загрози інформаційній безпеці.

Оскільки хеш-функція для однакових вхідних даних повертає ідентичний хеш-код, то зловмисник може використати словник найбільш уживаніших паролів для атаки або генерувати їх програмно. При цьому для кожного ймовірного пароля послідовно обчислюється хеш-код і порівнюється з атакованим хеш-кодом.

Такий підхід вимагає значних обчислювальних ресурсів, тому інший тип атаки передбачає використання величезних масивів наперед обчислених хеш-кодів (так званих *райдужних таблиць*) для різноманітних паролів. Пошук аргументу для хеш-функції в цьому випадку може відбуватися надзвичайно швидко [4].

2.3. ДОДАВАННЯ "СОЛІ"

Атаки на хеш можливі лише тому, що обчислення хеш-кодів здійснюється кожного разу ідентичним способом. Якщо для кожного пароля спосіб обчислення хешу модифікувати випадковим чином, то можна уникнути вразливості до атак із використанням райдужних таблиць. У цьому випадку доцільно додавати до пароля додаткову ентропію у вигляді криптографічно стійкої послідовності випадкових значень, яку називають *сіллю* [5].

Сіль додається до пароля перед обчисленням хеш-коду. Тому навіть якщо двоє різних користувачів використають однаковий пароль, хеш-коди у

них будуть різні. У випадку, коли сіль для кожного пароля генерується окремо й ніколи не використовується повторно, зловмисник не зможе наперед обчислити райдужні таблиці та скористатися ними для атаки. Оскільки сіль потрібно використовувати при обчисленні хешу для введеного пароля користувача, то її зберігають, як правило, у базі даних поряд з хеш-кодом, або як частину хеш-коду.

2.4. СТАНДАРТ PBKDF2

Використання солі при обчисленні хеш-коду пароля дозволяє уникнути вразливості з використанням райдужних таблиць. Але загроза прямого перебору залишається, особливо враховуючи те, що обчислювальна потужність центральних і графічних процесорів постійно зростає. Тому виникає потреба мати можливість змінювати складність обчислення хеш-коду, а отже й обсяг обчислень та час так, щоб значно ускладнити здійснення атаки, але не спричиняти дискомфорту кінцевому користувачу через затримку перевірки достовірності пароля.

Таке ускладнення обчислень досягається використанням спеціальних повільних алгоритмів обчислення хеш-коду, наприклад, PBKDF2 (Password-Based Key Derivation Function) [6]. PBKDF2 генерує похідний ключ, базуючись на основному ключі та додаткових параметрах. У ролі головного ключа виступає пароль, а додатковими параметрами є сіль та кількість ітерацій обчислення хеш-коду функцією, що лежить в основі алгоритму:

$$DK = PBKDF2 (P, S, c, dkLen),$$

де P – пароль, S – "сіль", c – кількість ітерацій, dkLen – довжина похідного ключа, DK – похідний ключ.

Для дослідження залежності часу, необхідного для обчислення хеш-коду пароля, від кількості ітерацій проведено серію обчислювальних експериментів.

Для цього створено консольний додаток на платформі Microsoft .NET Core 3.1, у якому для кожного базового алгоритму хешування та кожної кількості ітерацій проводилося 10 послідовних обчислень зі знаходження хеш-коду пароля. При цьому фіксувався час у мілісекундах, необхідний для виконання обчислень. Для фіксування часу обчислень використовувалися функції Start(), Stop() і властивість ElapsedMilliseconds класу Stopwatch із простору імен System.Diagnostics. Як результат обирали середнє арифметичне отриманих значень затраченого часу. Експерименти повторювали для кількості ітерацій від 10000 до 1280000, де кожне наступне значення обиралося вдвічі більше за попереднє.

Для генерації солі використовувався клас RNGCryptoServiceProvider із простору імен System.Security.Cryptography. Довжина солі становила 32 байти.

Обчислення хеш-коду за алгоритмом PBKDF2 виконувалося за допомогою функції GetBytes() класу Rfc2898DeriveBytes простору імен System.Security.Cryptography.

Для базових алгоритмів SHA1, SHA256, SHA384 та SHA512 були обрані ефективні довжини згенерованого хеш-коду 20, 32, 48 та 64 байти відповідно (табл. 2).

***Таблиця* 2**

Результати обчислювальних експериментів

Базова функція хешування	Довжина хеш-коду (байт)	Кількість ітерацій	Середнє значення витраченого часу (мс)
SHA1	20	1,00E+04	22,20
		2,00E+04	25,40
		4,00E+04	38,40
		8,00E+04	77,50
		1,60E+05	132,00
		3,20E+05	244,70
		6,40E+05	473,40
		1,28E+06	793,80
SHA256	32	1,00E+04	27,10
		2,00E+04	27,80
		4,00E+04	60,40
		8,00E+04	144,50
		1,60E+05	246,10
		3,20E+05	504,90
		6,40E+05	854,30
		1,28E+06	1262,80
SHA384	48	1,00E+04	35,50
		2,00E+04	42,00
		4,00E+04	76,90
		8,00E+04	155,20
		1,60E+05	276,60
		3,20E+05	544,10
		6,40E+05	933,50
		1,28E+06	1497,30
SHA512	64	1,00E+04	28,70
		2,00E+04	29,90
		4,00E+04	80,90
		8,00E+04	157,70
		1,60E+05	289,70
		3,20E+05	556,50
		6,40E+05	887,70
		1,28E+06	1480,70

Для кожного із зазначених алгоритмів за допомогою комп'ютерного моделювання визначено числові характеристики часу роботи алгоритму, залежно від установленої кількості ітерацій обчислення хеш-коду.

Обираючи кількість ітерацій для обчислення хеш-коду пароля за алгоритмом PBKDF2, потрібно враховувати, що при автентифікації користувачів буде зростати час відгуку сервера, що може спричинити негативне враження на користувача. Справді, для кожного випадку автентифікації потрібно буде обчислювати хеш-код для введеного пароля і порівнювати його з хеш-кодом, що зберігається у базі даних. Тому для інформаційних систем із великою інтенсивністю взаємодій із користувачами потрібно дотримуватися балансу між надійністю зберігання паролів і продуктивністю всієї системи (рисунок).

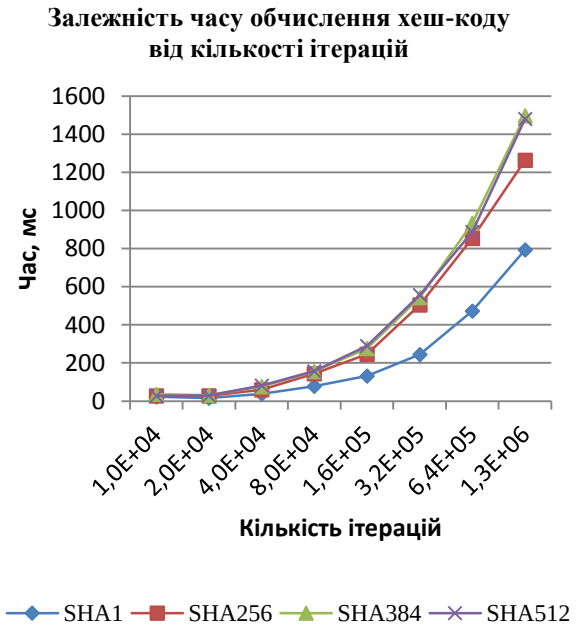

Рисунок. Графік залежності часу, необхідного для знаходження хеш-коду за алгоритмом PBKDF2, від кількості ітерацій

3. РЕКОМЕНДАЦІЇ ЩОДО ВИБОРУ СПОСОБІВ ЗБЕРІГАННЯ ПАРОЛІВ

На основі наведених у попередньому розділі статті результатів, можливо запропонувати такі рекомендації щодо вибору способу безпечного зберігання паролів:

1. Шифрування паролів слід використовувати лише в інформаційних системах, для яких критично важливим є забезпечення можливості відновлення оригінального пароля.

2. Використання "класичної" хеш-функції є доцільним в успадкованих інформаційних системах для забезпечення сумісності, або у системах, де критичною є швидкість обчислень. Причому рекомендується застосовувати алгоритми SHA256, SHA384, SHA512.

3. Розмір солі для хешування має бути не менше ефективної довжини згенерованого хеш-коду, для кожного випадку вона має генеруватися окремо у вигляді криптографічно стійкої послідовності випадкових значень, одна й та ж сіль ніколи не повинна використовуватися повторно.

4. Для найбільш критичних до стійкості від зовнішніх атак додатків доцільно обирати алгоритм PBKDF2 хешування паролів, як досить гнучкий у налаштуваннях і порівняно стійкий до різного роду атак на хеш-код, у цьому разі кількість ітерацій має забезпечувати достатню обчислювальну складність для уникнення атаки прямим перебором, але разом із тим не перевантажувати власний сервер.

4. ВИСНОВКИ

Таким чином у результаті дослідження проведено порівняльний аналіз відомих способів безпечного зберігання паролів, установлено їхні переваги й недоліки, експериментально визначено часові показники застосування алгоритму стандарту PBKDF2, окреслено доцільні галузі застосування кожного способу.

Це дало змогу запропонувати рекомендації, які можливо використовувати для вибору способу зберігання паролів, залежно від типу інформаційної системи та важливості даних, що в ній зберігаються.

Отримані за результатами обчислювального експерименту дані можуть бути використані для вибору способу безпечного зберігання паролів.

Подальші дослідження доцільно проводити у напрямку створення моделей оцінювання ефективності способів зберігання паролів в інформаційних системах.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

[1] Н. Смарт. *Криптография*, Техносфера, Москва, 2005, 528 с.

[2] Stephen Haunts, *Applied Cryptography in .NET and Azure Key Vault. A Practical Guide to Encryption in .NET and .NET Core*, Apress, Berkeley, CA, 2019, 228 p.

[3] *Microsoft documentation*, [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.hashalgorithmname?view=netcore-3.1

[4] *Salted Password Hashing – Doing it Right*, [Online]. Available: https://crackstation.net/hashing-security.htm

[5] Yasser M. Alginahi, Muhammad Nomani Kabir, *Authentication Technologies for Cloud Computing, IoT and Big Data*, The Institution of Engineering and Technology, London, UK, 2019, 354 p.

[6] *RFC2898 PKCS #5: Password-Based Cryptography Specification, Version 2.0*, [Online]. Available: https://tools.ietf.org/html/rfc2898

Стаття надійшла до редколегії **17.11.2020**

Recommendations for choosing a method of safe storage of passwords

Using passwords remains the most common way to authenticate users for various types of information systems. This poses the challenge of securing the storage of user authentication information and protecting it from unauthorized access. In practice, various algorithms for secure password storage have become widespread. Mutually contradictory requirements for such algorithms for secure password storage, which on the one hand must be complex enough to counter various attacks, and on the other – simple to ensure the speed of the information system – determine the relevance of the study. There is a significant threat of direct search, especially given the fact that the computing power of CPUs and GPUs is constantly growing. Therefore, there is a need to be able to change the complexity of the hash code calculation, and therefore the amount of computation and time so as to significantly complicate the attack, but not cause discomfort to the end user due to the delay in password verification. Among the known methods of secure password storage are password encryption, the use of the hash function in the classical version, as well as the addition of salt and the use of iterations to calculate the hash code. The comparative analysis of the given methods is carried out in the work, their advantages and disadvantages are established, expedient areas of application of each method are outlined, the corresponding recommendations are developed. For the computational experiment, the tools of the Microsoft .NET Core 3.1 platform were used, which made it possible to set the time indicators of the hash code generation algorithm depending on the set parameters of the algorithm. The data obtained from the experiment can be used to select a method of securely storing passwords.

Keywords: *information protection; user authentication; encryption; hash code calculation algorithms; computational experiment.*

Олександр Махович,
кандидат технічних наук, асистент кафедри мережевих та інтернет-технологій факультету інформаційних технологій Київського національного університету імені Тараса Шевченка.

Olexander Mahovich,
Candidate of Technical Sciences, Assistant of the Department of Network and Internet Technologies, Faculty of Information Technologies, Taras Shevchenko National University of Kyiv.

Віра Миколайчук,
асистент кафедри інформаційних систем та технологій інституту інформаційних технологій Державного університету телекомунікацій.

Vira Mykolaichuk,
Assistant of the Department of Information Systems and Technologies of the Institute of Information Technologies of the State University of Telecommunications.

Роман Миколайчук,
доктор технічних наук, доцент, доцент кафедри мережевих та інтернет-технологій факультету інформаційних технологій Київського національного університету імені Тараса Шевченка.

Roman Mykolaichuk,
Dr. Tech. Sciences, Associate Professor, Associate Professor of the Department of Network and Internet Technologies, Faculty of Information Technology, Taras Shevchenko National University of Kyiv.