Methods of information systems protection

UDC 004.056.3 doi: https://doi.org/10.20998/2522-9052.2024.4.11

Volodymyr Fedorchenko[1], Olha Yeroshenko[1], Oleksandr Shmatko[2], Oleksii Kolomiitsev[2], Murad Omarov[1]

[1] Kharkiv National University of Radio Electronics, Kharkiv, Ukraine
[2] National Technical University "Kharkiv Polytechnic Institute", Kharkiv, Ukraine

PASSWORD HASHING METHODS AND ALGORITHMS ON THE .NET PLATFORM

Abstract: Web applications, which are widely used to provide services and collect information, have become a major target for attackers, especially with the emergence of government services that process sensitive data. The .NET software platform, popular for developing web applications, includes built-in hashing algorithms (HA) and key generation functions (KDF) to protect passwords. However, these were developed over two decades ago for different levels of threats. More modern alternatives, such as Bcrypt, Scrypt, and Argon2, offer improved protection against modern GPU, ASIC, and FPGA attacks, but require third-party implementation. Given the critical role of password protection in protecting user information, this research investigates the effectiveness of various hashing mechanisms on the .NET platform, which is an urgent need for securing modern web applications. **The subject of study** in the article is the features of hashing algorithms built-in and available in the libraries of the .NET software platform for password protection as the main aspect of user authentication. **The purpose of the work** is to compare and analyse the hashing algorithms built-in and available in the libraries of the .NET software platform for password protection as the main aspect of user authentication. **Objectives**: to review built-in algorithms such as MD5, SHA and PBKDF2, as well as third-party implementations of modern key derivation functions such as Bcrypt, Scrypt and Argon2, and to investigate their performance and cryptographic strength. **Methods used:** This included measuring hashing speeds for different password sets and analysing attack resistance using tools such as Hashcat and data from independent security research. **The results** show that while built-in algorithms such as MD5 and SHA256 are fast, they do not provide protection against modern threats such as rainbow table attacks and GPU-accelerated brute-force attempts. PBKDF2, which is standard in ASP.NET Core Identity, provides better security but is vulnerable to attacks using specialised hardware. Among the modern algorithms, Argon2 demonstrated the best balance of security and performance, providing protection against GPU, ASIC, and FPGA-based attacks. Conclusions. The study **concluded** that Argon2 is the recommended algorithm for password hashing on the .NET platform, while Bcrypt is a suitable alternative for legacy applications. PBKDF2 with a high number of iterations can still provide strong protection. **A promising direction** for further research may be to determine whether modern memory-intensive key derivation functions can be used to improve password security in .NET applications.

Keywords: hash function; hashing algorithm; key generation function; computer systems.

Introduction

Web applications are the most popular means of providing services and gathering information for many institutions [1, 2]. Because of their wide distribution, they are an attractive target for attackers. This is worsened by the emergence of state services, "DIIA" and others, which process important confidential information. Software platform (SP) .NET is a popular solution for developing web applications. This is because it has strong industry support from Microsoft, has quality developer tools, and uses multiple programming languages. The vast majority of computer systems (CS) uses passwords to ensure data security [3, 4]. A password is a user-remembered secret [5, 6] consisting of several typeable characters. Passwords are secured using hashing and other techniques.

A hashing algorithm (HA) is a mathematical function that distorts data and makes it unreadable [7]. AHs are one-way programs, so the input cannot be deciphered by anyone else. Hashing always protects the input data, so even if someone gains access to the storage where the hash values are stored, the input data will remain secure. Hashing is also used for digital signatures and when indexing data.

Hash functions (HF) also have disadvantages, because of them, key generation functions (KDF) are now more actively used to protect passwords, some of them are based on HF or encryption algorithms.

SP .NET has built-in APIs such as MD5, SHA-1, SHA-2 and the key generation function (KDF) PBKDF2, which ASP.NET Core Identity uses for hashing by default. These HAs were created more than 20 years ago for a different level and nature of threats.

There are also more modern KDF alternatives used for hashing passwords, namely Bcrypt, Scrypt and Argon2, these HAs promise protection against graphics processing units (GPUs), application-specific integrated circuits (ASICs), and user-programmable gate arrays (FPGA). These KDFs are not built into the .NET framework, but their implementations can be obtained from third-party libraries. Due to the fact that password protection is one of the main aspects of protecting confidential user information, the study of this mechanism on SP .NET is an urgent task.

1. Literature Survey

1.1. Features of hashing functions and algorithms. Hashing functions play an important role in cryptography because they have important characteristics that help in data authentication and ensure the security of sensitive data such as passwords [8, 9]. Such HFs are known as cryptographic and their characteristics include the following [10, 11]:

- irreversibility;
- determinism;
- resistance to collisions;
- avalanche effect;
- speed.

Irreversibility of HA. HAs are one-way functions – that is, it is not possible to calculate the input data using the hash value. This means, it is possible to easily hash the input data, but it is not possible to extract the input data from its hash value, until the HA is broken. This is important because hashes are used to store passwords on public servers. Since hashing is irreversible, attackers will not be able to recover the password from the hash, even if they get their hands on a database of password hashes. That is why passwords are never stored in public.

HA determinism. The output length of all HA results must be the same, regardless of the length of the input data. This is convenient for allocating space for a hash value in a data structure, file format, or network protocol field because you know how long the hash value is. This also helps prevent attackers from knowing how large the input value was because all output hashes, regardless of how long or short the input value is, are of fixed length and do not change.

Resistance to collisions or collisions of HA. When hashing, a collision is considered to have occurred when identical hash values were obtained as a result of hashing different input data. If the attackers become aware of the collision, it will be possible to bypass the CS protection. This is known as a hash collision attack.

Another problem is rainbow tables. Attackers can create a huge number of precomputed combinations of input data and their hash values. This table will allow you to quickly search for input data.

This is why all HAs must be collision resistant. One way to reduce password hashing collisions and reduce the risk of rainbow table attacks is to use salting.

Avalanche effect of HA. A feature of hash functions is that even the slightest change in the input data leads to a significant change in the output hash value. This ensures that no one can decipher the original text.

HA speed. Many areas of use of hashing require high-speed algorithms to calculate the hash value. However, not all hash functions need to be fast. Some functionality requires hash functions to be slow. This is necessary so that it is more difficult for attackers to use the method of sorting or rainbow tables to obtain input data.

1.2. Key generation functions. KDF is used to generate one or more cryptographic keys from a closed input value [12]. KDFs return bytes suitable for cryptographic operations from passwords or other data sources using a pseudorandom function. Different KDFs are suitable for different tasks such as [13]:

- obtaining or stretching a cryptographic key;
- hashing passwords.

In cryptography, key stretching techniques are used to make a potentially weak key, usually a password or passphrase, more secure against an iterative attack by increasing the resources, time, and possibly space required to test each possible key. Human-generated passwords or passphrases are often short or predictable enough to make passwords easy to crack, and key stretching is designed to make such attacks more difficult by making the basic step of trying to pick a single candidate password more difficult. Key stretching also improves security in some real-world applications where the key length was limited by simulating a longer key length from the perspective of an attacker using an iterative method.

When storing passwords, it is advisable to use an algorithm that requires high computational costs. Users would only need to calculate it once, while attackers would need to do it billions of times. An ideal password store with KDF would be demanding on both computational and memory resources [13].

Different KDFs have different additional parameters such as duty ratio and context information fields.

At a minimum, KDF for hashing passwords has the following parameters [14]:

- incoming data;
- salt;
- number of iterations or work factor.

KDFs with adjustable duty ratios are used to store passwords. KDF is better than simple HF even with the use of salt, because the work factor can be chosen in such a way as to make an exhaustive search in the space of probable passwords expensive (Fig. 1).

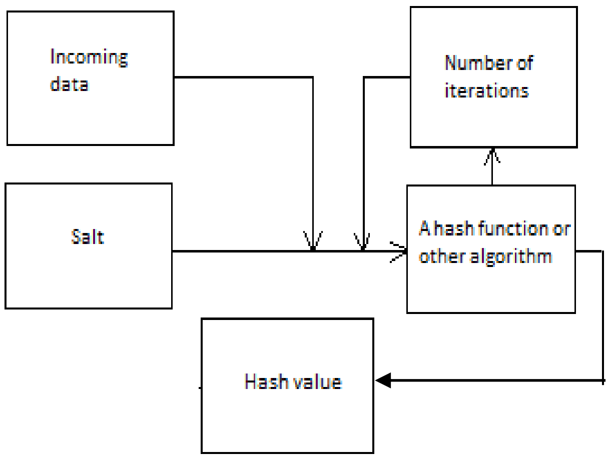


Fig. 1. General view of the key generation function

Therefore, for SP .NET, it is an urgent task to establish the most optimal algorithm for password protection. The work analyzes built-in algorithms, namely HF MD5, SHA and KDF PDKDF2. Among the algorithms that are not built into SP.NET, the following KDFs are selected:

- Bcrypt;
- Scrypt;
- Argon2.

Data AH was chosen according to the recommendations of the OWASP organization regarding password protection [15]. Indicators based on which analysis will be performed [16]:

- minimum, maximum and average hash value calculation speed;
- the time required to obtain the input value by the method of complete enumeration.

1.3. Built-in hash functions and key generation functions. Hashing functions are represented as classes in the namespace System.Security.Cryptography SP .NET:

- MD5;
- SHA1;
- SHA256;
- SHA384;
- SHA512.

These abstract classes are derived from a class HashAlgorithm. To calculate the hash value using these functions, the following steps are required [17]:

- instantiate the hash algorithm. You can choose from MD5, SHA1, SHA256, SHA384 and SHA512;
- call the ComputeHash method, passing an array of bytes. A byte array can be derived from any raw data;
- the ComputeHash method, upon successful execution, will return a byte array representing the hash value.

After a successful ComputeHash execution, the byte array of the hash value can be converted to another type for more convenient storage.

Key generation functions are presented as classes in the System.Security.Cryptography namespace SP.NET (Fig. 2**)**:

- PasswordDeriveBytes;
- Rfc2898DeriveBytes.

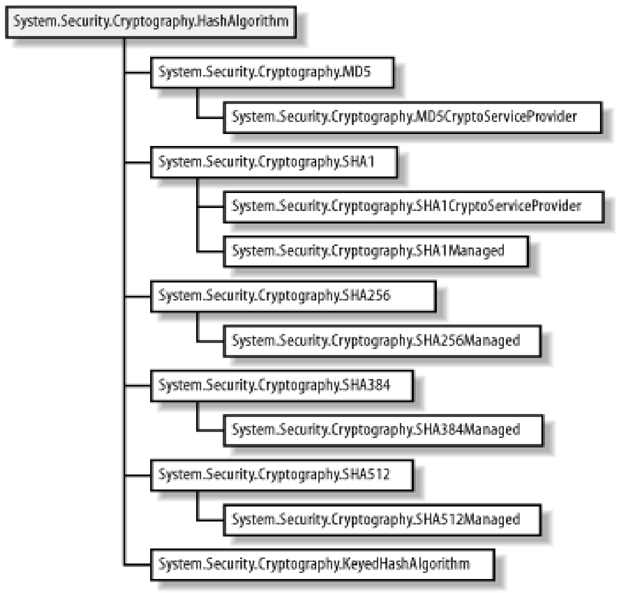

System.Security.Cryptography.HashAlgorithm
System.Security.Cryptography.MD5
System.Security.Cryptography.MD5CryptoServiceProvider
System.Security.Cryptography.SHA1
System.Security.Cryptography.SHA1CryptoServiceProvider
System.Security.Cryptography.SHA1Managed
System.Security.Cryptography.SHA256
System.Security.Cryptography.SHA256Managed
System.Security.Cryptography.SHA384
System.Security.Cryptography.SHA384Managed
System.Security.Cryptography.SHA512
System.Security.Cryptography.SHA512Managed
System.Security.Cryptography.KeyedHashAlgorithm

Fig. 2. Built-in hashing functions

Rfc2898DeriveBytes is an implementation of the PDKDF2 AX and is used by default in applications using ASP.NET. The implementation of the algorithm, depending on the environment where the application is deployed, is selected using the KeyDerivation.Pbkdf2 wrapper class from the Microsoft.AspNetCore.Cryptography.KeyDerivation namespace.

The conducted analysis of built-in HF shows that in SP.NET built-in HA passwords that provided sufficient protection at the time of their creation. But since then, the demand for password protection has increased [18–20].

HF MD5 has been cracked, but is still used by a large number of CS. HFs of the SHA family can provide protection against attackers with limited capabilities, but by default are vulnerable to attacks using rainbow tables. The PBKDF2 algorithm, which is a standard means of hashing passwords in .NET, like other classic HFs, is vulnerable to attacks using GPUs, ASICs, and FPGAs that were not widespread at the time of their creation. To protect against these threats, more modern algorithms are proposed, the implementations of which are not built into SP.NET.

2. Current key generation functions

Although the PBKDF2 key function has advantages over MD5 and SHA, it also has the disadvantage that it is not resistant to brute-force attacks using GPUs and ASICs, because PBKDF2 uses a relatively small amount of RAM and can be computed on GPUs or ASICs.

Modern key generation functions such as Bcrypt, Scrypt, and Argon2 are designed to be resistant to dictionary, GPU, and ASIC attacks. These functions generate a key from a password and require a large amount of memory, which does not allow for fast parallel computations on GPUs or ASICs.

Algorithms such as Bcrypt, Scrypt, and Argon2 are considered more secure key generation functions because they require:

- salt;
- a large number of iteration iterations;
- a lot of CPU resources;
- a large amount of RAM.

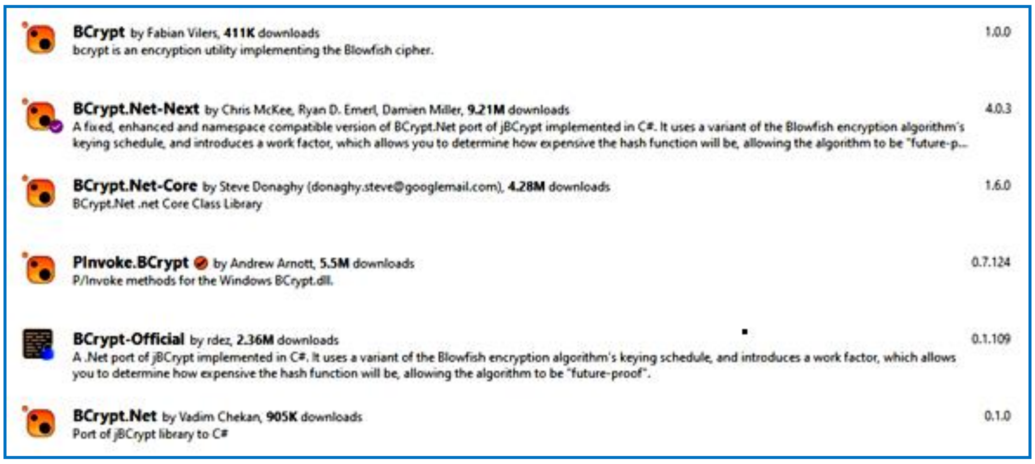
This makes it very difficult to develop hardware to significantly accelerate password cracking. Modern computer computing is more limited by the speed of memory, so memory access is the bottleneck of computing [21, 22]. Faster access to RAM will speed up calculations.

When deriving a key from a given password uses a lot of CPU resources and a lot of RAM, password cracking is slow and inefficient, even with very good password cracking hardware and software. The purpose of modern key generation functions is to make it virtually impossible to perform a brute-force attack.

2.1. Bcrypt key generation function. Bcrypt is an adaptive HF based on the Blowfish symmetric block cipher cryptographic algorithm. It uses a key factor that governs the hashing cost, which is the most prominent feature of Bcrypt. This provides an opportunity to increase the cost – time and computing costs of hashing in the future when CSs become more powerful.

Bcrypt uses a 128-bit salt and encrypts a 192-bit magic value. The Bcrypt algorithm encrypts the input password according to the established key factor using Blowfish. It takes advantage of the expensive key setup in eksblowfish [23].

Bcrypt was created at the same time as PBKDF2, the use of the Blowfish cipher at its core makes Bcrypt slower and more secure than PBKDF2 against brute force attacks, also Bcrypt requires more memory than modern GPU computing units have. But Bcrypt remains vulnerable when using ASICs and FPGAs. One of its drawbacks is the input data size limit of 72 bytes. Using longer passwords will require prior hashing with a different algorithm. There are a large number of implementations of this algorithm in the SP .NET libraries, they have a significant number of downloads and most were updated in 2022 (Fig. 3**)**.



BCrypt by Fabian Vilers, **411K** downloads 1.0.0
bcrypt is an encryption utility implementing the Blowfish cipher.

BCrypt.Net-Next by Chris McKee, Ryan D. Emerl, Damien Miller, **9.21M** downloads 4.0.3
A fixed, enhanced and namespace compatible version of BCrypt.Net port of jBCrypt implemented in C#. It uses a variant of the Blowfish encryption algorithm's keying schedule, and introduces a work factor, which allows you to determine how expensive the hash function will be, allowing the algorithm to be "future-p...

BCrypt.Net-Core by Steve Donaghy (donaghy.steve@googlemail.com), **4.28M** downloads 1.6.0
BCrypt.Net .net Core Class Library

PInvoke.BCrypt by Andrew Arnott, **5.5M** downloads 0.7.124
P/Invoke methods for the Windows BCrypt.dll.

BCrypt-Official by rdez, **2.36M** downloads 0.1.109
A .Net port of jBCrypt implemented in C#. It uses a variant of the Blowfish encryption algorithm's keying schedule, and introduces a work factor, which allows you to determine how expensive the hash function will be, allowing the algorithm to be "future-proof".

BCrypt.Net by Vadim Chekan, **905K** downloads 0.1.0
Port of jBCrypt library to C#

Fig. 3: Implementations of the Bcrypt algorithm in the libraries of the .NET software platform

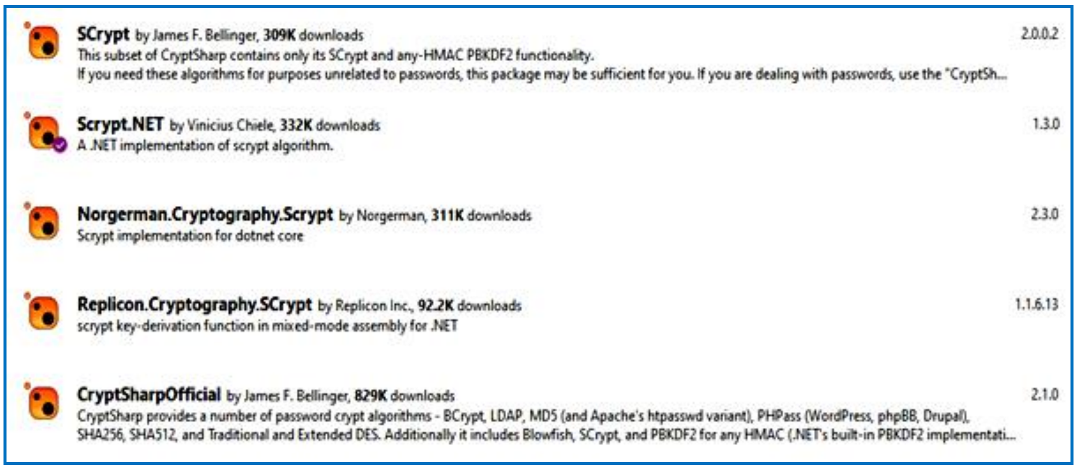
2.2. Scrypt key generation function. Scrypt is a secure cryptographic key generation function The Scrypt algorithm is described in the Internet standard RFC 7914. It is memory intensive and designed to prevent attacks using GPUs, ASICs and FPGAs, highly efficient password cracking hardware. The Scrypt algorithm takes several input parameters and outputs a derived key as output. The Scrypt parameters are as follows:

- the number of iterations affects memory and processor usage;
- block size affects memory and processor usage;
- the parallelism factor, threads are executed in parallel, affects the amount of memory and processor load;
- input password, a minimum length of 8-10 characters is recommended;
- target generated random bytes, minimum 64 bits, 128 bits recommended;
- length of the generated key, how many bytes to generate as output data.

Memory access in Scrypt is performed in a strictly dependent order at each step, so memory access speed is the bottleneck of the algorithm.

When properly configured, Scrypt is considered a highly secure key generation function, so it can be used as a general-purpose password hashing algorithm for, for example, encryption of wallet, file, or application passwords. Scrypt is popular in cryptocurrency validation schemes, primarily Litecoin as well as Tenebrix and Dogecoin, and inspired the design of one of the winners of the Argon2d password hashing contest [24]. On the other hand, large memory requirements make this algorithm unpopular for password protection in most resource-constrained applications.

This algorithm has a small number of implementations on SP .NET, they have a small number of downloads and most were updated more than 5 years ago, which creates risks when using this algorithm (Fig. 4**)**.



SCrypt by James F. Bellinger, **309K** downloads 2.0.0.2
This subset of CryptSharp contains only its SCrypt and any-HMAC PBKDF2 functionality.
If you need these algorithms for purposes unrelated to passwords, this package may be sufficient for you. If you are dealing with passwords, use the "CryptSh...

Scrypt.NET by Vinicius Chiele, **332K** downloads 1.3.0
A .NET implementation of scrypt algorithm.

Norgerman.Cryptography.Scrypt by Norgerman, **311K** downloads 2.3.0
Scrypt implementation for dotnet core

Replicon.Cryptography.SCrypt by Replicon Inc., **92.2K** downloads 1.1.6.13
scrypt key-derivation function in mixed-mode assembly for .NET

CryptSharpOfficial by James F. Bellinger, **829K** downloads 2.1.0
CryptSharp provides a number of password crypt algorithms - BCrypt, LDAP, MD5 (and Apache's htpasswd variant), PHPass (WordPress, phpBB, Drupal), SHA256, SHA512, and Traditional and Extended DES. Additionally it includes Blowfish, SCrypt, and PBKDF2 for any HMAC (.NET's built-in PBKDF2 implementati...

Fig. 4: Implementations of the Scrypt algorithm in the libraries of the .NET software platform

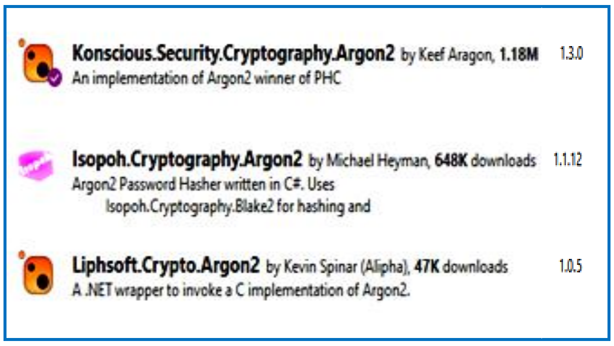
2.3. Argon2 key generation function. Argon2 is a state-of-the-art secure key generation function that is ASIC and GPU resistant. It has better resistance to password cracking, when properly configured, than PBKDF2, Bcrypt, and Scrypt, for similar CPU and RAM usage configuration parameters.

Argon2 was designed with the following primary goal in mind: to maximize the cost of exhaustive search on non-x86 architectures, so that moving even to dedicated ASICs would not provide a significant advantage over performing exhaustive search on a desktop [17].

The Argon2 function has several options:

- Argon2d, provides strong protection when using the GPU, but has potential side-channel attacks (possible in very special situations);
- Argon2i, provides less protection when using the GPU, but does not have side-channel attacks;
- Argon2id is recommended because it combines Argon2d and Argon2i.

This algorithm has a small number of implementations on SP .NET, which can be explained by the novelty of the algorithm, but there are implementations with a significant number of downloads and updates in 2022 (Fig. 5**)**.



Konscious.Security.Cryptography.Argon2 by Keef Aragon, **1.18M** 1.3.0
An implementation of Argon2 winner of PHC

Isopoh.Cryptography.Argon2 by Michael Heyman, **648K downloads** 1.1.12
Argon2 Password Hasher written in C#. Uses Isopoh.Cryptography.Blake2 for hashing and

Liphsoft.Crypto.Argon2 by Kevin Spinar (Alipha), **47K downloads** 1.0.5
A .NET wrapper to invoke a C implementation of Argon2.

Fig. 5. Implementations of the Argon2 algorithm in the libraries of the .NET software platform

2.4. Security of implementations of third-party hashing functions. The security and absence of errors in the built-in hash functions is guaranteed by the Microsoft corporation, which supports SP .NET. The HA considered in this section are not built into the .NET software platform and their implementation must be obtained from the SP libraries.

The security of libraries is solely the responsibility of its developer, and if the library is not already popular, updates of such libraries will not be regular, which excludes the possibility of quickly fixing vulnerabilities. Statistically, among the vulnerabilities of SP .NET libraries, vulnerabilities of a high degree of seriousness make up 70.7% of the total number, and their correction by developers is extremely important [5].

Among the algorithms that are being considered, implementations of the Scrypt algorithm are raising doubts about their security.

Alternative HAs promise more reliable protection against overrun attacks than PBKDF2, which is the standard SP .NET defense. Modern algorithms are also designed to protect against attacks using GPUs, ASICs, and FPGAs, but this protection requires more computing time and CS resources.

3. Methodology

3.1. Hash speed comparison. To analyze the speed, the average, minimum and maximum time spent on calculating the hash value using the studied algorithms was measured. The hash value was calculated for several sets of passwords:

- set of 1,500 worst passwords;
- set 2, 10,000 most common passwords;
- set 3, 20 high entropy generated passwords include lowercase and uppercase letters, numbers and other characters, length 8 characters;
- set 4, 20 high entropy generated passwords include lowercase and uppercase letters, numbers and other characters, length 12 characters;
- set 5, 20 high entropy generated passwords include lowercase and uppercase letters, numbers and other characters, length 15 characters.

The calculation time of the key generation functions depends on the function parameters. The current parameters were obtained from the resources of the organization OWASP, an open project on the security of web applications.

The following parameters were selected for key generation functions:

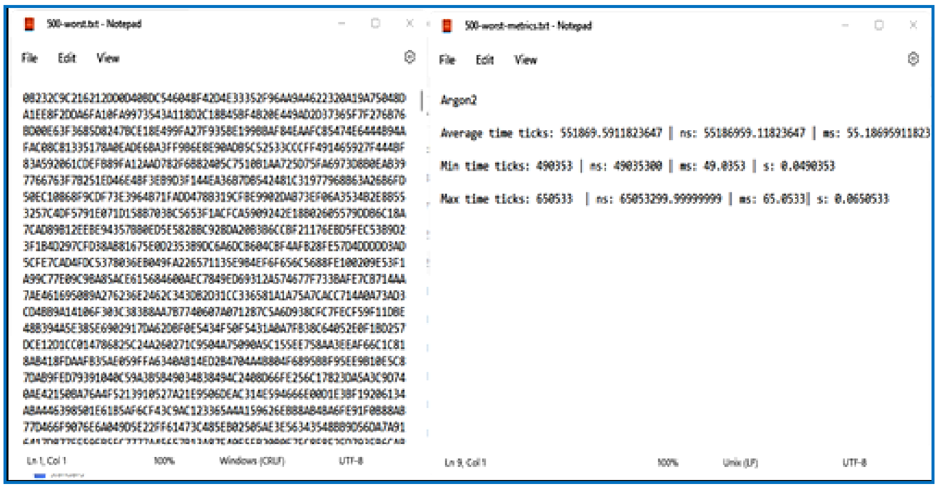
- PBKDF2 (on order), correspond to the configuration in ASP.NET Core Identity. Number of iterations 10000, salt size 128 bits, hash size 256 bits;
- PBKDF2 (recommended), modified according to recommendations. Number of iterations 310000, salt size 128 bits, hash size 256 bits;
- Bcrypt, the parameters are left unchanged because the operating factor is set by default to 11, which exceeds the recommended 10;
- Scrypt (degree of parallelization 1), degree of parallelization 1, block size 8, memory size 64 MiB;
- Scrypt (degree of parallelization 4), degree of parallelization 4, block size 8, memory size 16 MiB;
- Argon2 (1 iteration), degree of parallelization 1, minimum number of iterations 1, minimum memory size 37 MiB;
- Argon2 (2 iterations), degree of parallelization 1, minimum number of iterations 2, minimum memory size 15 MiB.

Hashing and measurements were performed on a CS with an AMD Ryzen 7 6800H CPU and an NVIDIA GeForce RTX 3050 Ti Laptop GPU. Operating system CS Windows 11, version SP .NET 6. .NET and Visual Studio diagnostic tools were used to measure metrics 2022.

As a result of hashing the sets, files with hash values and calculation metrics were obtained. For demonstration, the results for Argon2 AX are given (Fig. 6).

The obtained hash values were used for security analysis, and the metrics allowed us to compare the speed.

For all sets of passwords, measurements of the average, minimum and maximum time to calculate the hash value were performed. The analysis was performed on the average value of the results for all sets, the graph for the relative comparison of the HA speed is shown below (Fig. 7). A detailed comparison of the hash value calculation results for each HA and its set parameters provides the average time, minimum and maximum time in nanoseconds (Table 1).


500-worst.txt - Notepad

File Edit View

```
08232C9C2162120D0D400DC546048F4204E33352F96AA9A4622320A19A75040D
A1EE8F2DDA6FA10FA9973543A11802C188458F4820E449AD2D37365F7F276876
BD00E63F3685D8247BCE18E499FA27F9358E19988AF84EAAFC85474E6444894A
FAC08C81335178A0EADE6BA3FF986E8E90AD85C52533CCCFF491465927F4448F
83A592061CDEFB89FA12AAD782F6882405C751081AA725D75FA69730880EAB39
7766763F78251ED46E48F3EB9D3F144EA3687D8542481C31977968863A2686FD
58EC10868F9CDF73E3964871FAD04788319CF8E9902DA873EF06A3534B2E8855
3257C4DF5791E071D15887038C5653F1ACFCA5909242E18802605579DD86C18A
7CAD89B12EEBE943578B0ED5E5828BC928DA20B386CCBF21176EBD5FEC538902
3F1B4D297CFD38A881675E0D2353B9DC6A6DCB604CBF4AFB28FE57D4DDDDD3AD
5CFE7CAD4FDC5378036E8049FA226571135E984EF6F656C5688FE100209E53F1
A99C77E09C98A85ACE615684600AEC7849ED69312A574677F733BAFE7C8714AA
7AE461695089A276236E2462C343D82031CC336581A1A75A7CACC714A0A73AD3
CD48B9A14106F303C38388AA7B7740607A071287C5A6D938CFC7FECF59F11D8E
488394A5E385E6902917DA620BF0E5434F50F5431A0A7F838C64052E0F180257
DCE1201CC014786825C24A260271C9504A75090A5C155EE758AA3EEAF66C1C81
8AB418FDAAFB35AE059FFA6348A814ED2B4704A48804F689588F95EE9810E5C8
7DAB9FED79391040C59A3B5849034838494C2480D66FE256C17823DA5A3C9074
0AE421508A76A4F5213910527A21E9506DEAC314E594666E0001E38F19206134
A8A446398501E61B5AF6CF43C9AC123365A4A159626E888AB48A6FE91F0888A8
77D466F9076E6A04905E22FF61473C485EB02585AE3E563435488B9D560A7A91
[illegible]
```

Ln 1, Col 1 100% Windows (CRLF) UTF-8

500-worst-metrics.txt - Notepad

File Edit View

```
Argon2

Average time ticks: 551869.5911823647 | ns: 55186959.11823647 | ms: 55.18695911823

Min time ticks: 490353 | ns: 49035300 | ms: 49.0353 | s: 0.0490353

Max time ticks: 650533  | ns: 65053299.99999999 | ms: 65.0533| s: 0.0650533
```

Ln 9, Col 1 100% Unix (LF) UTF-8

Fig, 6: Calculated hash values and calculation metrics

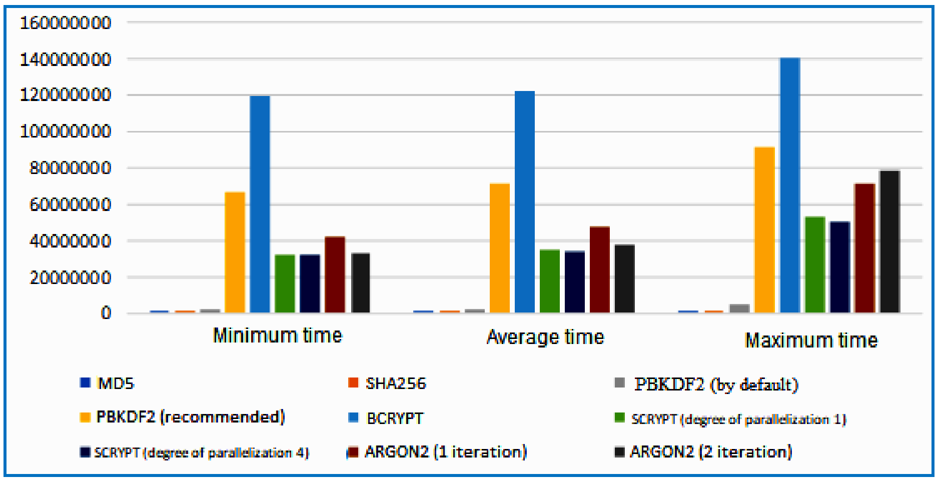


Fig. 7. Average password hash rates for all sets

Table 1 – **Metrics for calculating hash values in nanoseconds**

Metrics	Minimum time	Average time	Maximum time
MD5	320	14042.2	395240
SHA256	180	1123.2	532380
PBKDF2 (by default)	2001580	2178747.4	4974460
PBKDF2 (recommended)	66961080	70902501	90793120
BCRYPT	119422860	121843986.8	140523459.6
SCRYPT (degree of parallelization 1)	31593020	34807792.4	52724700
SCRYPT (degree of parallelization 4)	31740859.8	34016152.6	49895700
ARGON2 (1 iteration)	41610720	47300997.6	70995859.8
ARGON2 (2 iteration)	33106679.8	37634215.2	78174880

From the measurement results, it can be seen that classical HF and PBKDF2 with default parameters are very fast. Other algorithms take more time, but this is still enough for use in web applications. Bcrypt has the slowest speed, the other key generation functions take about the same amount of time to calculate the hash value.

3.2. Security comparison. To analyze the security of the algorithms, attacks on sets of passwords were performed. The attacks used a full sweep method, a dictionary attack was also performed for sets with common passwords, the Rockyou dictionary was used, this dictionary contains 32 million passwords that were obtained in the attack on the company of the same name. To carry out the attack, a CS with an AMD Ryzen 7 6800H central processor and an NVIDIA GeForce RTX 3050 Ti Laptop GPU was used. Hashcat attack tool.

For algorithms that have sufficient protection against existing CS or that have failed to be attacked using the Hashcat tool. The analysis was carried out using research materials of independent organizations.

As mentioned above, the MD5 algorithm is currently not considered suitable for hashing passwords, primarily due to the presence of collisions.

Dictionary attacks performed. For the worst 500 passwords, it was possible to get 497 passwords or 99.60% in 2 seconds. For the most common 10,000 passwords, it was possible to get 9,471 passwords or 94.71% in 31 seconds.

The worst 500 passwords brute force attack (Fig. 8):

- 358 passwords or 71.74% were received in 1 second;
- 454 passwords or 90.98% were received in 21 seconds;
- 498 passwords or 99.80% were received in 3 minutes and 16 seconds;
- it took 8 minutes and 3 seconds to get all the passwords.

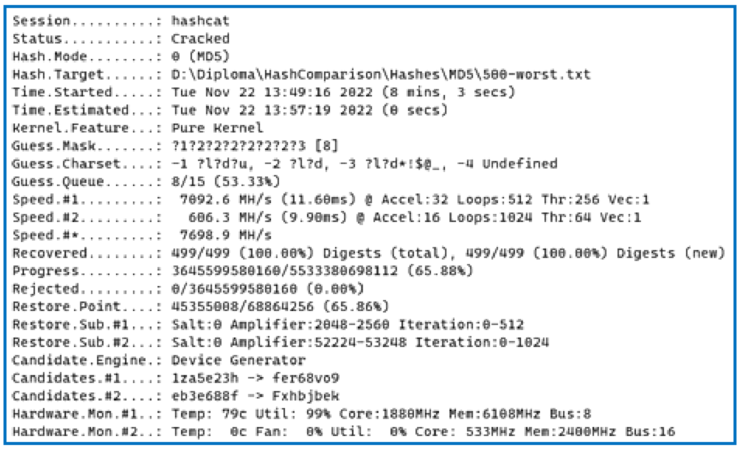
The most common 10,000 passwords attack by the method of exhaustive search:

- 1078 passwords or 10.78% were received in 4 seconds;
- 5495 passwords or 54.95% were received in 10 seconds;
- 7451 passwords or 74.51% were received in 27 seconds;
- 8977 passwords or 89.77% were received in 2 minutes 36 seconds;
- 9198 passwords or 91.98% were obtained in 4 minutes and 19 seconds;
- 9326 passwords were obtained in 9 minutes 34 seconds or 93.26% maximum speed of 7892 mega hashes per second, then the sorting of passwords with a length of 9 characters began, which is unproductive for equipment of this level;
- in 2 hours, 9931 passwords or 99.31% were obtained, the predicted time for sorting other 9-character combinations was 6 hours 14 minutes, the attack was stopped.

From the above data, it can be seen that the MD5 algorithm is vulnerable to overrun attacks with a general-purpose CS, increasing the length of the password increases the time to obtain it, but for more powerful systems, this is not a problem.

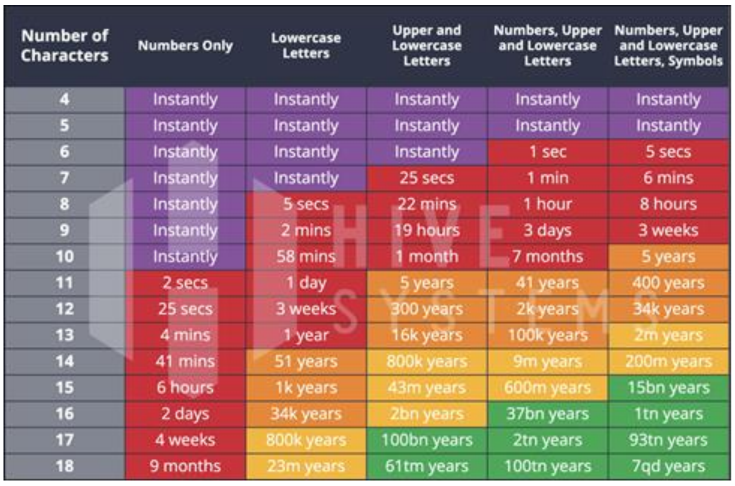
HiveSystems research results were used to confirm the results for long passwords [25].

The MD5 algorithm research results from HiveSystems are shown below (Fig. 9).



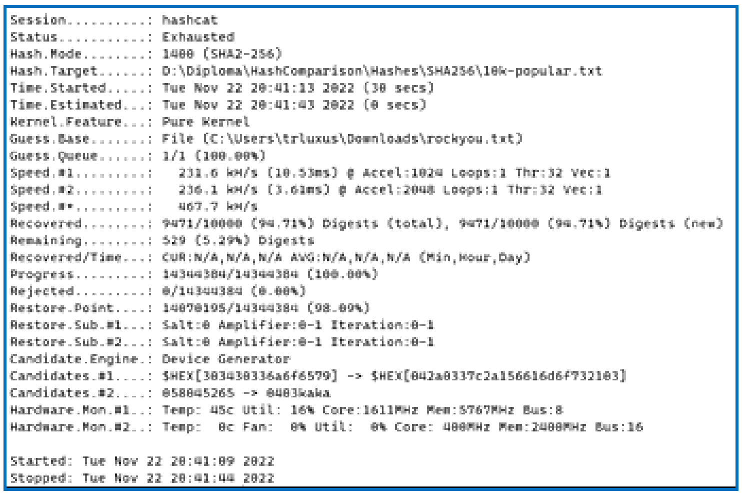
```
Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 0 (MD5)
Hash.Target......: D:\Diploma\HashComparison\Hashes\MD5\500-worst.txt
Time.Started.....: Tue Nov 22 13:49:16 2022 (8 mins, 3 secs)
Time.Estimated...: Tue Nov 22 13:57:19 2022 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Mask.......: ?1?2?2?2?2?2?2?3 [8]
Guess.Charset....: -1 ?l?d?u, -2 ?l?d, -3 ?l?d*!$@_, -4 Undefined
Guess.Queue......: 8/15 (53.33%)
Speed.#1.........:  7092.6 MH/s (11.60ms) @ Accel:32 Loops:512 Thr:256 Vec:1
Speed.#2.........:   606.3 MH/s (9.90ms) @ Accel:16 Loops:1024 Thr:64 Vec:1
Speed.#*.........:  7698.9 MH/s
Recovered........: 499/499 (100.00%) Digests (total), 499/499 (100.00%) Digests (new)
Progress.........: 3645599580160/5533380698112 (65.88%)
Rejected.........: 0/3645599580160 (0.00%)
Restore.Point....: 45355008/68864256 (65.86%)
Restore.Sub.#1...: Salt:0 Amplifier:2048-2560 Iteration:0-512
Restore.Sub.#2...: Salt:0 Amplifier:52224-53248 Iteration:0-1024
Candidate.Engine.: Device Generator
Candidates.#1....: 1za5e23h -> fer68vo9
Candidates.#2....: eb3e688f -> Fxhbjbek
Hardware.Mon.#1..: Temp: 79c Util: 99% Core:1880MHz Mem:6108MHz Bus:8
Hardware.Mon.#2..: Temp:  0c Fan:  0% Util:  0% Core: 533MHz Mem:2400MHz Bus:16
```

Fig. 8. The worst 500 passwords method of exhaustive search



Number of Characters	Numbers Only	Lowercase Letters	Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters, Symbols
4	Instantly	Instantly	Instantly	Instantly	Instantly
5	Instantly	Instantly	Instantly	Instantly	Instantly
6	Instantly	Instantly	Instantly	1 sec	5 secs
7	Instantly	Instantly	25 secs	1 min	6 mins
8	Instantly	5 secs	22 mins	1 hour	8 hours
9	Instantly	2 mins	19 hours	3 days	3 weeks
10	Instantly	58 mins	1 month	7 months	5 years
11	2 secs	1 day	5 years	41 years	400 years
12	25 secs	3 weeks	300 years	2k years	34k years
13	4 mins	1 year	16k years	100k years	2m years
14	41 mins	51 years	800k years	9m years	200m years
15	6 hours	1k years	43m years	600m years	15bn years
16	2 days	34k years	2bn years	37bn years	1tn years
17	4 weeks	800k years	100bn years	2tn years	93tn years
18	9 months	23m years	61tm years	100tn years	7qd years

Fig. 9. Time to get an MD5-protected password using the RTX 2080 [25]



```
Session..........: hashcat
Status...........: Exhausted
Hash.Mode........: 1400 (SHA2-256)
Hash.Target......: D:\Diploma\HashComparison\Hashes\SHA256\10k-popular.txt
Time.Started.....: Tue Nov 22 20:41:13 2022 (30 secs)
Time.Estimated...: Tue Nov 22 20:41:43 2022 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (C:\Users\trluxus\Downloads\rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:   231.6 kH/s (10.53ms) @ Accel:1024 Loops:1 Thr:32 Vec:1
Speed.#2.........:   236.1 kH/s (3.61ms) @ Accel:2048 Loops:1 Thr:32 Vec:1
Speed.#*.........:   467.7 kH/s
Recovered........: 9471/10000 (94.71%) Digests (total), 9471/10000 (94.71%) Digests (new)
Remaining........: 529 (5.29%) Digests
Recovered/Time...: CUR:N/A,N/A,N/A AVG:N/A,N/A,N/A (Min,Hour,Day)
Progress.........: 14344384/14344384 (100.00%)
Rejected.........: 0/14344384 (0.00%)
Restore.Point....: 14070195/14344384 (98.09%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Restore.Sub.#2...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: $HEX[303430336a6f6579] -> $HEX[042a0337c2a156616d6f732103]
Candidates.#2....: 058045265 -> 0403kaka
Hardware.Mon.#1..: Temp: 45c Util: 16% Core:1611MHz Mem:5767MHz Bus:8
Hardware.Mon.#2..: Temp:  0c Fan:  0% Util:  0% Core: 400MHz Mem:2400MHz Bus:16

Started: Tue Nov 22 20:41:09 2022
Stopped: Tue Nov 22 20:41:44 2022
```

Fig. 10. The most common 10,000 password attack by dictionary

The number of hash values that the RTX 2080 calculates in 1 second is 37085000000, for the current most powerful GPU RTX 3090, this value is 69379700000. MD5 also lacks protection against attacks using rainbow tables. For this reason, this algorithm is not recommended for password protection.

SHA256 is still considered suitable for protecting passwords, but it lacks protection against dictionary attacks and rainbow tables.

Let's perform dictionary attacks. For the worst 500 passwords, it was possible to get 497 passwords or 99.60% in 3 seconds.

For the most common 10,000 passwords, it was possible to get 9,471 passwords or 94.71% in 30 seconds (Fig. 10).

The worst 500 passwords brute force attack (Fig. 11):

- 358 passwords or 71.74% were received in 2 seconds, these are passwords up to 6 characters long;

- 454 passwords or 90.98% were received in 1 minute, these are passwords up to 7 characters long;
- it took 26 minutes and 28 seconds to get all the passwords.

The most common 10,000 passwords attack by the method of exhaustive search (Fig. 12):

- 5852 passwords or 58.52% were received in 11 seconds, these are passwords up to 6 characters long;
- 7904 passwords or 79.04% were received in 1 minute 2 seconds, these are passwords up to 7 characters long;
- 9834 passwords or 98.34% were received in 40 minutes 34 seconds, these are passwords with a length of 8 characters;
- 9326 passwords or 93.26% were obtained in 9 minutes 34 seconds, then the search for passwords with a length of 9 characters began, which is unproductive for equipment of this level;
- estimated time to go through other combinations of 9 symbols 29 hours, the attack was stopped.

SHA256 remains a reliable CHF, but lacks protection against dictionary attacks and rainbow tables require self-salting. Using SHA256 is possible, but not recommended because there are algorithms based on this HF, which require salt and number of iterations to slow down the algorithm.

The PBKDF2 algorithm is a key generation function, it requires a salt and the number of iterations, it basically uses the HF of the SHA family.

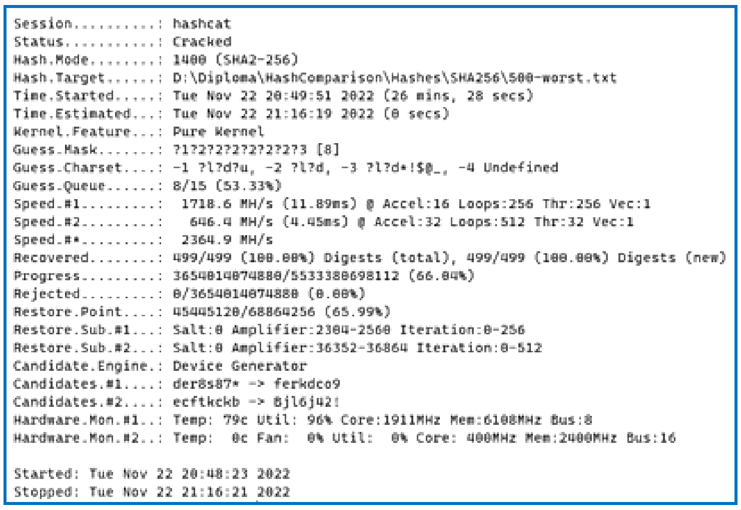
Applications that use SHA256-based PBKDF2 include password storage solutions LastPass, 1Password, and Bitwarden. The Hashcat tool supports PBKDF2 hashes in a specific format, so open source data was used for analysis. Below are the results of a full sweep attack using the RTX 3090 GPU from HiveSystems (Fig. 13) [25].

It can be seen that passwords with a significant length and high entropy are quite secure, the disadvantage of PBKDF2 is only the possibility of increasing the security by increasing the iterations.

Bcrypt is one of the slowest algorithms, which should make brute force attacks more difficult (Fig. 14).

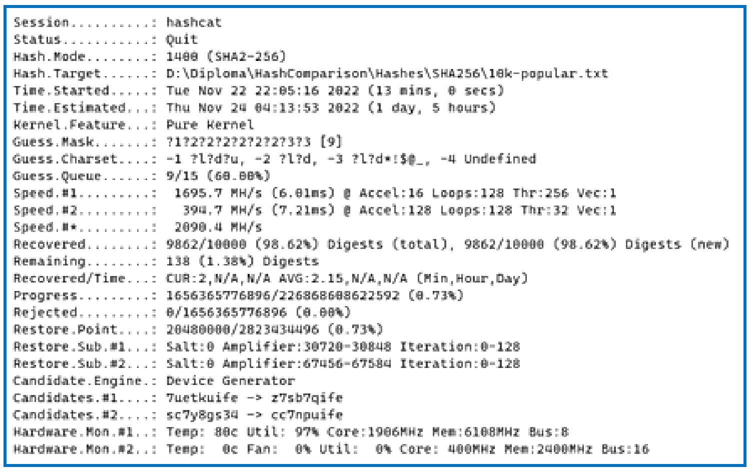
Even when running a dictionary attack in 6 minutes, only 146 passwords or 29.26% were obtained. Which shows the impossibility of attacks by the method of complete search on this CS.

ScatteredSecrets research was studied to investigate protection against more powerful CS and FPGA [24]. Below for Bcrypt are hash calculations per second for AMD EPYC 7401P CPU and Nvidia RTX-2080Ti GPU (Table 2).



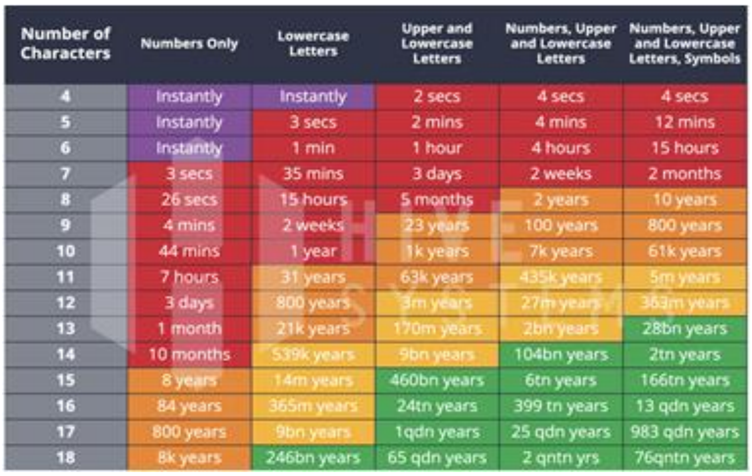
```
Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 1400 (SHA2-256)
Hash.Target......: D:\Diploma\HashComparison\Hashes\SHA256\500-worst.txt
Time.Started.....: Tue Nov 22 20:49:51 2022 (26 mins, 28 secs)
Time.Estimated...: Tue Nov 22 21:16:19 2022 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Mask.......: ?1?2?2?2?2?2?2?3 [8]
Guess.Charset....: -1 ?l?d?u, -2 ?l?d, -3 ?l?d*!$@_, -4 Undefined
Guess.Queue......: 8/15 (53.33%)
Speed.#1.........:  1718.6 MH/s (11.89ms) @ Accel:16 Loops:256 Thr:256 Vec:1
Speed.#2.........:   646.4 MH/s (4.45ms) @ Accel:32 Loops:512 Thr:32 Vec:1
Speed.#*.........:  2364.9 MH/s
Recovered........: 499/499 (100.00%) Digests (total), 499/499 (100.00%) Digests (new)
Progress.........: 3654014074880/5533380698112 (66.04%)
Rejected.........: 0/3654014074880 (0.00%)
Restore.Point....: 45445120/68864256 (65.99%)
Restore.Sub.#1...: Salt:0 Amplifier:2304-2560 Iteration:0-256
Restore.Sub.#2...: Salt:0 Amplifier:36352-36864 Iteration:0-512
Candidate.Engine.: Device Generator
Candidates.#1....: der8s87* -> ferkdco9
Candidates.#2....: ecftkckb -> 8jl6j42!
Hardware.Mon.#1..: Temp: 79c Util: 96% Core:1911MHz Mem:6108MHz Bus:8
Hardware.Mon.#2..: Temp:  0c Fan:  0% Util:  0% Core: 400MHz Mem:2400MHz Bus:16

Started: Tue Nov 22 20:48:23 2022
Stopped: Tue Nov 22 21:16:21 2022
```

Fig. 11. The worst 500 passwords brute force attack



```
Session..........: hashcat
Status...........: Quit
Hash.Mode........: 1400 (SHA2-256)
Hash.Target......: D:\Diploma\HashComparison\Hashes\SHA256\10k-popular.txt
Time.Started.....: Tue Nov 22 22:05:16 2022 (13 mins, 0 secs)
Time.Estimated...: Thu Nov 24 04:13:53 2022 (1 day, 5 hours)
Kernel.Feature...: Pure Kernel
Guess.Mask.......: ?1?2?2?2?2?2?2?3?3 [9]
Guess.Charset....: -1 ?l?d?u, -2 ?l?d, -3 ?l?d*!$@_, -4 Undefined
Guess.Queue......: 9/15 (60.00%)
Speed.#1.........:  1695.7 MH/s (6.01ms) @ Accel:16 Loops:128 Thr:256 Vec:1
Speed.#2.........:   394.7 MH/s (7.21ms) @ Accel:128 Loops:128 Thr:32 Vec:1
Speed.#*.........:  2090.4 MH/s
Recovered........: 9862/10000 (98.62%) Digests (total), 9862/10000 (98.62%) Digests (new)
Remaining........: 138 (1.38%) Digests
Recovered/Time...: CUR:2,N/A,N/A AVG:2.15,N/A,N/A (Min,Hour,Day)
Progress.........: 1656365776896/226868608622592 (0.73%)
Rejected.........: 0/1656365776896 (0.00%)
Restore.Point....: 20480000/2823434496 (0.73%)
Restore.Sub.#1...: Salt:0 Amplifier:30720-30848 Iteration:0-128
Restore.Sub.#2...: Salt:0 Amplifier:67456-67584 Iteration:0-128
Candidate.Engine.: Device Generator
Candidates.#1....: 7uetkuife -> z7sb7qife
Candidates.#2....: sc7y8gs34 -> cc7npuife
Hardware.Mon.#1..: Temp: 80c Util: 97% Core:1906MHz Mem:6108MHz Bus:8
Hardware.Mon.#2..: Temp:  0c Fan:  0% Util:  0% Core: 400MHz Mem:2400MHz Bus:16
```

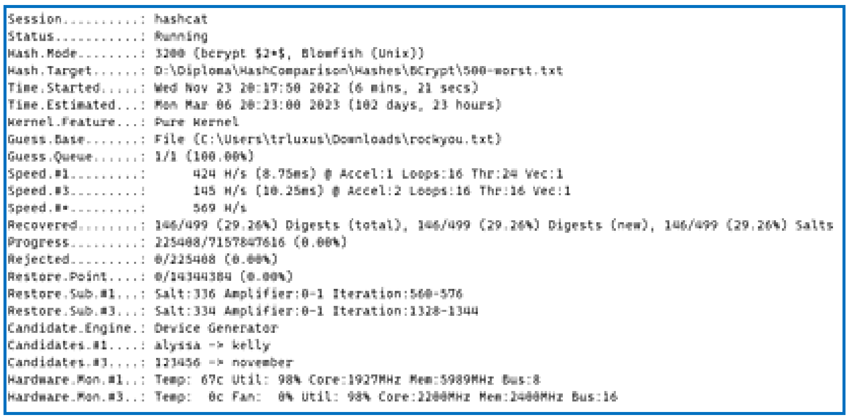
Fig. 12. The most common 10,000 passwords attack by the method of exhaustive search



Number of Characters	Numbers Only	Lowercase Letters	Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters, Symbols
4	Instantly	Instantly	2 secs	4 secs	4 secs
5	Instantly	3 secs	2 mins	4 mins	12 mins
6	Instantly	1 min	1 hour	4 hours	15 hours
7	3 secs	35 mins	3 days	2 weeks	2 months
8	26 secs	15 hours	5 months	2 years	10 years
9	4 mins	2 weeks	23 years	100 years	800 years
10	44 mins	1 year	1k years	7k years	61k years
11	7 hours	31 years	63k years	435k years	5m years
12	3 days	800 years	3m years	27m years	363m years
13	1 month	21k years	170m years	2bn years	28bn years
14	10 months	539k years	9bn years	104bn years	2tn years
15	8 years	14m years	460bn years	6tn years	166tn years
16	84 years	365m years	24tn years	399 tn years	13 qdn years
17	800 years	9bn years	1qdn years	25 qdn years	983 qdn years
18	8k years	246bn years	65 qdn years	2 qntn yrs	76qntn years

Fig. 13. Time to get a PBKDF2 SHA256 protected password using an RTX 3090 [25]

The slow calculation speed on the GPU is due to the fact that Bcrypt requires more than 4 kilobytes of memory for maximum speed, while the RTX-2080Ti only has 1 kilobyte. So, the current protection thanks to the memory from graphics processors is enough, but this parameter is unchanged and over time there may be devices for which this will not be an obstacle.



```
Session..........: hashcat
Status...........: Running
Hash.Mode........: 3200 (bcrypt $2*$, Blowfish (Unix))
Hash.Target......: D:\Diploma\HashComparison\Hashes\BCrypt\500-worst.txt
Time.Started.....: Wed Nov 23 20:17:50 2022 (6 mins, 21 secs)
Time.Estimated...: Mon Mar 06 20:23:00 2023 (102 days, 23 hours)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (C:\Users\trluxus\Downloads\rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:      424 H/s (8.75ms) @ Accel:1 Loops:16 Thr:24 Vec:1
Speed.#3.........:      145 H/s (10.25ms) @ Accel:2 Loops:16 Thr:16 Vec:1
Speed.#*.........:      569 H/s
Recovered........: 146/499 (29.26%) Digests (total), 146/499 (29.26%) Digests (new), 146/499 (29.26%) Salts
Progress.........: 225408/7157847616 (0.00%)
Rejected.........: 0/225408 (0.00%)
Restore.Point....: 0/14344384 (0.00%)
Restore.Sub.#1...: Salt:336 Amplifier:0-1 Iteration:560-576
Restore.Sub.#3...: Salt:334 Amplifier:0-1 Iteration:1328-1344
Candidate.Engine.: Device Generator
Candidates.#1....: alyssa -> kelly
Candidates.#3....: 123456 -> november
Hardware.Mon.#1..: Temp: 67c Util: 98% Core:1927MHz Mem:5989MHz Bus:8
Hardware.Mon.#3..: Temp:  0c Fan:  0% Util: 98% Core:2200MHz Mem:2400MHz Bus:16
```

Fig. 14. The worst 500 passwords dictionary attack

Table 2 – **Calculating Bcrypt hash values**

Work factor	FPGAs: hashes per second	GPU: hashes per second
05	25,200	28,000
07	6,300	7,000
08	3,150	3,500
10	788	875
11	394	438
12	197	219
14	98	109

John The Ripper password security auditing and password recovery tool supports FPGA boards. They are efficient enough to run 124 optimized Bcrypt cores on an FPGA. This results in Bcrypt's high hash rate, higher than the hash rate of the latest generation of high-end GPUs (Table 3).

Table 3 – **Calculation of hash values of Bcrypt using FPGA**

Work factor	FPGAs: hashes per second	GPU: hashes per second
05	120k	28k

So, Bcrypt is a protected algorithm against modern threats, but not the ability to configure protection due to memory creates risks when using ASIC and FPGA.

Scrypt is an extension of the ideas of Bcrypt and adds the ability to configure the amount of memory required. But on SP.NET there are no popular implementations that can be recommended for use in real applications. According to research, the cost of calculating a password using Scrypt is several times more expensive than Bcrypt and several times more expensive than PBKDF2 [23].

Argon2 is the winning algorithm for the 2015 Password Hashing Contest, the goal of the contest is to select one or more password hashing functions that can be recognized as a recommended standard.

According to research by RedHat, cracking the eight-character passphrase used to unlock an encrypted volume in about two seconds on a Raspberry PI could require up to 1,085 NVIDIA Tesla P100 GPUs, costing about $120 million [24].

4. Result Analysis

Below are the threats against which each of the algorithms provides protection.

It can be seen from Table 4 that Argon2 and Scrypt provide protection against the most threats, but Scrypt does not have an implementation that can be recommended as secure. Bcrypt does not have the ability to configure the required memory, but the amount used is enough to protect against GPU-based attacks.

PBKDF2 provides protection against attacks using the method of full traversal and the use of rainbow tables, which is the minimum protection required. MD5 and SHA256 do not provide protection against modern threats.

Table 4 – **Defense analysis**

Metrics	Rainbow table attack (presence of salt)	Overrun attack (slow function)	GPU protection	FPGA/ ASIC protection	Implementation security
MD5	No	No	No	No	Yes
SHA256	No	No	No	No	Yes
PBKDF2	Yes	Yes	No	No	Yes
Bcrypt	Yes	Yes	Yes	No	Yes
Scrypt	Yes	Yes	Yes	Yes	No
Argon2	Yes	Yes	Yes	Yes	Yes

Conclusion and Future Work

Password protection on SP .NET, which is widespread in enterprise and web applications, is an important issue. Hashing algorithms built into SP are characterized by high speed and sufficient protection against threats that existed at the time of their creation. But the increasing power of CSs and the proliferation of GPUs, ASICs, and FPGAs make CS that use these algorithms vulnerable. More modern alternative algorithms have been created to protect against these threats.

As a result of the performed analysis, Argon2 is the best algorithm for use on SP .NET. This algorithm provides the most reliable protection at optimal speed, the possibility of memory configuration provides protection from ASIC and GPU. To support legacy applications, you can use Bcrypt, the main threat of which is ASIC. PBKDF2 with a sufficiently large number of iterations is able to provide fairly reliable protection, it should be used for applications.

Due to the lack of popular implementations, the Scrypt algorithm is not recommended.

Further work in this direction may include the creation of CS and software for full local analysis of all algorithms.

References

1. Dotsenko, N., Chumachenko, I., Galkin, A., Kuchuk, H. and Chumachenko, D. (2023), "Modeling the Transformation of Configuration Management Processes in a Multi-Project Environment", *Sustainability (Switzerland)*, Vol. 15(19), 14308, doi: https://doi.org/10.3390/su151914308
2. Yaloveha, V., Orlova, T., Podorozhniak, A., Kuchuk, H. and Gorbulik, V. (2023), "Modern Applications of High-Resolution Multispectral EuroPlanet Dataset", *2023 IEEE 4th KhPI Week on Advanced Technology*, KhPI Week 2023 - Conference Proceedings, doi: https://doi.org/10.1109/KhPIWeek61412.2023.10312851
3. Pittalia, Prashant P. (2019), "A comparative study of hash algorithms in cryptography", *International Journal of Computer Science and Mobile Computing*, vol. 8, is. 6, pp. 147–152, available at: https://ijcsmc.com/docs/papers/June2019/V8I6201928.pdf
4. Datsenko, S., and Kuchuk, H. (2023), "Biometric authentication utilizing convolutional neural networks", Advanced Information Systems, vol. 7, no. 2, pp. 67–73, doi: https://doi.org/10.20998/2522-9052.2023.2.12
5. Bowne, S. (2018), *Hands-On Cryptography with Python: Leverage the power of Python to encrypt and decrypt data*, Packt Publishing Ltd, 100 p., available at: https://github.com/PacktPublishing/Hands-On-Cryptography-with-Python
6. Rezanov, B. And Kuchuk, H. (2022), Fast Two-Factor Authentication Method in Systems With a Centralized User's Database, *2022 IEEE 4th KhPI Week on Advanced Technology*, KhPI Week 2022 - Conference Proceedings, 03-07 October 2022, Code 183771, doi: https://doi.org/10.1109/KhPIWeek57572.2022.9916491
7. Singh, A., Jain, M. and Goyal, S. (2022), "A 3-Lock based Password Hashing Algorithm", *2022 IEEE Conference on Interdisciplinary Approaches in Technology and Management for Social Innovation*, IATMSI 2022, doi: https://doi.org/10.1109/IATMSI56455.2022.10119411
8. Pise, A.A., Singh, S., Hemachandran, K., Pise, G.S. and Imuede, J. (2024), "Utilizing Asymmetric Cryptography and Advanced Hashing Algorithms for Securing Communication Channels in IoT Networks Against Cyber Espionage", *Journal of Cybersecurity and Information Management*, vol. 13(1), pp. 46–59, doi: https://doi.org/10.54216/JCIM.130105
9. Kuchuk, N., Mozhaiev, O., Semenov, S., Haichenko, A., Kuchuk, H., Tiulieniev, S., Mozhaiev, M., Davydov, V., Brusakova, O. and Gnusov, Y. (2023), "Devising a method for balancing the load on a territorially distributed foggy environment", Eastern-European Journal of Enterprise Technologies, vol. 1(4 (121), pp. 48–55, doi: https://doi.org/10.15587/1729-4061.2023.274177
10. Menezes, A.J., van Oorschot, P.C. and Vanstone, S.A. (1997), *Handbook of Applied Cryptography*, 1st ed., CRC Press, doi: https://doi.org/10.1201/9780429466335
11. Semenov, S., Zhang, M., Mozhaiev, O., Onishchenko, Y. and Kuchuk, H. (2023), "Construction of a model of steganographic embedding of the UAV identifier into ADS-B data", *Eastern-European Journal of Enterprise Technologies*, vol. 5(4(125)), pp. 6–16, doi: https://doi.org/10.15587/1729-4061.2023.288178
12. Catalin, C. (2019), "A quarter of major CMSs use outdated MD5 as the default password hashing scheme", *ZDNet*, available at: https://www.zdnet.com/article/a-quarter-of-major-cmss-use-outdated-md5-as-the-default-password-hashing-scheme
13. (1995), *FIPS Publication 180-1: Secure Hash Standard*, National Institute of Standards and Technology (NIST), available at: https://csrc.nist.gov/pubs/fips/180-1/final
14. Bai, E., Jiang, X.-Q. and Wu, Y. (2022), "Memory-Saving and High-Speed Privacy Amplification Algorithm Using LFSR-Based Hash Function for Key Generation", *Electronics (Switzerland)*, vol. 11(3), 377, doi: https://doi.org/10.3390/electronics11030377
15. (2002), *FIPS Publication 180-2: Secure Hash Standard*, National Institute of Standards and Technology (NIST), available at: https://csrc.nist.gov/files/pubs/fips/180-2/final/docs/fips180-2.pdf
16. Haunts, S. (2019), "Safely Storing Passwords", *Applied Cryptography in .NET and Azure Key Vault*, Apress Berkeley, CA, Berkeley, doi: https://doi.org/10.1007/978-1-4842-4375-6_5
17. Tyagi, K., Yadav, S. K. and Singh, M. (2021), "Novel cryptographic approach to enhance cloud data security", *Journal of Physics: Conference Series*, vol. 1998, no. 1: 3rd International Conference on Smart and Intelligent Learning for Information Optimization, 9-10 July 2021, Hyderabad, India, IOP Publi, doi: https://doi.org/10.1088/1742-6596/1998/1/012022
18. Alwen, J., Chen, B., Pietrzak, K., Reyzin, L. and Tessaro, S. (2017), "Scrypt Is Maximally Memory-Hard", *Advances in Cryptology – EUROCRYPT 2017*, vol 10212, Springer, Cham, doi: https://doi.org/10.1007/978-3-319-56617-7_2
19. Prasol, I. and Yeroshenko, O. (2023), "Modeling and estimating the model adequacy in muscle tissue electrical stimulator designing", Radioelectronic and Computer Systems, vol. 2(106), pp. 18–26,doi: https://doi.org/10.32620/reks.2023.2.02
20. Fedorchenko, V., Prasol, I. and Yeroshenko, O. (2021), "Information Technology For Identification Of Electric Stimulating Effects Parameters", *CEUR Workshop Proceedings*, pp. 189-195, available at: https://ceur-ws.org/Vol-3200/paper26.pdf
21. Petrovska, I. and Kuchuk, H. (2023), "Adaptive resource allocation method for data processing and security in cloud environment", *Advanced Information Systems*, vol. *7*(3), pp. 67–73, doi: https://doi.org/10.20998/2522-9052.2023.3.10
22. Kuchuk, H. and Malokhvii, E. (2024), "Integration of IOT with Cloud, Fog, and Edge Computing: A Review", *Advanced Information Systems*, vol. *8*(2), pp. 65–78, doi: https://doi.org/10.20998/2522-9052.2024.2.08
23. (2023), *Are Your Passwords in the Green*?, available at: https://www.hivesystems.io/blog/are-your-passwords-in-the-green
24. (2023), *Bcrypt password cracking extremely slow? Not if you are using hundreds of FPGAs*!, available at: https://scatteredsecrets.medium.com/bcrypt-password-cracking-extremely-slow-not-if-you-are-using-hundreds-of-fpgas-7ae42e3272f6
25. (2023), *What Is a Hash Function in Cryptography*?, A Beginner's Guide,. available at: https://www.thesslstore.com/blog/what-is-a-hash-function-in-cryptography-a-beginners-guide/

Received (Надійшла) 13.07.2024
Accepted for publication (Прийнята до друку) 16.10.2024

Відомості про авторів/ About the Authors

Федорченко Володимир Миколайович – кандидат технічних наук, доцент, доцент кафедри електронних обчислювальних машин, Харківський національний університет радіоелектроніки, Харків, Україна;
Volodymyr Fedorchenko – PhD in Engineering, Associate Professor, Associate Professor of the Department of Electronic Computers, Kharkiv National University of Radio Electronics, Kharkiv, Ukraine;
e-mail: volodymyr.fedorchenko@nure.ua; ORCID Author ID: https://orcid.org/0000-0001-7359-1460;
Scopus ID: https://www.scopus.com/authid/detail.uri?authorId=57716883800.

Єрошенко Ольга Артурівна – асистент кафедри електронних обчислювальних машин, Харківський національний університет радіоелектроніки, Харків, Україна;
Olha Yeroshenko – Assistant of the Department of Electronic Computers, Kharkiv National University of Radio Electronics, Kharkiv, Ukraine;
e-mail: olha.yeroshenko@nure.ua; ORCID Author ID: https://orcid.org/0000-0001-6221-7158;
Scopus ID: https://www.scopus.com/authid/detail.uri?authorId=57808290700.

Шматко Олександр Віталійович – кандидат технічних наук, доцент, доцент кафедри програмної інженерії та інтелектуальних технологій управління, Національний технічний університет "ХПІ", Харків, Україна;
Oleksandr Shmatko – PhD in Engineering, Associate Professor, Associate Professor of the Department of Software Engineering and Intelligent Management Technologies, National Technical University "KhPI", Kharkiv, Ukraine;
e-mail: oleksandr.shmatko@khpi.edu.ua; ORCID Author ID: https://orcid.org/0000-0002-2426-900X;
Scopus ID: https://www.scopus.com/authid/detail.uri?authorId=6602623478.

Коломійцев Олексій Володимирович –доктор технічних наук, професор, професор кафедри комп'ютерної інженерії та програмування Національного технічного університету «Харківський політехнічний інститут», Харків, Україна;
Oleksii Kolomiitsev – Doctor of Technical Sciences, Professor, Professor of Computer Engineering and Programming Department, National Technical University «Kharkiv Polytechnic Institute», Kharkiv, Ukraine;
e-mail: alexus_k@ukr.net; ORCID ID: https://orcid.org/0000-0001-8228-8404;
Scopus ID: https://www.scopus.com/authid/detail.uri?authorId=57211278112.

Омаров Мурад Анвер огли – доктор технічних наук, професор, професор кафедри комп'ютерно-інтегрованих технологій, автоматизації та робототехніки, проректор з міжнародного співробітництва, Харківський національний університет радіоелектроніки, Харків, Україна;
Murad Omarov – Doctor of Technical Sciences, Professor, Professor of the Department of Computer-Integrated Technologies, Automation and Robotics, Vice-Rector on International Cooperation, Kharkiv National University of Radio Electronics, Kharkiv, Ukraine;
e-mail: murad.omarov@nure.ua; ORCID Author ID: http://orcid.org/0000-0003-4842-4972;
Scopus ID: https://www.scopus.com/authid/detail.uri?authorId=55659255500.

Методи та алгоритми хешування паролів на платформі .NET

В. М. Федорченко, О. А. Єрошенко, О. В. Шматко, О. В. Коломійцев, М. А. Омаров

Анотація. Веб-додатки, які широко використовуються для надання послуг та збору інформації, стали основною мішенню для зловмисників, особливо з появою державних сервісів, що обробляють конфіденційні дані. Програмна платформа .NET, популярна для розробки веб-додатків, включає вбудовані алгоритми хешування (HA) та функції генерації ключів (KDF) для захисту паролів. Однак вони були розроблені понад два десятиліття тому для різних рівнів загроз. Більш сучасні альтернативи, такі як Bcrypt, Scrypt та Argon2, пропонують покращений захист від сучасних атак з використанням GPU, ASIC та FPGA, але потребують стороннього впровадження. Враховуючи критичну роль захисту паролів у захисті інформації користувача, це дослідження вивчає ефективність різних механізмів хешування на платформі .NET, що є нагальною потребою у забезпеченні безпеки сучасних веб-додатків. **Предметом** вивчення в статті є особливості алгоритмів хешування, вбудованих та доступних в бібліотеках програмної платформи .NET для захисту паролів, як основного аспекту аутентифікації користувачів. **Метою** роботи є порівняння та аналіз алгоритмів хешування, вбудованих та доступних в бібліотеках програмної платформи .NET для захисту паролів, як основного аспекту аутентифікації користувачів. проведено аналіз та порівняння алгоритмів хешування, доступних на програмній платформі .NET. **Завдання:** розглянути вбудовані алгоритми, такі як MD5, SHA та PBKDF2, а також сторонні реалізації сучасних функцій виведення ключів, таких як Bcrypt, Scrypt та Argon2 та дослідити їх швидкодію та криптостійкість. Використані **методи:** включали вимірювання швидкості хешування для різних наборів паролів та аналіз стійкості до атак за допомогою таких інструментів, як Hashcat, та даних незалежних досліджень з безпеки. **Результати** показують, що хоча вбудовані алгоритми, такі як MD5 та SHA256, є швидкими, вони не забезпечують захисту від сучасних загроз, таких як атаки з використанням райдужних таблиць та спроби повного перебору з прискоренням на GPU. PBKDF2, який є стандартним в ASP.NET Core Identity, забезпечує кращу безпеку, але вразливий до атак з використанням спеціалізованого обладнання. Серед сучасних алгоритмів Argon2 продемонстрував найкращий баланс безпеки та продуктивності, забезпечуючи захист від атак на основі GPU, ASIC та FPGA. **Висновки.** Дослідження дійшло висновку, що Argon2 є рекомендованим алгоритмом для хешування паролів на платформі .NET, а Bcrypt є підходящою альтернативою для застарілих додатків. PBKDF2 з високою кількістю ітерацій все ще може забезпечити надійний захист. **Перспективним напрямком** подальших досліджень може бути визначення можливості використання сучасних функцій виведення ключів з інтенсивним використанням пам'яті для підвищення безпеки паролів у додатках .NET.

Ключові слова: хеш-функція; алгоритм хешування; функція генерації ключів; комп'ютерні системи.