STUDIA INFORMATICA 2008
Volume 29 Number 4B (81)

Wiktor A. FRYZE, Ireneusz J. JÓŹWIAK, Andrzej M. KAŁUŻA
Wroclaw University of Technology, Institute of Applied Computer Science

HASHING ALGORITHMS – A WAY TO SECURE YOUR APPLICATION

Summary. This article presents one of the mechanisms of ensuring security in applications designed in Java programming language – hashing algorithms. It describes selected elements of Java Cryptographic Architecture library. Chosen algorithms were scrutinized in terms of the security and how fast they work.

Keywords: algorithm, hashing, java, security

ALGORYTMY HASZUJĄCE – SPOSOBY OCHRONY APLIKACJI

Streszczenie. W artykule został przedstawiony jeden z mechanizmów zapewniania bezpieczeństwa w aplikacjach wytwarzanych w środowisku Java – algorytmy haszujące. Zostały również opisane wybrane elementy biblioteki Java Cryptographic Architecture. Wybrane algorytmy zostały przeanalizowane pod kątem zarówno bezpieczeństwa, jak i szybkości działania.

Słowa kluczowe: java, bezpieczeństwo, haszowanie, algorytm

1. Introduction

Cryptography is a very significant discipline of science nowadays [1, 2]. Along with widespread and easy access to the Internet, the role of security in computer systems becomes especially important. That is why the contemporary cryptography is the discipline from the borderland of computer science and mathematics. It is applied in a great deal of solutions, ensuring security of computer passwords, cash cards, foreign trade. The programming languages' designers do their best to meet the expectations and needs of the market, and they implements many mechanisms ensuring relatively high level of security, both for developers and final users of designed system.

One of the most popular and most rapidly growing programming language is Java. The main reason behind the achieved success was undoubtedly the high level of security of the software designed in Java and its portability, obtained thanks to usage of its own virtual machine. The Java Platform gives us three basic mechanisms, ensuring the security [3]. These are:

- the features of the Java programming languages itself (for example the lack of multiple inheritance and elimination of pointers' arithmetic),
- control of access mechanism, supervising how the code works (access to network, access to files),
- code's signatures, which is available thanks to usage of cryptography.

All security mechanisms are available in the standard library JCA (Java Cryptography Architecture), which since the 1.5 version includes also packages of library JCE (Java Cryptography Extension) [4, 5]. JCA gives the programmer possibility of creating electronic certificates and signatures, coding data through generating keys, using the algorithms like Blowfish, DES, RSA and so on, mechanisms of authorization and authentication, secure pseudorandom numbers generators, and creating the hash functions, using the hashing algorithms.

The subject of further analysis will be the unilateral hashing algorithms, which generate the hash codes with specified length, relying on any sequence of data. Each hashing function should generate unique output sequence of data for two different input data sequence. Otherwise our algorithm is not safe. These functions characterize various computational complexity and the time of operation. Java gives us the following hash functions: MD2, MD5, SHA-0, SHA-1, SHA-256, SHA-384, SHA-512.

Using these algorithms in our own code is very easy. We only need to create an object of MessageDigest class. The procedure of creating of object is:

```
MessageDigest digest = MessageDigets.getInstance(algorithm)
```

The static method *getInstance* is evoked, which gets a name of used hash function as a parameter. Later on, to calculate the hash code, we need to pass the chart of bytes, being the input data sentence to this object. We can do it in two ways:

- code's signatures, which is available thanks to usage of cryptography.using the method update() of MessageDigest class, where the input data sequence is one of the parameters. This method is especially useful, if the length of input sequence is unknown, as it can be used repeatedly, and each time the next bytes of input sequence are added to the buffer, which is stored in digest object,
- through direct passing of the input data sequence as a parameter of digest() method:

```
digest.digest(input)
```

Method *digest()* returns coded sequence of bytes, that is the hash code, which later on can be formatted from binary code into the readable sequence of marks.

2. The comparison of efficiency of hashing codes available in Java Platform

Cryptography Hashing algorithms characterize the various time of execution. The conducted experiment let us compare the times of hashing files of varied size, using all hashing functions available in Java [6]. The analysis of them let us find out the general relationship between the time of generating a hash code and the size of input sequence. The measured value is the execution time of digest() method, which gets a chart of bytes as a parameter. So all auxiliary operations connected with gaining the content of the file took place beyond the operation block, what let us measure the time of generating a hash code exclusively.

Unilateral hashing functions are often used to secure the files. If the user changes the content of the file, the value of hashing function changes automatically. So the hash code lets the user detect undesirable changes of the file, made by the user not entitled to do so.

Table 1

Relationship between execution time of algorithm and the size of input file

File size (B)	File size (KB)	MD2 (ms)	MD5 (ms)	SHA-0 (ms)	SHA-1 (ms)	SHA-256 (ms)	SHA-384 (ms)	SHA-512 (ms)
334	0,33	0	0	0	0	2	0	0
19456	19,00	8	3	2	2	0	2	2
48961	47,81	19	0	2	2	3	3	3
95678	93,44	39	0	3	2	5	8	8
217271	212,18	86	2	6	5	9	17	19
393259	384,04	156	3	11	11	19	31	31
671454	655,72	267	5	19	19	30	59	55
1530776	1494,90	606	14	42	42	67	125	123
2688973	2625,95	1095	23	75	81	119	222	220
4847718	4734,10	1925	42	136	142	214	430	395
8056996	7868,16	3241	74	233	238	355	658	658
11056402	10797,27	4416	103	322	319	483	898	902
12748800	12450,00	5113	120	370	355	555	1036	1045
26573008	25950,20	10653	252	781	738	1198	2163	2167
49221632	48068,00	19727	442	1378	1403	2189	4011	4023

The Table 1 presents the times of generating hash codes for fifteen files of various size. The times shown in Table 1 are the average values computed on the basis of 10 independent measurements.

The Fig. 1 presents the direct comparison of these algorithms in terms of execution time. There is a relationship between the times of execution, on average MD5 is three times faster than SHA-1. The next comparison will concern the algorithms of SHA class exclusively (Fig. 2). The chart doesn't include the information about SHA-0 function, which actually is already outdated. However its execution time is very similar to SHA-1.

After having analysing the results, we come to the following conclusions: the fastest function is the SHA-1, SHA-256 computations take about 1.5 times more time than SHA-1 and SHA-384 and SHA-512 are executed in a similar time and are approximately three times slower than SHA-1.

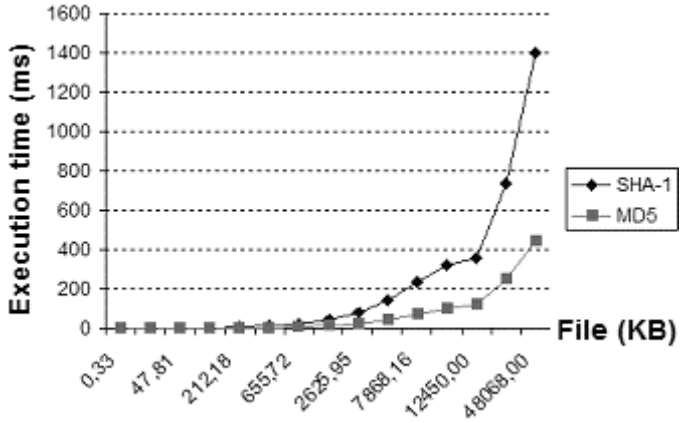

Fig. 1. Relationship between execution time of algorithm and the size of input file for SHA-1 and MD5 algorithms

Rys. 1. Wykres zależności czasu wykonania algorytmów od wielkości pliku wejściowego dla agorytmów SHA-1 i MD5

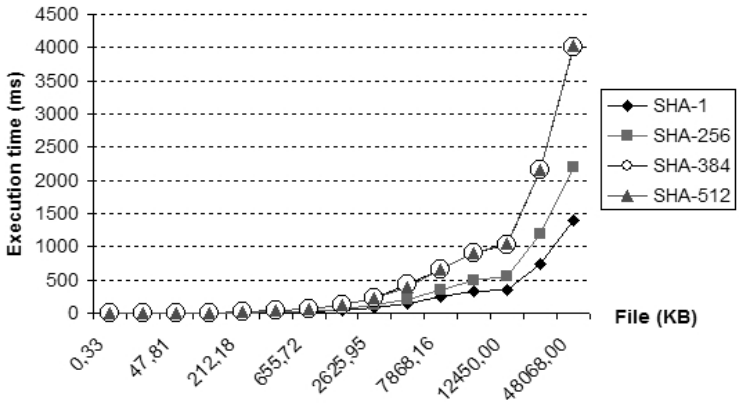

Fig. 2. The relationship between execution time and the size of input file for algorithms of SHA class

Rys. 2. Wykres zależności czasu wykonania algorytmów od wielkości pliku wejściowego dla algorytmów rodziny SHA

Table 2

The comparison of average processing speed for hashing algorithms

Algorithm	Average speed (B/ms)	Average speed (KB/ms)	Average speed (MB/s)	Standard devation (B/ms)	Standard devation (%)
MD2	2503,16	2,44	2,39	21,28	0,85
MD5	114087,19	111,41	108,80	9876,27	8,66
SHA-0	35305,78	34,48	33,67	820,92	2,33
SHA-1	35803,85	34,69	34,15	2727,90	7,62
SHA-256	22594,75	22,07	21,55	807,76	3,57
SHA-384	12172,64	11,89	11,61	462,84	3,80
SHA-512	12224,52	11,94	11,66	298,43	2,44

So we can draw more general conclusions, which will concern mutual relations between all hashing algorithms. For files bigger than 200 KB the relationship between execution time and the size of input data stabilizes (the quotient of the size of file and the execution time is constant). The Table 2 presents the average speed of processing input data for each of these algorithms. Next we compare the execution time of all algorithms (Table 3 and Fig. 3). The basic unit assumed is the execution time of SHA-1 [8].

Table 3

The comparison of processing times of all hashing algorithms

Algorithm	Processing speed
MD2	14,30
MD5	0,31
SHA-0	1,01
SHA-1	1,00
SHA-256	1,58
SHA-384	2,94
SHA-512	2,95

Fig. 3. The comparison of execution time for all hashing algorithms (SHA-1 algorithm represents the unit value here)

Rys. 3. Zestawienie zależności czasów wykonania wszystkich algorytmów haszujących od rozmiaru pliku wejściowego

Definitely the slowest algorithm is MD2, that's why it is not used to hash input sequences of big size. The fastest one is MD5. If we try to find something about the history of these algorithms we noticed that Ronald Rivest, the inventor of MD5,was also involved with the review of SHA-1. Investigating papers that compare these two hash functions it is not difficult to notice that they have very similar design principles. SHA-1 uses five rounds to produce five 32 bit blocks, whereas MD5 uses four rounds to produce four 32 bit blocks. The new feature of SHA-1 is an expansion step before starting the first round. It is a simple linear feedback shift register expansion that mixes all the bits of the input block. More rounds of operations confirm the results of researching and makes MD5 the fastest hashing function. But the next question appears here – is the speed of generating the hash code reflected in its safety, are these algorithms safe?

3. MD algorithms

MD Algorithms (Message-Digest Algorithms) are the set of the most often used hashing functions for data ciphering. Thanks to them we have the possibility of generating 128-byte hash code for any data sequence. MD5 (belonging to MD algorithms) is one of the algorithms designed by the co-author or RSA, professor Ronald Rivest from z Massachusetts Institute of Technology in 1991. Here's how the algorithm works:

- the bit with value 1 is added to the input message, being the object being hashed,
- to the resulted sequence we add as many zeros as we need to let the whole sequence consist of 512-bit blocks (the last block has 448 bits),
- the 64-bit message, representing the size of input message is added to the resulted sequence,
- the initial state is set to 0123456789abcdeffedcba9876543210,
- on each of 512-bit block the function changing the state is run,
- after processing all of the blocks, the final state is returned, which is the hashcode of the message from the input.

The function of state change consists of 64 steps (4 cycles). The state is interpreted as four 32-bit numbers, to which in each step are added:

- one of sixteen 32-bit parts of input block,
- a constant dependent on the number of the block currently being executed,
- the simple boolean function of three remaining numbers.

Later each number is moved with some number of bits, dependent on the current step, and the number of the rest of numbers is added to it.

The example hash code generated for the sequence which consists of the first name and the last name of the algorithm's author is:

```
MD5(„Ronald Rivest”) = 6c08f52fe3d4dd0ff5c64468c8959043
```

What is important and characteristic for all hashing functions, even minor change of input sequence results in completely different hash code generated. In the following example the space has been added to the last name:

```
MD5(„Ronald Rivest ”) = 7385e7c3c5f104554469b4a0df1110a6
```

The next characteristic of the hashing functions of MD group is generating hash codes of the same length (128-bit number coded in hexadecimal way) regardless of how big the input data is. For example the hashing function for the 5,54 MB file looks like this:

```
MD5(file) = f4a87acc3d5897893d2de67d830c9e36
```

The predecessor of MD5 algorithm was MD4, which as a result of Hans Dobbertin's researches was considered as not very safe (the possibility of causing collisions on the common PC in the time not exceeding several seconds). Contrary to its successor the hashing operation was done in 3 cycles, 16 steps each. MD4 algorithm gained its popularity mainly thanks to the speed, at which it conducted computations. In its newer version MD5 the great records in speed were resigned in order to enhance its safety. The first version of the MD algorithm is MD2 designed in 1989 for 8-bit computer. In 2004 the official attack on MD2 with complexity of 2104 was presented (Mulle, 2004). Since that moment the algorithm in second version stopped to be used in the purpose of ensuring data security [7].

4. SHA algorithms

When the frequent attempts of trying to undermine the reliability of MD algorithms appeared, the new group of algorithms was created, called SHA (Secure Hash Algorithms) [1]. They were designed by National Security Agency, and next published by National Institute of Standards and Technology.

Hashing function SHA-1 (like MD algorithms and relying on similar operations) let compute the message digest. The size of output message changed (160 bits), what on the considerable scale enhanced the safety of data (the protection from brute-force attacks), extending the time of computation.

The predecessor of SHA-1 was SHA, in which right after introduction, the very serious fault was detected. As a result of necessary modifications (in one position the 1-bit shift was added) the SHA-1 was created. In next years from 2001 four next versions appeared, know as

SHA-2. They consist of SHA-224, SHA-256, SHA-384, SHA-512. In 2004 the successful attempts of generating collisions in the algorithm resembling SHA-1 in structure were announced, what forced the NIST organization to announce the statement, that until 2010 the SHA-1 will have not been used, and instead the new versions of SHA-2 will be applied.

5. Time of breaking the function

Very researching of average speed of hashing algorithms in B/ms units lets you get the average time to break this function. Usually hashing functions are used to cipher passwords which are indeed seta of characters. Let's say that user of system makes the password using characters from keyboard which includes 62 characters: 52 letters of English alphabet (26 small letters and 26 capital letters) and 10 figures. The Fig. 1 presents formula of number of all possible letter combinations to get correct password.

Presented way of breaking the function lies in generating hash-codes right up to the moment when generated code for input set is identical as ciphered set of letters. We can suppose that after generating R(N)/2 codes we can find ciphered set of letters that we look for. Besides it is important to know (according to Java specification) that one simple character is set on 2 bytes. The general formula of time to encipher the password with length of N characters is

$$Z(N) = \frac{1}{2} R(N) * \frac{N * 2}{S} = \frac{N}{S} 62^N$$

where S is the speed of algorithm.

Table 4 presents the comparison of breaking speed of various hashing algorithms (speed of generating all possible combinations of passwords) for passwords with various lengths.

Table 4

Comparison of breaking speed of various hashing algorithms

Algorithm	Braking time for N=5 [s]	Braking time for N=7 [s]	Braking time for N=9 [s]	Braking time for N=11 [s]
MD2	1,83E3	9,84E6	4,87E10	2,29E14
MD5	4,02E1	2,16E5	1,07E9	5.01E12
SHA-0	1,29E2	6,98E5	3,45E9	1,62E13
SHA-1	1,28E2	6,89E5	3,40E9	1,60E13
SHA-256	2,02E2	1,09E6	5,39E9	2,53E13
SHA-384	3,76E2	2,03E6	1.00E10	4,7E13
SHA-512	3,75E2	2,02E6	9,97E9	4,68E13

As we can see the faster the function is the easier it is to brake it. Certainly that statement is quite bold. The time of generating all possible hash-codes to find long passwords is quite long. For example it takes 34 years for MD5 function (which is the fastest one) to break the password of 9 characters.

6. The future of hashing algorithms

Very often we receive the information about consecutive attempts of attacks on hashing functions, and in the aftermath of this undermining its reliability. Such gossips, despite being considerable in terms of mathematical meaning, should not be the reason to arouse panic among the scientists researching this discipline of science. We should consider if it's not the right time for finding out the other way of coding data, the new standard of hashing.

One-way functions described in this chapter are used in many applications. They are applied to make date secret, generate digital signature and so on. Since the oldest version of SHA algorithm to its currently the most often used successor SHA-1, through MD functions, the safety rate of applying them constantly increases.

One-way hashing functions characterizes one important property. They are one-way. We can generate hashed value of any input data sequence. But it's impossible (in a reasonable period of time) reversing the input data, having only the generated hash code. Thinking about ideal world hashing functions should be free of collision. It should be impossible to find two different sets of data, for which hash codes are identical. Unfortunately it is not truth. It results directly from the fact that the number of all possible combinations of 160-bit information is finite according to opened set of data we would like to hash.

As it appeared algorithm SHA-1 is not faultless algorithm. The group of scientists (Xiaoyun Wang, Yiqun Lisa, Hongbo Yu; Shandong University in China) published their results of researches on the SHA-1 algorithm [8]. These informations present the generating a collision in a full SHA-1 in 269 operations and the collision in 58-round SHA-1 of 233 operations. We should also add that the second hashing function in terms of popularity MD5 was broken earlier on by the group of the same scientists. The result of breaking the full SHA-1 in 269 operations is surprisingly good, fortunately it is completely beyond the capabilities of personal computers. But it doesn't change the fact that the reputation of the SHA-1 safety was considerably undermined.

7. Conclusions

It can't be said that the most popular hashing algorithms (SHA-1 and MD5), used in the majority of applications requiring safety of data, became suddenly "dangerous". As the old saying of the scientists from National Security Agency states: "The attacks are more and more effective, never less". The techniques of attacks are going to be improved and probably one day the official braking of the SHA-1 algorithm in a reasonable period of time will be announced. The time has come, when we should introduce new hashing algorithms. The

designers of SHA have already have the ready solutions, but we should think, if the new standard of coding is not needed, instead of introducing next versions of hashing functions. The work of the most popular nowadays hashing functions is based on the algorithm used in MD4, which was broken in 1996. All these information should not evoke panic among people working on cryptography and data protection. But they should be a valuable hint and a warning that the progress nowadays in the field of computer science calls the safety of used hashing solutions into question.

REFERENCES

1. Bauer F. L.: Sekrety kryptografii. Helion, Gliwice 2002.
2. Ferguson N., Schneier B.: Kryptografia w praktyce. Helion, Gliwice 2004.
3. Hook D.: Kryptografia w Javie: od podstaw. Helion, Gliwice 2006.
4. Horstmann C.S., Cornell G.: Core Java 2. Wydanie 2 – Techniki zaawansowane. Rozdział 9, Helion, Gliwice 2005.
5. Oaks S.: Java a bezpieczeństwo. Wydawnictwo RM, Warszawa 2002.
6. Richardson W. C., Avondolio D., Vitale J., Schrager S.,Mitchell M.W., Scanlon J.: Professional Java JDK 5 Edition. Wrox, Indianapolis 2005.
7. Schwaha P., Heinzl R.: Zagrożenia związane ze stosowaniem algorytmu MD5. hakin9 Magazine 1/2005, Wydawnictwo Software, Warszawa 2005.
8. Wang X., Yin Y. L., Yu H.: Collision Search Attacks on SHA1. 2005.

Recenzent: Dr inż. Adam Świtoński

Wpłynęło do Redakcji 25 września 2008 r.

Omówienie

Bez względu na język programowania, używany podczas tworzenia systemu, bezpieczeństwo powstającej aplikacji jest niezwykle ważnym aspektem. Tworząc aplikację zawsze istnieje pewna grupa ludzi, których głównym zadaniem jest zapewnienie takiego bezpieczeństwa. Kiedy aplikacja jest gotowa do wdrożenia, użytkownicy oczekują użycia w niej pewnych mechanizmów zapewniających prywatność. Chcieliby mieć możliwość przechowywania pewnych informacji bez możliwości dostępu do nich ze strony innych użytkow-

ników. Dlatego jednym z głównych celów zakładanych podczas tworzenia oprogramowania jest użycie takich mechanizmów zapewniania bezpieczeństwa, które będą satysfakcjonujące dla przyszłych użytkowników. Z takimi zabiegami związane jest zastosowanie odpowiednich mechanizmów autentykacji oraz autoryzacji. Autentykacja jest procesem, w którym stwierdza się, czy dana osoba (użytkownik) jest rzeczywiście osobą, za którą się podaje. Autoryzacja natomiast jest procesem nadawania użytkownikowi uprawnień do pewnych operacji bądź informacji przechowywanych w systemie. Często użytkownik chciałby mieć pewność, że jego prywatne informacje nie były modyfikowane podczas jego nieobecności. Każdy z języków programowania ma własny mechanizm zapewniania bezpieczeństwa prywatnych danych. W tym artykule zostały zaprezentowane mechanizmy zapewniania bezpieczeństwa aplikacji tworzonych w języku programowania Java, zwane algorytmami haszującymi. Algorytm haszujący otrzymuje pewien zestaw danych (dowolnej długości) na wejściu i tworzy na podstawie jego zawartości pewien ciąg bitów, który zwraca na wyjściu. W artykule tym poruszony został sposób działania algorytmów jednokierunkowych (nie istnieje możliwość odtworzenia danych wejściowych na podstawie wiadomości wyjściowej). Opisane zostały dwie rodziny algorytmów, SHA oraz MD, a także wyniki przeprowadzonych badań z ich użyciem. Rezultaty zostały przedstawione na wykresach – porównanie prędkości działania algorytmów z obydwu rodzin oraz kolejnych wersji w każdej rodzinie. Badania polegały na podawaniu na wejściu danych różnego typu i różnej długości, a następnie obserwowaniu czasu tworzenia wiadomości wyjściowej. Artykuł opisuje również kolejne próby złamania algorytmów haszujących oraz wybrane elementy biblioteki Java Cryptographic Architecture, zawierającej przedstawione mechanizmy zapewniania bezpieczeństwa.

Addresses

Wiktor Adam FRYZE: Wroclaw University of Technology, Institute of Applied Computer Science, ul. Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland, wiktor.fryze@o2.pl.
Ireneusz Józef JÓŹWIAK: Wroclaw University of Technology, Institute of Applied Computer Science, ul. Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland, ireneusz.jozwiak@pwr.wroc.pl.
Andrzej Mateusz KAŁUŻA: Wroclaw University of Technology, Institute of Applied Computer Science, ul. Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland, andrzej.kaluza@gmail.com.