Modification of the perfect cipher for practical use

PETR VOBORNÍK
Department of Informatics
University of Hradec Králové
Rokitanského 62, Hradec Králové, 500 03
Czech Republic
petr.vobornik@uhk.cz   www.petrvobornik.cz

Abstract: - The principle of the perfect cipher has been known for almost a century. Unfortunately conditions that must be complied with its use still complicate its practical use. Principle of the perfect Vernam cipher will be summarized in this article. Pursuant of it and with the use of modern hash algorithms the modification of work with the encryption keys will be proposed so that the major factors that prevent the practical use of the cipher will be removed and most of their benefits will be preserved. The results of measuring the speed at which this cipher is able to work and its comparison with other ciphers will also be presented.

Key-Words: - Cipher, hash, communication, data transportation, one-time pad, Vernam, algorithm.

1 Introduction

In 1917 Gilbert Sandford Vernam improved the Vigenèr's cipher from 1586 [1]. He was inspired by the work of the German cryptologist Hermann from 1892 [2]. The system was based on a randomly generated one-time passwords (originally called "one-time pad"). Claude Elwood Shannon proved mathematically that the cipher is absolutely impenetrable in 1949 [3]. The perfect impenetrability can be achieved only when three strict conditions of reliability are observed:

1. The key must be perfectly random.
2. The key must be as long as the encrypted message.
3. The key must not be used repeatedly. [4]

When the conditions of confidence are complied, the cipher is completely safe against any attempt to break, including a brute force attack. If the correct key is not known, there is no way to decipher the message not even in an arbitrarily long period of time. It is possible to find a key that could convert encrypted data into readable text of the same length, but as more keys like that can be found, this data can in fact make arbitrary sense, and it is not possible to predict which of these interpretations was correct. [5]

The conditions that must be observed for using Vernam cipher makes its practical using considerably more complicated. In the following text therefore we will try to propose a modification of the cipher, which eliminates the drawbacks while keeping most of their benefits.

2 Modified interpretation the perfect cipher

Individual terms of reliability will be solved gradually through the modern practices and technologies.

2.1 The key must be perfectly random

To the work more effectively with Vernam cipher a function was used that was not available when the cipher was created and that is the hash. The hash is one-way (irreversible) computationally efficient function mapping binary strings of arbitrary length to strings of fixed length, it is called the hash-value.

The statistical tests of randomness of hash code generated by the algorithm SHA-1[1] according to [6] and [7] demonstrated that the generated bit sequence meets from a statistical point of view the conditions of random uniform distribution. Other hashing algorithms (such as MD5, SHA-256, SHA-512, etc.) should by definition have the same characteristics, which can be checked according to the procedures specified in [8] using the software which is described in [6].

[1] SHA-1 - Secure Hash Algorithm, returning hash code of 160 bits, which was designed by NIST for U.S. government applications. [8]

2.2 The key must be as long as the encrypted message

The key which satisfies the condition of randomness and it does not allow the re-calculation of the password it can be created by using the hash algorithm on any password. Its length however is predetermined to constant number of bits according to a specific algorithm applied. Nevertheless a key much longer than the hash code is needed.

The key of needed length can be created by using a multilevel hash. Its calculation is performed so that the hash of the original password is used to encrypt the first block of data while also serve as an input to generate a new hash code (hash level 2). It again encrypts the next block of data and a hash of the third level is generated from it as the key for the next block of data and so it continues until it covers the entire data message (see Fig. 1).

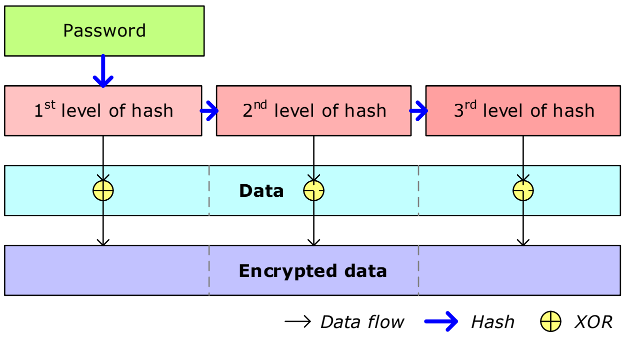

Fig. 1 – Diagram of data encryption by the XOR operations, where the key consists of a simple multi-level hash of the password

However this approach brings one major flaw. If an attacker knew some part of data, he would be able to decipher their other unknown parts. For example, when would he know the part of the data encrypted by second level of hash, he can then perform the XOR operation between these data and the encrypted data for acquiring a part of the key (hash level 2). He cannot calculate the first level of hash or password from it, but he can generate a third level of hash, then fourth level, etc. (see Fig. 2). Thanks to it, he is able to decrypt the data from the block whose content he knows (e.g., greeting or a document header) or he reveals by a dictionary attack.

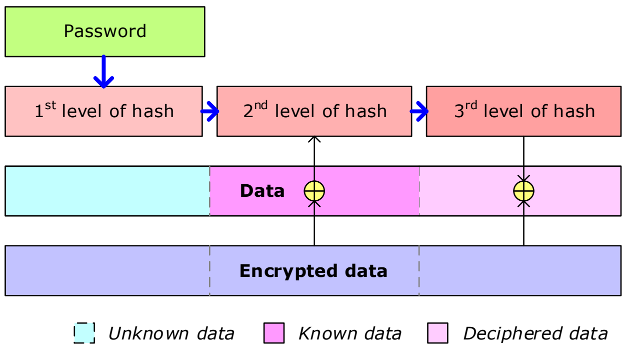

Fig. 2 – Diagram of deciphering the data encrypted by the key from a simple multi-level password hasht

The first level of hash was removed from the encoding process to eliminate this weakness. This block of the key cannot be determined nor with the knowledge of content whole message (data). The first level of hash is combined with each level of hashes by XOR operations (see Fig. 3).

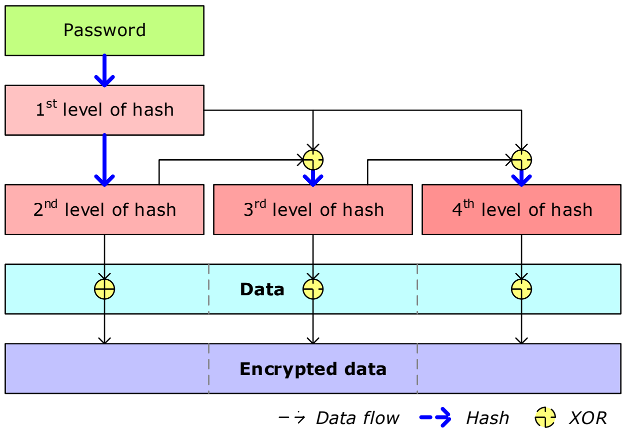

Fig. 3 – Diagram of data encryption using a key from the combined multi-level password hash

If the attacker knows a certain part of data, he can decipher the part of the key by which was encrypted this section, but he cannot determine the key for the next (and of course previous) block of data. He would need to know either the first level of password hash or the source of hash for the key of the next block of data. Both requirements would mean determination of the reverse hash, which is computationally impossible according to the basic definition of this function.

The only way to determine the unknown part of the data is "guess" the password or first level of hash. This mainly depends on the "strength" of chosen password, i.e. how it can withstand dictionary attack and brute force attacks. There are a number of techniques how to select easy-to-remember passwords (e.g. see [9]) which are resistant to these types of attacks (e.g. see [10]).

2.3 The key must not be used repeatedly

In order the key for data encryption was different every time, even if the password was still the same, we proceeded from the fact that for the complete change of the whole key (consisting of a multi-level hash) it is needed to change only a single bit of the password or first level of hash. In this type of change the description of this change can be a part of the message containing the encrypted data. For example this can be an additional text string that could be added to the password before calculating the first level of hash. This supplement of passwords is called "salt" and it is a good tool against dictionary attacks based on prepared dictionaries of hashes [11]. Its publication does not reduce the difficulty of deciphering the data because for the calculation of the key is still necessary to know the original part of the password.

With the addition of the salt the key for the data is completely different every time and if its part (or the whole key) in some previous data transmission is determined, the data transferred in the future has no reduced security, without requirement to change the password. It is only necessary to ensure that the salt is always different. This can be achieved for example using a generator of the GUID[2] values.

The salt can be transmitted as one parameter of the message in communication between two sides. But when the cipher is used for the long-term self-saved files the salt needs to be saved directly into the encrypted file, preferably right at the its beginning (see Fig. 4).

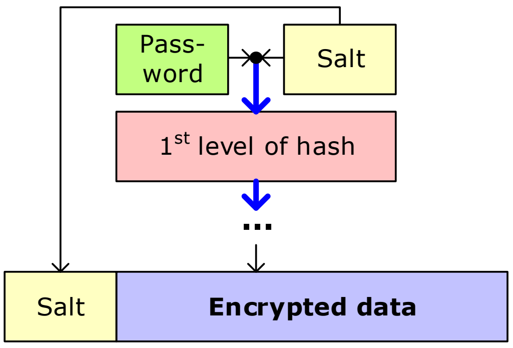

Fig. 4 – Diagram of data encryption using the salt

When the salt is saved at the beginning, the encryption and the decryption of the data can be proceed by the usual way as a block or a stream. Beginning of the data where the salt is stored must be read always and thus it is not possible to decrypt only certain sections of the data independently of the order. It also complicates the need ti recalculate the appropriate level of the hash.

3 Speed of the encryption

The proposed cipher algorithm is built directly on a specific key, which consists of multi-level hash, which is also in each level again combined with the first level of the hash. Calculating of the hash is of course not trivial computational operations, but a complicated algorithm that takes some computing time. It is obvious that this hash algorithm has the greatest share of the speed of encryption. Since the cipher can work with any hash algorithm, we tried to choose the fastest of them.

For this purpose, the following comparisons were performed (see Table 1). In this test there were used different hash algorithms (table rows) which calculated keys of given lengths (table column) as multi-level hash. The experiment was repeated 3 times and each measured time required for the calculation of each key was recorded. Of these three repeated measurements there was always recorded the shortest time (table cell), expressed in milliseconds. Basic features of each hash algorithm (irreversibility and randomness) has been taken as matter of course and further no tested.

The measurement were performed under identical conditions, i.e. on the same computer with the same running processes. Parameters of the tested machine were following: CPU 2,2GHz Core i7-3632QM, RAM 8GB, OS Windows 8 Pro 64bit. To calculate of keys was used hash algorithms implemented in the library HashLib3 version 2.0.1 in programming environment Microsoft .NET Framework 4.5, language C#. Researched cipher algorithm was implemented and subsequently tested in the same environment. The program ran only on a single thread.

[22] GUID - Globally Unique IDentifier. Randomly generated value with negligibly small probability that anyone would ever have been generated by two identical values.

[3] HashLib project, see http://hashlib.codeplex.com

Data size [MB]	1	5	10	20	30	50	100
SHA-3-256	218	1 084	2 169	4 341	6 517	10 865	21 693
SHA-3-512	110	550	1 088	2 182	3 271	5 440	10 911
SHA-1	86	431	857	1 715	2 576	4 314	8 603
MD5	140	406	809	1 619	2 427	4 067	8 122
SHA-2-256	70	352	704	1 407	2 113	3 519	7 050
SHA-2-512	63	319	638	1 285	1 925	3 199	6 401

Table 1 – Comparison of computation speed [ms] of multi-level hash of the various algorithms

The comparison in Table 1 shows that the fastest of the test hash algorithms is SHA-2 with a length of 512 bits (also called as SHA-512). This algorithm was then subsequently used for comparison test of the speed of existing encryption algorithms (see Table 2). This comparison was made in a similar test under the same conditions as compared to the speed of hash algorithms. Again it was only the calculation of values within the computer's memory without the need loading or saving from/to the hard disk. The data were processed in stream in blocks of size 5 kB.

Data size [MB]	1	5	10	20	30	50	100
AES	26	130	259	527	774	1 289	2 611
DES	32	159	314	629	949	1 582	3 158
RC2	38	188	374	748	1 121	1 864	3 742
AES-Managed	43	219	431	865	1 292	2 161	4 309
TripleDES	61	301	601	1 202	1 802	3 008	6 028
Tested cipher	**95**	**472**	**945**	**1 891**	**2 848**	**4 740**	**9 458**

Table 2 – Comparison of speed encryption [ms] of datasets of the various sizes and by the different algorithms

Comparison shows in Table 5 and in the graph in Fig. 5 that the proposed cipher algorithm is slower than other algorithms, but about 67% of the time spans calculation of the key, even by the fastest hash algorithm SHA-512. Using a faster hash algorithm would lead to a significant acceleration of the cipher.

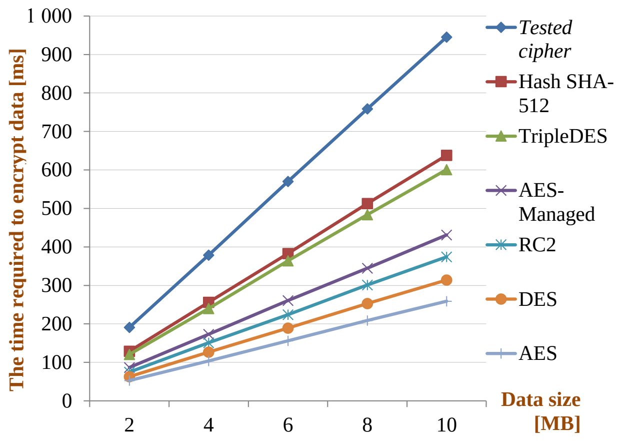

Fig. 5 – Comparison of speed encryption [ms] of datasets of the various sizes and by the different algorithms. Included is also the speed of generating the key using a hash algorithm SHA-512.

Other cipher algorithms had an advantage in that they have not been implemented in .NET controlled (managed) code, but they were built directly into the operating system. Here they are processed under the application layer through the Windows CryptoAPI without unnecessary overhead [12]. This difference is evident when comparing the speed of encryption by AES and AES-Managed, which in both cases is the same algorithm, but AES-Managed is implemented in .NET, as well as tested cipher. Other algorithms under the same conditions would be then also significantly slower, roughly in the same proportion (approximately 1.67x).

The speed of cipher is so limited for use in the classical application in real time such as on-line communication between two sides, where the speed of the connection is the one of the main parameters. Its use could be thus more in cases where the level of security has a higher priority than the time required to encrypt (e.g. archiving of confidential files).

4 Conclusion

The original principle of Vernam perfect cipher was outlined in the article. On the basis of it was proposed modifications to work with the keys, which previously very complicated its practical use. When combined with modern hash algorithms is possible to encrypt data by using mathematically proven of reverse incalculable procedures and also can be repeatedly used a "simple" password. So the strength of ciphers is always directly proportional to the strength of the chosen password.

Implementation of this procedure is a programmatically very simple. Speed of encryption and decryption of data mostly dependent on the speed of calculation of a hash code, i.e. on the selected hash algorithm. Slightly faster than the selected SHA-512 has been doing so Shabal-512 a candidate for the SHA-34 [13], or in a multi-threaded processing the winner of this competition Keccak [14].

The proposed cipher can be used due to its current lower speed to protect the transmission of data over a public network internet only in cases where does not matter a deceleration needed for data encryption. For encryption of archives and files for long-term storage, where is usually more important their safety than the time required to encrypt, this cipher can be used just now.

Acknowledgements:
This article is supported by the project Research of cipher algorithms, maintained on the Faculty of Science on the University of Hradec Králové.

References:

[1] Piper, F., Murphy, S., *Cryptography: A Very Short Introduction,* Oxford: Oxford University Press, 2002, ISBN 978-0192803153.

[2] Janeček, J., *Gentlemani nečtou cizí dopisy*, Brno: Books, 1998, ISBN 80-85914-90-5.

[3] Shannon, C. E., *Communication Theory of Secrecy Systems,* Bell System Technical Journal, 1949, 28, pp. 656–715.

[4] Hála, V., *Kvantová kryptografie,* Aldebaran Bulletin, 14/2005, vol. 4, ISSN 1214-1674, [Online], Available: http://aldebaran.cz/bulletin/2005_14_kry.php [12 Feb 2014].

[5] Voborník, P., *Téměř dokonalá šifra*, Matematika, fyzika, informatika, 2014, vol. 23, no. 1, pp. 54–69. ISSN 1210-1761.

[6] Rukhin, A., Soto, J., Nechvatal J, Smid, M. Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, A., Dray, J., Vo, S., Bassham, L., E., *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*, NIST Special Publications (800 Series), 2010, SP 800-22 Rev 1a, [Online], Available: http://csrc.nist.gov/publications/nistpubs/800-22-rev1a/SP800-22rev1a.pdf [25 Feb 2014].

[7] Pierre, L., Richard, S., *TestU01: A C Library for Empirical Testing of Random Number Generators*. Université de Montréal: ACM Trans, 2007, Math. softw. 33, 4, article 22, [Online], Available: http://dx.doi.org/10.1145/1268776 [10 Feb 2014].

[8] Menezes, A., J., Oorschot van, P., C., Vanstone, S., A., *Handbook of Applied Cryptography*, Boca Raton: CRC Press, 1996, ISBN 978-0-8493-8523-0.

[9] Strnadová, V., *Interpersonální komunikace*, Hradec Králové: Gaudeamus, 2011, 543 p., ISBN 978-80-7435-157-0.

[10] Hubálovský, Š., Musílek, M., *Počítačová bezpečnost ve výuce informatiky (Tvorba hesel a steganografie)*, Matematika-fyzika-informatika, Praha: Prometheus, 2010, vol. 20, November 2010, ISSN 1210-1761.

[11] Voborník, P., *Bezpečná autentizace aplikace klient-server v internetu pomocí povinně unikátních saltů*, Internet, bezpečnost a konkurenceschopnost organizací 2011, Zlín, UTB, pp. 347–354, ISBN 978-80-7454-012-7.

[12] Boon, C., Philippaerts, P., Piessens, F., *Practical experience with the .NET cryptographic API*, Katholieke Universiteit Leuven, CW Reports vol. CW531, 2008, [Online], Available: https://lirias.kuleuven.be/bitstream/123456789/208274/1/CW531.pdf [24 Feb 2014].

[13] Canteaut, A., Chevallier-Mames, B., Gouget, A., Paillier, P., *Shabal, a Submission to NIST's Cryptographic Hash Algorithm Competition*, Shabal, 2008, [Online], Available: https://www.shabal.com/wp-content/uploads/Shabal.pdf [19 Feb 2014].

[14] Bertoni, G., Daemen, J., Peeters, M., Assche G., V., *The Keccak reference,* The Keccak sponge function family, 2011, [Online], Available: http://keccak.noekeon.org/Keccak-reference-3.0.pdf [20 Feb 2014].

[4] The SHA-3 Cryptographic Hash Algorithm Competition, see http://csrc.nist.gov/groups/ST/hash/sha-3/