https://doi.org/10.26552/com.C.2015.1.73-80

Roman Jasek *

# SHA-1 AND MD5 CRYPTOGRAPHIC HASH FUNCTIONS: SECURITY OVERVIEW

*Despite their obsolescence and recommendations they are phased out from production environment, MD5 and SHA-1 cryptographic hash functions remain defaults frequently offered in many applications, e.g., database managers. In the article, we present a security overview of both algorithms and demonstrate the necessity to abandon them in favor of more resilient alternatives due to low computational requirements necessary to reverse engineer the message digests, or to future proof security due to advances in hardware performance and scalability. Suitability procedures and their methods of use are part of this article.*

***Keywords:*** *Algorithm, bcypt, function, hashing, MD5, PBKDF2, security, SHA-1, scrypt.*

## 1. Introduction

Sensitive data protection has been in focus of security researchers for a long time. While extensive academic coverage analyzing existing and proposed cryptographic hash algorithms exists, organizations are slow to adopt them due to inertia, backward compatibility issues, increased hardware requirements, and deployment costs. When benefits of these changes are not clearly communicated, keeping cryptographic systems up-to-date is deprioritized due to lacking technical background.

Website frontends are frequently vulnerable to one or more techniques such as SQL injection, null byte injection, buffer overflow, directory traversal, and uncontrolled format strings. Relying solely on network perimeter security elements should not constitute basis for leaving critical portions of data storages unencrypted. However, some data encryption schemes do not guarantee adequate level of security. To detect changes in databases consisting millions of records, various mathematical fingerprinting techniques were devised titled cryptographic hash functions which provide computationally efficient way to generate, store, and manipulate (compare, move, delete) the control strings with marginal time requirements. They are also used for storing sensitive user data in scrambled form, thereby reducing the attack surface.

Definition of sensitive data varies. Legal incentives, namely Payment Card Industry Data Security Standard codify proper handling and storage of financial data [1]. European Union's Data Protection Directive 95/46/EC was enacted in 1995 [2] and in 2012, a major reform titled General Data Protection Regulation has commenced which plans to streamline protection and sharing of personally identifiable data of all member states' citizens. Unless noted, sensitive data will refer to any confidential electronic assets users willingly disclosed which may compromise their electronic identities or integrity if obtained by unauthorized third party. To preclude such situations, data may be converted to a fixed-size output using a hash function.

Advanced chip designs with high integration of transistors (whose power is growing according to Moore's law [3]) represent high computing power for malicious code from third parties. This risk will also increase over time.

For this reason, revisions must be made as to what cryptographic hash algorithms are sufficient and suitable to protect sensitive with respect to brute-force and dictionary attacks, allowing attackers to enumerate billions of combinations per unit of time, rendering the hash scheme inefficient if deployed incorrectly.

The article provides security overview of two popular but obsoleted hashing algorithms still used in production environment: MD5 and SHA-1. While they have been proven computationally insecure or incapable to future proof applications as per Moore's law mentioned above, they are nevertheless widely deployed as alternatives to comparably more secure schemes for backward compatibility or legacy reasons. It is structured as follows: Section 2 provides security overview of MD5 and SHA-1 cryptographic functions, outlining their design and describing timeline of significant attacks. Section 3 lists best practices applicable to hashing sensitive data, including cryptographic salts to thwart exhaustive searches and dictionary attacks, key strengthening

* **Roman Jasek**
Department of Applied Informatics, Tomas Bata University of Zlin, Czech Republic
E-mail: jasek@fai.utb.cz

which imposes computational penalty when reverse engineering hashes. Section 4 continues by describing key strengthening and mentions suitable alternatives to MD5 and SHA-1 designed specifically with key stretching and strengthening in mind, and brief concluding remarks.

Security in various forms, i.e., message authentication [4] and protecting data in transit [5] is imperative for data confidentiality, integrity, and availability. We believe the article will contribute to safer organizational environments by incentivizing system administrators and appropriate parties to migrate from insecure or weakened cryptographic hash functions to alternatives scrutinized by academia and security community.

## 2. Cryptographic hash functions

A cryptographic hash function, commonly abbreviated as a hash is a "...function, mathematical or otherwise, that takes a variable-length input string (called a pre-image) and converts it to a fixed-length (generally smaller) output string (called a hash value)" [6]. Hash is alternatively titled checksum, although checksums validate homogeneity and consistency of a data block while hashes serve multitude of functions apart from integrity checks, e.g., authentication, watermarking, digital signature schemes, or MACs (Message Authentication Codes) mentioned in Section 4.

Important properties of a hash are fixed bit length, irreversibility, and fast calculation. Once the input is processed into a digest, no operation is theoretically capable to produce the pre-image. However, as the hash is of fixed size, a brute-force attack can be mounted where all candidate pre-images are converted and compared to the original fingerprint. If a match is made, the hash constitutes either the original pre-image, or a different one which hashes to the same value, i.e., a collision. The attack is extremely time- and resource-intensive and was considered impractical when first cryptographic hash functions were devised.

Another feature is that a bit change in the pre-image results in at least 50% change in the hash, a phenomenon known as (strict) avalanche effect [7]. Avalanche effect ensures the data has not been tampered by a simple fingerprint check. Hashing is a lossy process and input source information content is not preserved. The functions are thus unusable as a storage solution but only to ensure pre-image validity through comparison. It is demonstrated in Fig. 1.

Hashes have become a widely-utilized means of validating any type of data with the only requirement being binary input form. Software libraries are usually provided by database vendors out of the box with the option of purchasing additional packages. As most corporations nowadays limit expenditures into information technology, it is not reasonable to assume database management systems (DBMSs) as well as other applications will be enhanced in such a way. Therefore, encryption modules included by default in many instances of DBMSs will be considered: MD5 and SHA-1.

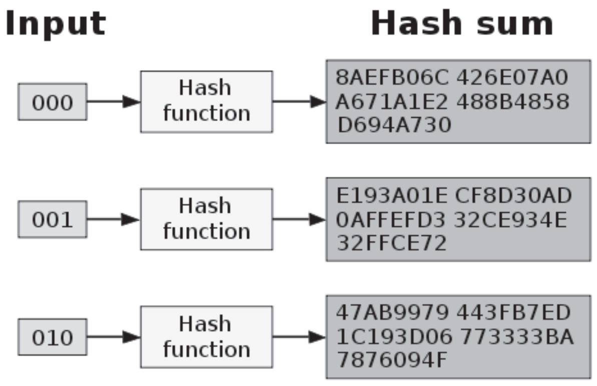
Input
Hash sum
000
Hash function
8AEFB06C 426E07A0 A671A1E2 488B4858 D694A730
001
Hash function
E193A01E CF8D30AD 0AFFEFD3 32CE934E 32FFCE72
010
Hash function
47AB9979 443FB7ED 1C193D06 773333BA 7876094F

*Fig. 1 Avalanche effect for SHA-1 [8]*

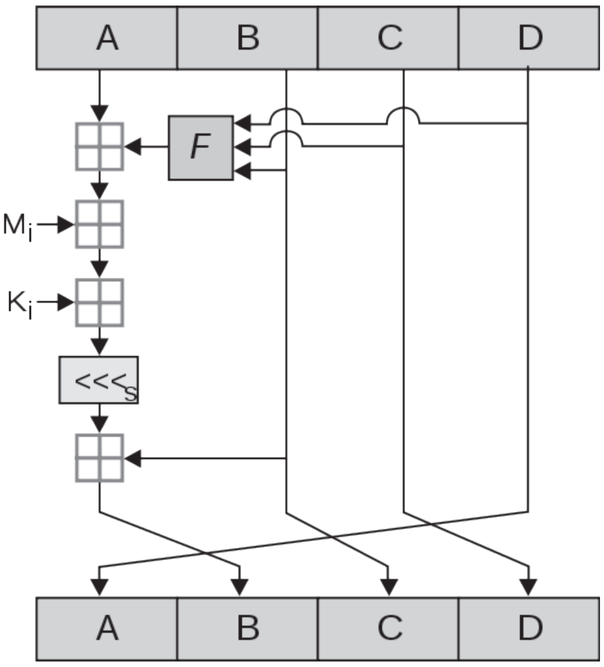

*Fig. 2 An MD5 encryption round Source: [9]*

*A. MD5*

The MD5 Message-Digest Algorithm is a 128-bit, 4-rounds function proposed by Ronald Rivest [10]. Successor to MD2 and MD4, it was designed as an industry standard and sanctioned by the Internet Engineering Task Force (IETF) to be a part of the Internet protocol suite. MD5 is represented by a 32-byte hexadecimal string.

As every encryption scheme seeing widespread adoption, MD5 was heavily scrutinized by both security researchers and academia. From the properties listed in Section 1, it was known the function is vulnerable to hash collisions where the attacker searches for a pre-image that hashes to the same product. If found, it can be exploited to impersonate legitimate users and invalidate the authentication procedure. Initially, it was shown a colliding hash can be found in 15 minutes on a supercomputer setup [11] which led to recommendations that MD5 be not used when generating digital certificates. In practice, adversary can parallelize the computations on consumer-grade hardware to gain performance comparable to low-tier supercomputer.

MD5 encryption round is schematically depicted in Fig. 2. Consisting of 64 iterations grouped by 16, $M_i$ denotes 32b data block obtained by dividing the input message into 512b chunks, padding if necessary. $K_i$ represents a constant added in each round, specified in the original standard [10]. $<<<_s$ denotes bitwise left shift for *s* positions, with *s* differing in each round, *F* a non-linear function, the primary source of complexity when reverse engineering the digest. The function is inputted to adder modulo $2^{32}$. The process is repeated with different constant supplied in each round.

MD5 utilizes Merkle-Damgard construction [12] and [13] which postulates that if the compression function is collision resistant, the hash function utilizing it will be also collision resistant. Depicted in Fig. 3, the "... message *m* is divided into equal size message blocks $x_1 || ... || x_n$, the one-way compression function is denoted as $H_k$ and $x_0$ denotes the initial value with the same size as message blocks $x_1...x_n$ ($x_0$ is implementation or algorithm specific and is represented by an initialization vector). The algorithm then starts by taking $x_0$ and $x_1$ as input to the compression function $H_k$ and outputs an intermediate value of the same size of $x_0$ and $x_1$. Then for each message block xi, the compression function $H_k$ takes the result so far, combines it with the message block, and produces an intermediate result. The last message block $x_n$ contains bits representing the length of the entire message *m*, optionally padded to a fixed length output" [14].

In 2004, first practical attack on MD5, its predecessor MD4 as well as several other hash functions was successfully executed [15]. Further improvements were made based on these findings. In 2005, a pair of digital certificates compliant with the X.509 Public Key Infrastructure (PKI) standard was produced, proving the attack was feasible in real-world applications [16]. The collision mechanism is depicted in Fig. 4.

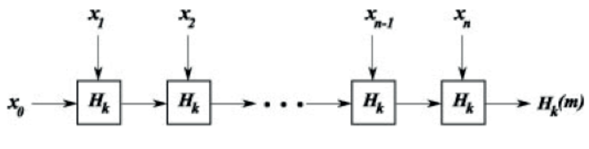

*Fig 3. Overview of Merkle-Damgård construction [14]*

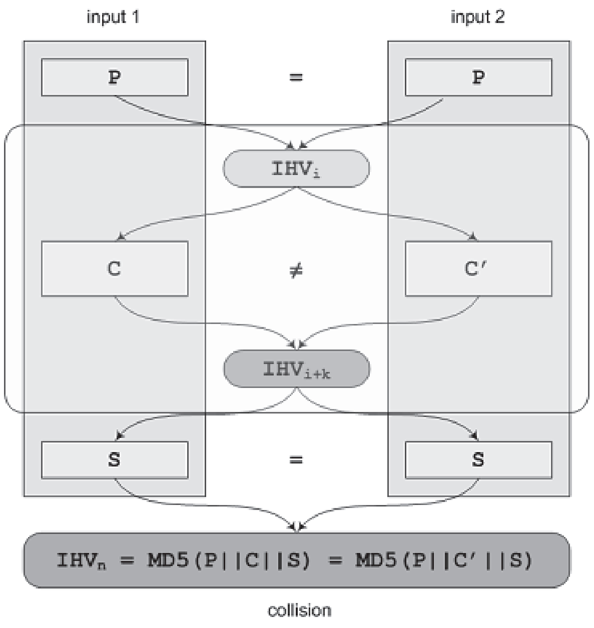

*Fig. 4 MD5 collision demonstration [17]*

Authors stated that "[d]ue to the iterative structure of MD5 and to the fact that $IHV_0$ [intermediate hash value] can have any 128 bit value, such collisions can be combined into larger inputs. Namely, for any given prefix *P* and any given suffix *S* a pair of "collision blocks" *[C,C']* can be computed such that *MD5(P||C||S) = MD5(P||C'||S)*. We use the term 'collision block' for a specially crafted bit string that is inserted into another bit string to achieve a collision. One collision block may consist of several input blocks, even including partial input blocks" [17]. Therefore, for two non-equal inputs, a collision block can be calculated which when inserted into the hashing sequence, produce two identical outputs. The only prerequisite is for the intermediate hash value to be identical for both pairs. A chosen-prefix attack extends it to arbitrary IHVs.

Two modifications of the flaw were demonstrated, allowing identical signatures to be produced on a single machine with the former capable of performing the operation in several hours using consumer-grade notebook, the latter achieving the same goal within 60 seconds on the same hardware; it was stated that "[w]e did not use any supercomputer to find the collisions, just ordinary desktop computers. The author conducted his experiments on his notebook where he found tens of thousands of collisions for the first block and subsequently complete MD5 collisions for both original IV [initialization vector] and chosen IVs" [18]. This proved MD5 can be trivially reverse engineered, significantly reducing security and making it unsuitable for protecting sensitive data.

A research was also conducted which tested propensity to collision attacks in the PKI model. The resulting certificate corroborated that "[it] allows us to impersonate any website on the Internet, including banking and e-commerce sites secured using the HTTPS [Hypertext Transfer Protocol Secure]" [17].

In 2008, the United States Computer Emergency Readiness Team (US-CERT) announced that "[s]oftware developers, Certification Authorities, website owners, and users should avoid using the MD5 algorithm in any capacity. As previous research has demonstrated, it should be considered cryptographically broken and unsuitable for further use" [19]. In 2012, a new attack purportedly demonstrated MD5's susceptibility to single-block collisions, enabling the attacker to forge 64-byte messages with arbitrary hash value, was announced [20]. Cryptographic community recommended migration to SHA-1.

## *B. SHA-1*

The Secure Hash Algorithm 1 was designed by the United States National Security Agency (NSA) in 1995 as a successor to the 1993's SHA-0. With a 160-bit digest iterated for 80 rounds, it was used for protecting sensitive unclassified information as well as in Internet protocols such as Secure Sockets Layer (SSL) and Secure Shell (SSH) [21]. SHA-1 is represented as a 40-character sequence.

SHA-1 encryption round is depicted in Fig. 5. *A-E* are 32b words identical to MD5, *F* a non-linear function, <<< bitwise shift for arbitrary number of positions, $K_t$ a round constant, and $W_t$ an input data block. The output of *F* enters an adder modulo $2^{32}$. The function is based on Merkle–Damgard construction.

Touted an MD5's replacement, SHA-1 saw enormous rise in applications which led to its thorough examination by the cryptographic community. Previously, research focused on its predecessor, SHA-0 for which a collision was found using disturbance vectors with "complexity [of] $2^{39}$ hash operations. Compared with existing attacks on SHA-0, our method is much more efficient and real collisions can be found quickly on a typical PC. The techniques... are also applicable to SHA-1. As SHA-0 may be viewed as a simple variant of SHA-1, the analysis... serves to verify effectiveness of these new techniques for other SHA variants" [22].

The result showed it is possible to find a collision in SHA-1 while requiring fewer computations than it would take to brute-force the hash, the most time- and resource-intensive cryptanalytic process.

Further attempts were made to reduce the number of operations after which the collision is found. A significant breakthrough was made in 2006 when "...for the first time an actual collision for 64-step SHA-1 is produced, with an expected work factor of $2^{35}$ compression function computations" [24]. Since then, several attempts have been made to extend the attack on full SHA-1 with mixed results. The latest discovery is dated to 2013 when a researcher estimated theoretical number of computations for full SHA-1 to $2^{61}$ operations [25].

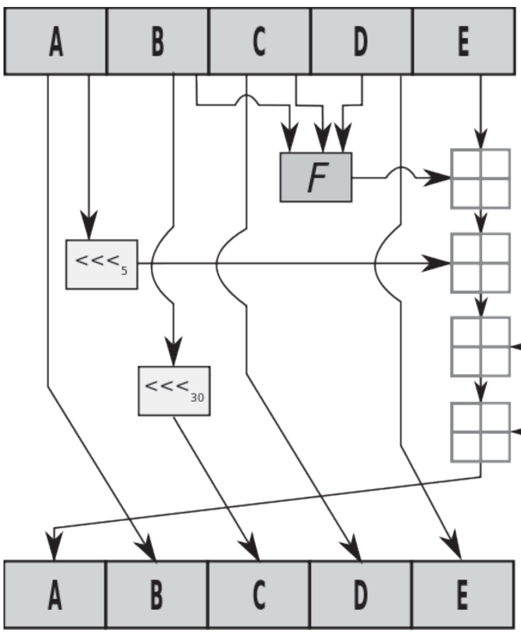

*Fig. 5 An SHA-1 encryption round [23]*

Because computational complexity of attacks on SHA-1 has been steadily decreasing, SHA-2 (SHA-224, SHA-256, SHA-384, SHA-512) class was devised as a direct successor. However, both systems are based on identical algorithmic operations and it is expected optimized SHA-1 attacks will be applicable to SHA-2, as well.

In 2012, a successor to SHA-1 and SHA-2 was selected by the NIST (National Institute of Standards and Technology) after an open competition, aiming to choose a function dissimilar to its predecessors. Currently published reverse engineering attempts break 46 out of 64 rounds for SHA-256 [26] and equivalent amount of rounds for the 80-round SHA-512, it is expected the full system will be targeted eventually despite it being computationally infeasible at present time. SHA-3 utilizes functions with sponge construction [27], making harder for the attacker to differentiate it from a random oracle, a theoretical scenario in which any input is encrypted randomly in a black-box setting. Any outside agent cannot discern whether the output was produced based on a random function or a genuine encryption algorithm if no other information (timing measurements, heat emissions, cycle counts) is known. Independent on SHA-2, known attack vectors are not applicable to SHA-3.

Two theoretical vectors against SHA-3 were proposed: a zero-sum attack applicable to the 9-round reduced version with no effect on its security [28]; and an improved zero-sum distinguisher which applies to all 24 rounds and lowers the number of operations from $2^{1579}$ to $2^{1570}$ [29]. Both were published before the final version of SHA-3 was selected; no practical cryptanalytic breakthroughs on the final implementation has been published as of yet.

In 2011, National Institute of Standards and Technology asserted that "...the known research results indicate that SHA-1 is not as collision resistant as expected. The collision security strength is significantly less than an ideal has function (i.e., [$2^{69}$] compared to [$2^{80}$]).... [C]ollision resistance has been shown to affect some (but not all) applications that use digital signatures" [30].

## 3. Best practices

Regardless of the hash function, the security best practice for storing sensitive data such as user credentials (logins, passwords) is to utilize randomized hashing. As the hash itself is deterministic (two identical strings produce identical outputs), additional probabilistically-generated data need to be supplanted and processed along with the input data stream. Titled cryptographic salt, its purpose is to increase time factor involved when adversary employs rainbow tables, a list of pre-computed values which speeds the process of iterating through the whole search space. Adding salt is depicted in Fig. 6.

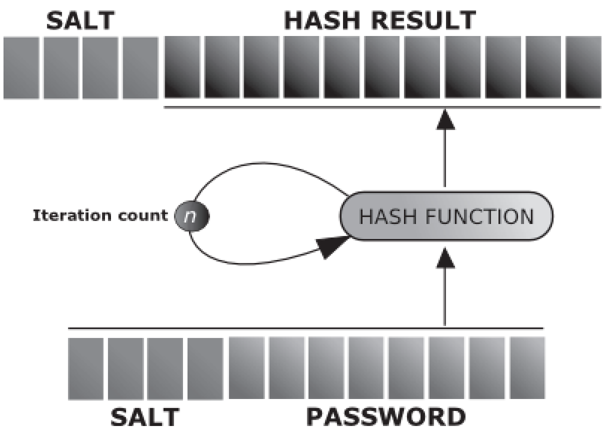

*Fig. 6 Adding cryptographic salt to a password before hashing as well as concatenating another salt with the message digest [31]*

NIST recommends "[t]he random value... [to] be a message-independent bit string of at least 80 bits, but no more than 1024 bits... [which] shall have sufficient randomness to meet the desired security strength..." [32]. Cryptographic salt should, therefore, be generated using a random number generator whose output meets randomness criteria, e.g., Linear Complexity, Approximate Entropy, Binary Matrix Rank, and Serial Tests [33]. A single value should not be used globally, instead a per-user or per-application salt stored in a database separate from the hashes is recommended. It is introduced to force threat agent to generate large sets of candidate hashes for to the string being reverse engineered. "[T]he number of possible resulting [hashes] is approximately $2^{sLen}$ where *sLen* is the length of the salt in bits. Therefore, using a salt makes it difficult for the attacker to generate a table of resulting [hashes] for even a small subset of the most-likely passwords" [34].

Even when generating (pseudo)random salts, they may be rendered ineffective if the attacker can exploit vulnerabilities in the way they are concatenated and added to the strings. Two frequent omissions are salt reuse and short salts. The former is "ineffective because if two users have the same password, they'll still have the same hash. An attacker can still use a reverse lookup table attack to run a dictionary attack on every hash at the same time. They just have to apply the salt to each password guess before they hash it," the latter does not prevent the attacker to "build a lookup table for every possible salt.... To make it impossible for an attacker to create a lookup table for every possible salt, the salt must be long. A good rule of thumb is to use a salt that is the same size as the output of the hash function" [35]. A one-time (pseudo) random data string is titled nonce; encryption schemes have been proposed which makes it impossible to decrypt (reverse engineer) the product without the nonce [36].

A break-through occurs when an attack vector enabling pre-image extraction after lower number of operations (and thus time factor involved) is discovered than during exhaustive search. It is defined as "[a]n attack that uses a brute-force technique of successively trying all the words in some large, exhaustive list" [37].

Cryptographic salts also make time-memory tradeoff difficult to implement. First described in 1980 [38], the technique trades time dedicated to calculating candidate solutions for a pre-computed lookup data array where a simple search algorithm can be applied to find the correct value. A threshold exists, though, above which table lookups become costly and ineffective. After several improvements, a new version was introduced in 2003 making use of non-merging rainbow chains, addressing the issue in the original proposal [39]. The technique achieved 99.9% success rate when reverse engineering Microsoft Windows LM hashes with a lookup table the size of 1.4GB. As the prices of storage media decreases per Moore's law, rainbow tables in tens of terabytes will proliferate which utilize high-speed storage media such as SSD (Solid-State Drive).

If the input to the hash function concatenated with a salt prior to being reduced to a fixed-size output, the attacker is forced to pre-compute the lookup array for every possible salt value. Therefore, security depends on uniqueness and length of the random value being appended or prepended to the (presumably) non-random input string. Salts, key strengthening and key stretching make time-memory tradeoff difficult to balance compared to a brute-force attack. Key stretching is discussed in Section 4. Key strengthening was devised in 1994 and splits the salt in two parts: public and secret [40]. While the public part is stored, the secret is securely deleted after first use and becomes unknown. When the user enters a password, the server must perform a brute-force attack using the public part of the salt to determine the secret portion, increasing both per-user computational requirements and security. The attacker, though, must exhaustively search the whole hash space, i.e., both parts of the salt. The salt or its part and the algorithm used to generate the output must be known server-side to allow comparison of the data to the stored value. No plaintext-formatted data should be stored at any point, only the fingerprints.

Compared to alternatives discussed below, implementing cryptographic hashes does not guarantee the adversary will not be able to extract the pre-image. Should system administrators be forced to select one of the countermeasures, i.e., transitioning to a new cryptographic hash function or adding salt to the existing infrastructure, the former should be strongly preferred as it results in significantly higher computational demands during reverse engineering. Moore's law dictates effectiveness of salting will decrease as hardware performance increase. By deploying strong cryptographic hash function, the work factor can be easily tweaked by means of parameters used during hashing, e.g., iteration count and parallelizability.

## 4. Alternatives and Discussion

In the paper, a security overview of SHA-1 and MD5 systems was presented. While MD5 phase out has been somewhat slow, by the time the shift to SHA-1 is completed, it may be necessary to discard the system in favor of a more secure one.

To ensure long-term resilience of the stored hashes to offline brute-force attacks, security community also recommends computationally-intensive hashing algorithms such as SHA-512 which generate more secure outputs; processing overhead is, however, increased. Every entity dealing with sensitive data must decide whether this benefit outweighs the disadvantage of higher computational demands, usually abundant in pervasive cloud infrastructure on a pay-per-use basis.

An alternative proposed in 1996 is titled Hash-based Message Authentication Code (HMAC). It guarantees increased security for MAC by combining hashing scheme with a cryptographic key [41]. Output strength is dependent on the hash function (SHA-1, MD5) and its bit size along with parameters of the key. Theoretical attacks exist which don't, however, in any way subvert HMAC security. Ways have been devised on how to compute HMAC efficiently in hardware to ensure minimum latency [42].

Several functions have been released which aim to make cryptographic hashing resource-demanding by introducing arbitrary computational overhead respected during reverse engineering. Implementation libraries utilizing key stretching are freely available: bcrypt, scrypt, PBKDF2, etc. Key stretching purposefully slows the process as much as possible by using longer salts, higher number of iterations (each round is repeated an arbitrary number of *times*), and limiting parallelizability through data arrays stored in memory. As demonstrated in Fig. 7, PBKDF2 utilizes HMAC.

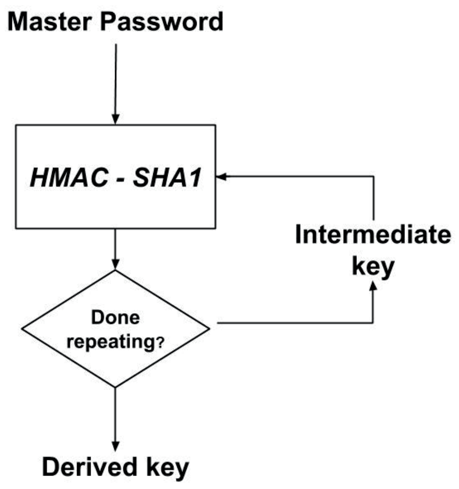

*Fig. 7 Password-Based Key Derivation Function 2 [43]*

Scrypt in particular hinders reverse engineering with rainbow tables as well as ASIC (Application-Specific Integrated Circuit), FPGA (Field-Programmable Gate Array) and GPU (Graphics Processing Unit), all dedicated hardware modules exhibiting high computational throughput for repetitive mathematical operations. Generating a large vector of pseudorandom bit strings in memory, it accesses the structure in a pseudorandom fashion [44]. Each element in the vector is resource-intensive to generate and can be accessed on many occasions during the algorithm's run, precluding workload distribution to a cluster of nodes. It is a memory-hard algorithm which "...asymptotically uses almost as many memory locations as it uses operations... [I]t can be also thought of as an algorithm which comes close to using the most memory possible for a given number of operations, since by treating memory addresses as keys to a hash table it is trivial to limit a Random Access Machine to an address space proportional to its running time" [44]. By tweaking three parameters, CPU/memory cost, parallelization, and block size, computational demands imposed on computing the hash can be increased arbitrarily.

Website administrators should be informed about advances in hash function cryptanalysis to ensure timely transitions to a well-developed and proven scheme with adequate security for data-storing infrastructures. Increasing the time factor involved via per-user salt, high iteration counts, and thorough testing should be considered priorities.

MD5 has been proven insecure against several attacks under realistic assumptions and its use is discouraged in favor of more resilient, key-stretching iterative hashing algorithms. Despite no full SHA-1 hash collisions have been produced so far, advances as per Moore's law and future proofing should be taken into account when selecting suitable cryptographic hash function to deploy.

Hash functions have seen increased use in areas such as concurrent algorithm design [45] and continue to be active research field.

**Acknowledgement**

The work was performed with financial support of research project NPU I No. MSMT-7778/2014 by the Ministry of Education of the Czech Republic and also by the European Regional Development Fund under the Project CEBIA-Tech No. CZ.1.05/2.1.00/03.0089.

## References

[1] PCI Security Standards Council. *Payment Card Industry Data Security Standard 2.0* [Online]. Available: https://www.pcisecuritystandards.org/security_standards/documents.php, 2010.

[2] EU: *Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the Protection of Individuals with Regard to the Processing of Personal Data and on the Free Movement of such Data* [Online]. Available: http://eur-lex.europa.eu/LexUriServ/LexUriServ. do?uri= CELEX:31995L0046:en:HTML, 1995.

[3] MOORE, G. E.: Cramming More Components onto Integrated Circuits, *Electronics*, vol. 38, No. 8, pp. 4-8, April 1965.

[4] LEE, T.-Y., LEE, H.-M.: Encryption and Decryption Algorithm of Data Transmission in Network Security, *WSEAS Trans. Inf. Sc. Appl.*, vol. 3, No. 12, pp. 2557-2562, 2006.

[5] QAWASMEH, E., MASADEH, E.: Developing and Investigation of a New Technique Combining Message Authentication and Encryption, *WSEAS Trans. Inf. Sc. Appl.*, vol. 3, no. 7, pp. 1417-1422, 2006.

[6] SCHNEIER, B.: *Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C*. New Jersey : Wiley, 1996.

[7] FEISTEL, H.: Cryptography and Computer Privacy, *Sci. Am.*, vol. 228, no. 5, pp. 15-23, May 1973.

[8] GOTHBERG, D.: *Avalanche effect.svg*, 2006 [Online]. Available: https://commons.wikimedia.org/ wiki/File:Avalanche_effect.svg

[9] SUNACHIT: *MD5.svg*, 2005 [Online] Available: https://commons.wikimedia.org/wiki/File: MD5.svg

[10] RIVEST, R.: *The MD5 Message Digest Algorithm*, 1992 [Online]. Available: http://tools.ietf.org/ html/rfc1321

[11] WANG, X., YU, H.: How to Break MD5 and Other Hash Functions, *Lect. Notes Comput. Sc.*, No. 3494, pp. 561-577, 2005.

[12] DAMGARD, I. B.: A Design Principle for Hash Functions, *Lect. Notes Comput. Sc.*, No. 435, pp. 416-427, 1990, doi: 10.1007/0-387-34805-0_39

[13] MERKLE, R. C.: A Certified Digital Signature, *Lect. Notes Comput. Sc.*, No. 435, pp. 218-238, 1990, doi: 10.1007/0-387-34805-0_21

[14] SPRENGERS, M.: *GPU-based Password Cracking: On the Security of Password Hacking Schemes regarding Advances in Graphics Processing Units*, M. S. thesis [Online]. Fac. Sc., Radboud Univ. Nijmegen, Nijmegen, The Netherlands, 2012. Available: http://enricopagliarini. com/wp-content/uploads/2012/02/thesis.pdf

[15] WANG, X., FENG, D., LAI, X, YU, H.: *Collision for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD*, 2004 [Online]. Available: http://eprint.iacr.org/2004/199

[16] LENSTRA, A., WANG, X., De WEGER, B.: *Colliding X.509 Certificates,* 2005 [Online]. Available: http://eprint.iacr.org/2005/067

[17] SOTIROV, A., STEVENS, M., APPELBAUM, J., LENSTRA, A. et al.: *MD5 Considered Harmful Today*, 2008 [Online]. Available: http://www.win.tue.nl/hashclash/rogue-ca/

[18] KLIMA, V.: *Finding MD5 Collisions - a Toy for a Notebook*, 2006 [Online]. Available: http://eprint.iacr.org/2005/075

[19] US-CERT: *MD5 Vulnerable to Collision Attacks*, 2008 [Online]. Available: http://www.kb.cert.org/vuls/id/836068

[20] STEVENS, M.: *Single-block Collision for MD5*, 2012 [Online]. Available: http://marc-stevens.nl/research/md5-1block-collision/

[21] EATLAKE, D. 3rd, JONES, P.: *US Secure Hash Algorithm 1 (SHA1)*, 2001 [Online]. Available: tools.ietf.org/html/rfc3174

[22] WANG, X., YU. H. IN, Y. L.: Efficient Collision Search Attacks on SHA-0, *Lect. Notes Comput. Sc.*, vol. 3621, pp. 1-16, 2005, doi: 10.1007/11535218_1

[23] PIETRYGA: *SHA-1.svg*, 2007 [Online]. Available: https://commons.wikimedia.org/wiki/File:SHA-1.svg

[24] CANNIERE, C. RECHBERGER, C.: Finding SHA-1 Characteristics: General Results and Applications, *Lect. Notes Comput. Sc.*, No. 4284, pp. 1-20, 2006.

[25] STEVENS, M.: New Collision Attacks on SHA-1 Based on Optimal Joint Local-collision Analysis, *Lect. Notes Comput. Sc.*, No. 7881, pp. 245-261, 2013, doi: 10.1007/978-3-642-38348-9_15

[26] LAMBERGER, M, MENDEL, F.: *Higher-Order Differential Attack on Reduced SHA-256*, 2011 [Online]. Available: http://eprint.iacr.org/2011/037
[27] BERTONI, G., DAEMEN, J., PEETERS, M. ASSCHE, G.: *Sponge Functions,* Proc. ECRYPT Hash Workshop 2007, Barcelona, 1997.
[28] AUMASSON, J. P., MEIER, W.: *Zero-sum Distinguishers for Reduced Keccak-f and for the Core Functions of Luffa and Hamsi*, 2009 [Online]. Available: https://131002.net/data/papers/AM09.pdf
[29] MING, D. XUAJIA, L.: *Improved Zero-sum Distinguisher for Full Round Keccak-f Permutation,* 2011 [Online]. Available: http://eprint.iacr.org/2011/023
[30] POLK, T., CHEN, L., TURNR, S., HOFFMAN, P.: *Security Considerations for the SHA-0 and SHA-1 Message-Digest Algorithms*, 2011 [Online]. Available: http://tools.ietf.org/html/rfc6194
[31] FERNANDEZ, D.: *How to Encrypt User Passwords,* 2013 [Online]. Available: http://www.jasypt.org/howtoencryptuserpasswords.html
[32] DANG, O.: *NIST Special Publication 800-106: Randomized Hashing for Digital Signatures,* 2009 [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-106/NIST-SP-800-106.pdf
[33] RUKHIN, A., SOTO, J., NECHVATAL, J., SMID, M.: *NIST Special Publication 800-22, Revision 1a: A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications,* 2010 [Online]. Available: http://csrc.nist.gov/publications/nistpubs/ 800-22-rev1a/SP800-22rev1a.pdf
[34] TURAN, M. S., BARKER, E., BURR, CHEN, L.: *NIST Special Publication 800-132: Recommendation for Password-Based Key Derivation, Part 1: Storage Applications,* 2010 [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-132/nist-sp800-132.pdf
[35] HORNBY, T.: *Salted Password Hashing - Doing it Right*, 2013 [Online]. Available: https://crackstation.net/hashing-security.htm
[36] WU, M.-L.: Nonce-aware Encryption Scheme, *WSEAS Trans. Inf. Sc. Appl.*, vol. 6, No. 9, pp. 1513-1522, 2009.
[37] SHIREY, R.: *Internet Security Glossary, Version 2*, 2007 [Online]. Available: https://tools.ietf. org/html/rfc4949
[38] HELLMAN, M.: A Cryptanalytic Time-Memory Trade-Off, *IEEE Trans. Inf. Th.*, vol. 26, No. 4, pp. 401-406, 1980.
[39] OECHSLIN, P.: *Making a Faster Time-Memory Trade-Off,* Proc. of 23rd Annu. Int. Cryptology Conf. (CRYPTO 2003), Santa Barbara, pp. 617-630, 2003.
[40] MANBER, U.: *A Simple Scheme to Make Passwords Based on One-Way Functions Much Harder to Crack,* 1994 [Online]. Available: http://webglimpse.net/pubs/TR94-34.pdf
[41] BELLARE, M., CANETTI, R., KRAWCZYK, H.: *Keying Hash Functions for Message Authentication,* 1996 [Online]. Available: http://cseweb.ucsd.edu/~mihir/papers/kmd5.pdf
[42] MICHAILH, E., KAKAROUNTAS, A.P., E. FOTOPOULOU, E., GOUTIS, C. E.: Novel Hardware Implementation for Generating Message Authentication Codes, *WSEAS Trans. Commun.*, vol. 4, No. 11, pp. 1276-1283, 2005.
[43] SHINER, J.: *Defending Against Crackers: Peanut Butter Keeps Dogs Friendly, Too,* 2011 [Online]. Available: http://blog.agilebits.com/2011/05/05/defending-against-crackers-peanut-butter-keeps-dogs-friendly-too/
[44] PERCIVAL: *Stronger Key Derivation via Sequential Memory-Hard Functions,* 2009 [Online]. *Proc. BSDCan'09*, Ottawa, 2009. Available: http://www.bsdcan.org/2009/schedule/attachments/ 87_scrypt.pdf
[45] DUDAS, A., JUHASZ, S.: Blocking and Non-blocking Concurrent Hash Tables in Multi-core Systems, *WSEAS Trans. Comput.*, vol. 12, No. 2, pp. 74-84, 2013.