Password Hashing: the Future is Now

2013.07.11

Jean-Philippe Aumasson (@aumasson)

Kudelski Security, Switzerland

1 Introduction

When?	Who?	How many?
March 2013	Evernote	≈50 million
April 2013	LivingSocial	≈50 million
May 2013	Reputation.com	?
July 2013	Ubisoft	≈58 million

This table reports the number of password hashes compromised in recent breaches of popular web services. It is probably the best argument in favor of secure storage of passwords. But why passwords can be compromised whereas hashing is supposed to protect them?

Most web services that authenticate their users (webmails, social network services, etc.) do it with pair username/password: to login in the web application of the service, you send your username and your password to the web server, which checks in its database that the given username is already registered and that the password is identical to the password set by that user. But how is this last step performed?

Some web servers store your password in clear in their database (these are the services that send you your password by email when you hit "I forgot my password"), therefore password verification is just a comparison of two strings. This is an extremely risky and irresponsible approach, because an attacker who gains access to the database directly gets the password of each user. Such an attacker may then impersonate a user on the website attacked, or on another website where this user is registered (most people reuse a same password accross several services).

Some other web servers store a *hash* of your password. A hash is computed by applying a function that transforms a string of arbitrary length to a random-looking string of fixed length (for example, 16 bytes). The goal is to prevent an attacker to read your passwords if she gains access to the database. However, if the attacker knows the hash function used, she can try different passwords until one matches the hash value observed (for example, using a dictionary of the most commonly used passwords). The degree of protection against such bruteforce attacks varies greatly with the hash function used:

- **Cryptographic hash functions**, such as MD5, SHA-1, or SHA-256: these functions are typically very fast (several hundreds of megabytes per second on a desktop CPU), which is undesirable against bruteforce attacks. Furthermore, a given password is always hashed to the same value regardless of the user; this exposes the system to time-memory trade-off attacks (for example, using ”rainbow tables”), which are much faster than dictionary attacks.

- **Cryptographic hash functions with a salt**: a salt is an auxiliary input to the hash function that is selected randomly when a user sets his password. The fundamental goal of salts is to simulate the use of different hashing algorithm. Therefore, a same password hashed with two different salts will have two different hash values. This prevents time-memory trade-off attacks, because an attacker does not know in advance the salt used. However bruteforce attacks remain as fast as with unsalted hash functions.

- **Password-hashing functions**, also called password-based key derivation functions: these functions mitigate bruteforce attacks by being significantly slower, and sometimes requiring a significant amount of memory (to increase the cost of bruteforce on technologies such as GPUs or FPGAs). Such functions thus provide a much greater protection. However, password-hashing function are not well understood, and only a handful of constructions have been proposed (PBKDF2 [5], bcrypt [4], and scrypt [3] are the most common).

The security and cryptography communities now have a much better understanding of password hashing than a few years ago. It is thus time to develop a mature design for protecting passwords, that will provide enhance security compared to previous proposals and that will be easy to deploy across platforms and systems. Indeed, password-based authentication is used more broadly than for just websites: mobile devices, operating systems, full-disk encryption, SSH keys, etc.

We advocate the development of the new password-hashing function will be performed through a public competition, a model that has proved effective to select cryptographic algorithms (see the AES, eSTREAM, or SHA-3 competitions). The goal of this competition, named *Password Hashing Competition* (PHC) is threefold:

- To promote the development of best-of-breed algorithms for securing passwords,

- To encourage cryptographic research in this area, and

- To develop standards and usage recommendations for password hashing algorithms.

In the remainder of this white paper, we highlight the technical challenges of developing new reliable password hashing methods, and introduce the PHC’s agenda and timeline.

2 Technical challenges

Developing new password hashing methods is arguably more challenging than (say) block ciphers or hash functions, due to the high dependence of security on the underlying technology—be it that of defenders of or attackers—and to the relative youth of the field, with few research works published.

Below we attempt to summarize the main challenges related to the design and deployment of new password hashing methods (we prefer to talk of "method" or "scheme" rather than "function" or "algorithm" because several algorithms and physical or logical components may be involved in a given method). This list is by far not exhaustive.

2.1 Software and hardware engineers

Perhaps the main challenge in the design of a password hashing scheme is the creation of a method with minimized efficiency on GPUs and FPGAs—and, to a lesser extent, ASICs—and maximized efficiency on general-purpose CPUs. For example, the method should not easily lend itself to pipelining and to parallelism of multiple instances; a corollary is that a single instance should have a reasonable degree of parallelism (for example to exploit AVX2 instructions in general-purpose CPUs on legit servers).

To measure the relative value of different hashing methods with respect to that fuzzy notion of "slower for attackers, faster for defenders", metrics—or at least heuristics—should be created. These might be developed with respect to specific technologies (for example, a given model of GPU card), or to more abstract models of computation (for example, non-uniform circuits to model [programmable] hardware). A somewhat similar challenge was encountered in cryptographic competitions to assess the relative security of block ciphers or hash functions, and the notion of "security margin" was considered. Although obviously imperfect, that notion helped comparing submissions. We expect performance metrics of password hashing to also be fuzzy and controversial, but hopefully they will be helpful guides in the selection process.

A related challenge is to foresee future advances in technology (for example, new types of platforms, or more "dedicated" hardware à la Xeon Phi), progress of existing hardware (how will Intel server chips look like in 10 years?), and the associated costs (how will the curve RAM GB vs. dollars look like?).

Hardware engineers are expected to contribute hardware architectures for FPGAs and ASICs for both defensive and offensive purposes.

2.2 Security engineers

Challenges for security engineeers are numerous; below we only list a handful of them, which may or may not be the most relevant during the competition:

- Should hashing be performed by servers, clients, or both? For which applications? For example, offloading the "slow" part of hasing to the client could help mitigate

the risk of DoS on the server. However the operator of a web service knows the hardware of his server (and can tune the hashing parameters accordingly) whereas clients can be a variety of platforms, from powerful stations to cheap mobiles. The speed of client-side hashing would thus be very variable.

- How to design methods that allow to update the hash database to a different security level (for example, to adapt to a new server's hardware or to attackers' progress) without requiring a fresh user login. The composition of a "fast hash" followed by a "slow hash" naturally comes to mind, however there may be more intelligent and secure solutions.

- What is the relative role of native implementations versus scripting languages? For example, browsers would easily integrate JavaScript implementations of password hashing methods, but the slowdown compared to a native code should not be too important.

2.3 Cryptographers

Besides being offered a new set of cryptanalysis targets (for attacks on security notions as collision resistance, preimage resistance, pseudorandomness, unpredictability, or indistinguishability from an "ideal" password hashing function—a notion yet to be defined), cryptographers will find new research problems associated with password hashing, whose results may be of independent interest.

For example, one wants to ensure that the large computation and memory requirements of a hashing method cannot be bypassed using some computation tricks (e.g. with precomputed lookup tables). Techniques from complexity theory or algorithms analysis may be used to prove lower bounds on the time and/or space complexity of a given (class of) algorithm(s), that is, to show evidence that the complexity claims are true. Such proofs may be established in specific computation models; for example, one may prove that a given hash cannot be computed by a circuit with fewer than N NAND gates and with a depth lower than D.

Another type of challenge to cryptographers, close to the typical research published in conferences as CRYPTO or EUROCRYPT, is the design of constructions (a.k.a. modes of operation) proved to be secure given "ideal" underlying primitives, such as pseudorandom functions (PRFs), universal hash functions, etc. Such result are expected to provide methods that are simpler (that is, as simple as possible to achieve the target security), and to considerably increase the confidence.

2.4 Attackers

Password crackers (who may also be, or work with, cryptographers) will play a critical role in the competition, as they will simulate future real attackers. Professionals of password cracking will help to optimize implementations for high-performance platforms such

as GPUs, and to find any trick to reduce the cost of retrieving passwords (be it with a single target or as a batch attack).

3 The Password Hashing Competition

The Password Hashing Competition (PHC) is an initiative inspired by previous public cryptographic competitions: AES, eSTREAM, SHA-3, and more recently CAESAR; we refer to http://competitions.cr.yp.to for an overview of those projects. Such public, targeted crypto competitions proved effective to crowdsource the design and analysis effort, so as to eventually select one or more primitives. It is thus natural to adopt the same model for the development of password hashing schemes.

Initiated in fall 2012, the PHC is organized by a panel of experts from industry, academia, and government institutions (NIST), which includes the leading experts in both the defensive and offensive aspects. Motivations behind the PHC include:

- The poor state of passwords protection in web services: passwords are too often either stored in clear (these are the services that send you your password by email after hitting "I forgot my password"), or just hashed with a cryptographic hash function (like MD5 or SHA-1), which exposes users' passwords to efficient brute force cracking methods.
- The low variety of methods available: the only standardized construction is PBKDF2, and there are mainly just two alternatives, bcrypt and scrypt, which both have several undesirable properties.
- A number of new ideas discussed within the security and cryptography communities, but which have not yet led to a concrete proposal.

We stress that the PHC is organized by a group of individuals, not by a standardization body. However this does not exclude the future standardization of one or more of the schemes selected.

After publishing the call for submissions in February 2013, the next stage of the competition starts on January 31, 2014, the submission deadline. The selection of finalists submission (a shortlist of candidates for the final selection) is expected in Q3 2014, and the selection of a final portfolio on Q2 2015. PHC aims to identify diverse methods covering a broad range of applications, and providing innovative techniques to better protect passwords (or PINs, passphrases, etc.).

For the sake of completeness, the list of panel members and the call for submissions of PHC are copied in Appendix of this paper. More details are available on the website of the project, https://password-hashing.net.

References

[1] Burt Kaliski. PKCS #5: Password-Based Cryptography Specification Version 2.0. RFC 2898 (Informational), 2000.

[2] Burt Kaliski. PKCS #5: Password-Based Key Derivation Function 2 (PBKDF2) Test Vectors. RFC 6070 (Informational), 2011.

[3] Colin Percival. Stronger key derivation via sequential memory-hard functions. In *BSDCan*, 2009. See also http://www.tarsnap.com/scrypt.html.

[4] Niels Provos and David Mazières. A future-adaptable password scheme. In *USENIX Annual Technical Conference, FREENIX Track*. USENIX, 1999.

[5] Meltem Sönmez Turan, Elaine Barker, William Burr, and Lily Chen. NIST SP 800-132: Recommendation for password-based key derivation part 1: Storage applications, 2010. See also [1, 2].

A PHC panel members

The PHC is organized by a panel of experts consisting of

Tony Arcieri (@bascule, LivingSocial)
Jean-Philippe Aumasson (@aumasson, Kudelski Security)
Dmitry Chestnykh (@dchest, Coding Robots)
Jeremi Gosney (@jmgosney, Stricture Consulting Group)
Russell Graves (@bitweasil, Cryptohaze)
Matthew Green (@matthew_d_green, Johns Hopkins University)
Peter Gutmann (University of Auckland)
Pascal Junod (@cryptopathe, HEIG-VD)
Poul-Henning Kamp (FreeBSD)
Stefan Lucks (Bauhaus-Universität Weimar)
Samuel Neves (@sevenps, University of Coimbra)
Colin Percival (@cperciva, Tarsnap)
Alexander Peslyak (@solardiz, Openwall)
Marsh Ray (@marshray, Microsoft)
Jens Steube (@hashcat, Hashcat project)
Steve Thomas (@Sc00bzT, TobTu)
Meltem Sonmez Turan (NIST)
Zooko Wilcox-O'Hearn (@zooko, Least Authority Enterprises)
Christian Winnerlein (@codesinchaos, LMU Munich)
Elias Yarrkov (@yarrkov)

These experts will be responsible for the final selection of a portfolio of schemes, based on the public contribution and on their assessment of the submissions received. They

will be permitted to submit schemes, however they will not participate in discussions regarding their own submission.

B PHC call for submissions

The Password Hashing Competition (PHC) organizers solicit proposals from any interested party for candidate password hashing schemes, to be considered for inclusion in a portfolio of schemes suitable for widespread adoption, and covering a broad range of applications.

Submissions are due by January 31, 2014. All submissions received that comply with the submission requirements below will be made available on the website of the project, https://password-hashing.net.

Technical guidelines

The submitted password hashing scheme should take as input at least

- A password of any length between 0 and 128 bytes (regardless of the encoding).
- A salt of 16 bytes.
- One or more cost parameters, to tune time and/or space usage.

The scheme should be able to produce (but is not limited to) 16-byte outputs. If multiple output lengths are supported, the output length should be a parameter of the scheme. Similarly, if multiple salt lengths are supported, the salt length should be a parameter. Passwords longer than 128 bytes may be supported, but that is not mandatory. Other optional inputs include local parameters such as a personalization string, a secret key, or any application-specific parameter.

Submissions will be evaluated according the following criteria:

Security

- Cryptographic security: the function should behave as a random function (random-looking output, one-way, collision resistant, immune to length extension, etc.).
- Speed-up or other efficiency improvement (e.g., in terms of memory usage per password tested) of cracking-optimized implementations (checking multiple sets of inputs in parallel, and doing so in a CPU's native code) compared to implementations intended for password validation should be minimal.
- Speed-up or other efficiency improvement (e.g., in terms of area-time product per password tested) of cracking-optimized ASIC, FPGA, and GPU implementations

(checking multiple sets of inputs in parallel) compared to CPU implementations intended for password validation should be minimal.

- Resilience to side-channel attacks (timing attacks, leakages, etc.). In particular, information should not leak on a password's length.

Simplicity

- Overall clarity of the scheme (design symmetries, modularity, etc.).
- Ease of implementation (coding, testing, debugging, integration).
- Use of other primitives or constructions internally (the fewer, the better).

Functionality

- Effectiveness of the cost parameter (e.g. can the time and space expected requirements be bypassed?).
- Ability to transform an existing hash to a different cost setting without knowledge of the password.

Submitters are encouraged to propose innovative constructions and methods for protecting passwords against attackers that have fully or partially compromised a server storing password hashes. For example, one may design a scheme that is slow to evaluate except on a server given some server-specific shortcut. Submissions may also be specific to a specific application, such as mobile devices (e.g. to protect PINs), key derivation (e.g. for full-disk encryption), scripting languages (as opposed to native implementations), etc.

Submission requirements

Submissions should be sent to submissions@password-hashing.net on or before January 31, 2014 as a compressed archive (tar.bz2, tar.gz, or zip). All submissions will be acknowledged.

The following are to be provided with any submission:

Cover sheet

- Name of the submitted scheme (preferably a valid C identifier).
- Name and email address of the submitter(s).

Specification

- Complete and unambiguous description of the scheme; however if the scheme reuses an existing primitive, this primitive need not be described (for example, if the scheme uses AES, it is not necessary to copy the specification of AES).
- Statement that there are no deliberately hidden weaknesses (backdoor, etc.); any sign of such ill intent will be grounds for disqualification.

Initial security analysis

- Discussion of the security claims and usage constraints of the proposed algorithm: For which usage scenarios do the designers claim their algorithm secure, and when should it not be used?
- Discussion of the security of the algorithm, and its dependence on the security of cryptographic primitives used by the algorithm.

Efficiency analysis

- Discussion of the performance of the scheme on the target platforms (that is, mainstream software): expected speed of an optimized implementation, ability to exploit modern CPUs features (SIMD or multicore), etc.
- Discussion of the performance of the algorithm on platforms that may be used for high-speed password cracking (ASIC, FPGAs, GPUs); if possible, an argument why password-cracking on those platforms is not quite cost-effective.

Code

- Reference implementation in portable C(++) with necessary build instructions (e.g. a Makefile). Using C++ internally is allowed, but the program should provide an external C API. OpenSSL's libcrypto may be used (e.g. for AES, SHA-256). The API should include, but may not be limited to, a function with the following prototype:
  ```
  int PHS(void *out, size_t outlen, const void *in, size_t inlen, const void *salt, size_t saltlen, unsigned int t_cost, unsigned int m_cost);
  ```
  The reference implementation should aim at simplicity and readability, rather than at performance.
- Comprehensive set of test vectors (preferably including all byte values in the 0 to 255 range for both the password and the salt inputs).
- Optionally, implementations in other languages or specific to a given CPU/GPU, microarchitecture, etc.

Intellectual property statement

Statement that the scheme is and will remain available worldwide on a royalty free basis, and that the designer is unaware of any patent of patent application that covers the use or implementation of the submitted algorithm.