Password Hashing Competition

JP Aumasson, Kudelski Security

This talk

- The competition process
- Special recognitions
- The winner: Argon2
- Post-mortem / Aftermath

Why PHC?

- Legacy hashes not satisfying:
  - PBKDF2: low-memory
  - bcrypt: 4KB isn’t enough memory today
  - scrypt: complex to use, therefore not used
- Public crypto competitions work well so far

Support

- No funding, sponsoring, nor donations
- All work made on our free time
- No dedicated workshop, but talks at PasswordsCon

Timeline

- 2012 Q4: Idea shared on Twitter, panel created
- 2013 Q1: Call for submissions published
- 2014 Q1: 24 submissions received
- 2014 Q4: 9 finalists selected
- 2015 Q3: 1 winner announced (Argon2)

The panel

- Cryptographers, hackers, password crackers, software engineers
- From industry, FOSS community, academia, gov
- Diversity crucial to deliver relevant work

Call

- Requirements are the most important
- Then, evaluation criteria (can still be changed later)
- Should leave enough freedom to submitters
- No major disagreement within the panel

https://password-hashing.net/cfh.html

Minimal requirements likely sufficient for most applications

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) 32-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.

Tentative evaluation 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.

No distinction reference vs. optimized code, but asked to prioritize simplicity over performance

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 reference implementation should aim at simplicity and readability, rather than at performance.
  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 `t_cost` and `m_cost` arguments are intended to parameterize time and memory usage, respectively, however this is not a strict requirement (only one parameter may be effective, `m_cost` might affect time, etc.).
- 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.

Submissions

- Two extremes
  - “Academic": rigorous specs, rationale, formal proofs, lengthy documentation
  - "Dirty": specs = code, handwaved claims, succinct documentation
- High design diversity, new ideas
- Need to identify the best of each submission

Custom or non-custom?

- Argument for known crypto (AES, etc.): confidence, code readily available, native instructions
- Arguments for custom design: strong crypto overkill when iterated, bloats the design (scrypt)
- Full custom: Makwa, POMELO
- Others using AES, BLAKE2, SHA-2
- Yescrypt based on scrypt, Pufferfish on Blowfish

Side-channel defenses?

- By side-channel we mean cache-timing-like attacks
- Usually when input-dependent memory addressing
- Is it really a concern for password hash?
  - Not for local key derivation, unless cold-boot attacks are a threat
  - Perhaps when on co-located virtual machines

Server relief, hash upgrade?

- **Server relief**
  H(pwd, salt) = ServerHash(ClientHash(pwd, salt))

- **Hash upgrade**
  H(p, s, cost2) = Upgrade(H(p, s, cost1), cost2)

- Not supported by PBKDF2/bcrypt/scrypt

- Nice to have

Time-space tradeoffs?

- Possible when memory addresses and content partially predictable
- Conflicts with side-channel protection
- Now much better understood than before PHC

Decision making

- **Finalists**:

    1. Panel members asked to write their 5 favorite and 5 least favorite submissions, with rationale

    2. First ranking established, basis for private discussions that would decide the 9 finalists

- Submitters weren't allowed to vote, just to comment

Decision making

- **Winner**: One or more?

- Panel members asked to score in 1-3 each finalist in 4 categories: Technical superiority, ease of deployment, features, confidence

- Everyone participated, even submitters

- Private, but told panel that scores/comments may be published later (several but not all were)

- Then, as much discussions as possible held on the public mailing list, except for the final decision

Decision making

- **Argon2 fine-tuning**: all public discussions
  - Tweaks proposed by the designers and the panel
  - Review of specs / code for consistency and quality
  - Took about 3 months

Special recognitions

- Or “second-place” winners
- Quality, innovative submissions
- We thought PHC would have more impact if we gave a single recommendation, rather than (say) 5 recommendations for different use cases
- Catena, Lyra2, Makwa, yescrypt

Catena

- By Christian Forler, Stefan Lucks, Jakob Wenzel
- Most comprehensive submission:
  - Framework for password hashes
  - 1-round BLAKE2b as
  - All aspects analyzed: side-channel, TMTO, etc.
- Proofs based on graph theory (pebble games)

Lyra2

- By Marcos A. Simplicio Jr, Leonardo C. Almeida, Ewerton R. Andrade, Paulo C. F. dos Santos, Paulo S. L. M. Barreto

- Sponge-based design, well analyzed

- One of the fastest, on defenders platforms

- Uses a dedicated BLAKE2 variant, BlaMka, with MUL operations

Makwa

- By Thomas Pornin
- Totally different from the rest: bignum arithmetic
- Iterates x^2 mod n, inverse as hard as factoring
- Server offload: private key allows for efficient verification, client has to do it the hard way

yescrypt

- By Solar Designer
- Evolution of scrypt
- Many tunable features:
  - Large ROM lookup table
  - scrypt-compatibility mode
  - Parallelization parameters
  - Alternative to Salsa20 (PWXform)

Argon2

- PHC winner!
- By Alex Biryukov, Daniel Dinu, Dmitry Khovratovich
- Overhaul of the initial candidate Argon
- Two versions: Argon2d and Argon2i

https://www.cryptolux.org/index.php/Argon2
https://github.com/P-H-C/phc-winner-argon2
https://github.com/khovratovich/Argon2

Argon2 I/O

- Mandatory password and salt
- Optional secret value and associated data
- Cost parameters:
  - Memory size (in KB)
  - Number of iterations
  - Parallelism
- Returns a tag of at least 4 bytes

Argon2 in a nutshell

- Aimed to be as simple as it can be

$$
\begin{aligned}
B[0] &= H(P, S); \\
\text{for } j &\quad \text{from 1 to } t \\
& B[j] = G\big(B[\phi_1(j)], B[\phi_2(j)], \cdots, B[\phi_k(j)]\big),
\end{aligned}
$$

- H and G based on BLAKE2b

- Indexing different for Argon2d and Argon2i

Argon2 features

- Good security analysis and performance
- Yet a simple design, using trusted crypto
- Three knobs for three distinct parameters
- Leverages understanding from PHC discussions

Attack on Argon2?

Balloon Hashing: Provably Space-Hard Hash Functions with Data-Independent Access Patterns

Henry Corrigan-Gibbs
Stanford University

Dan Boneh
Stanford University

Stuart Schechter
Microsoft Research

January 14, 2016

- TMTO with lower memory than expected/proved
- For Argon2i only (side-channel resistant version)
- Precomputation of the “useless" memory blocks

Argon2 today

- Argon2 main reference for users https://github.com/P-H-C/phc-winner-argon2
  - 34 issues submitted, 39 pull requests
    - Non-trivial bugs, portabilities issues, API, etc.
    - Third-party bindings for 8 languages
    - Support by Argon2 designers, Samuel Neves, me
- Being integrated in Sodium, Debian (more to come)

PHC today

- Archives available on https://password-hashing.net/

- Mailing list still active

PHC: What went well

- Quality of the submissions
- Agressive timeline, with only minor delays
- Active public ML discussions, public archives
- Flexibility of the process and criteria
- Transparency, higher than in other competitions

Could’ve been better

- Reports and justifications of our choices
- Clarity of rules on tweaks (Argon2 first rejected)
- Description of the voting process
- Amount of third-party cryptanalysis

Lessons learned

- As much progress in 2 years as in the past 20 years; competition is a good research motivator

- Rules need be flexible enough to integrate progress made during the competition

- Processes and deliberations should be as transparent and open as possible

Thanks

- NIST for this invitation
- PHC submitters and panel members
- Peter Gutmann, for letting me borrow from his pres https://www.cs.auckland.ac.nz/~pgut001/pubs/phc.pdf