Argon2 for password hashing and cryptocurrencies

Alex Biryukov, Daniel Dinu,
Dmitry Khovratovich

University of Luxembourg

2nd April 2016

Motivation

Password-based authentication

Keyless password authentication:

- User registers with name I and password p;
- Server selects hash function H, generates salt s, and stores $(I, H(s, p))$;
- User sends (I, p') during the login;
- Server matches $(I, H(s, p'))$ with its password file.

Problems:

- Password files are often leaked unencrypted;
- Passwords have low entropy ("123456");
- Regular cryptographic hash functions are cracked on GPU/FPGA/ASIC;
- Many iterations of SHA-256 do little help as this slows down *everyone*.

Understanding brute-force

Brute-force attacks (such as key guessing) are most efficient on custom hardware: multiple computing cores on large ASICs.

Practical example of SHA-2 hashing (Bitcoin):

- 2^{32} hashes/joule on ASIC;
- 2^{17} hashes/joule on laptop.

Consequences

- Keys lose 15 bits;
- Passwords become 3 lowercase letters shorter;
- PINs lose 5 digits.

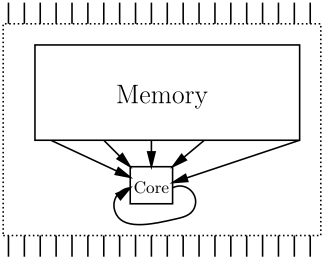
ASIC-equipped attackers are the threat from the near future.

ASICs have high entry costs, but FPGA and GPU are employed too.

Solution

Since 2003, *memory-intensive* computations have been proposed.

Computing with a lot of memory would require a very large and expensive chip.



With large memory on-chip, the ASIC advantage vanishes.

Scrypt

Scrypt [another IETF draft] — designed by Percival in 2009.
Memory-intensive but has problems:

- Too many distinct primitives involved (PBKDF2, SHA-256, Salsa/ChaCha);
- Time and memory tied;
- Simple time-memory tradeoff;
- Timingattack-vulnerable.

Password Hashing Competition (2013-2015)

Requirements for a new scheme:

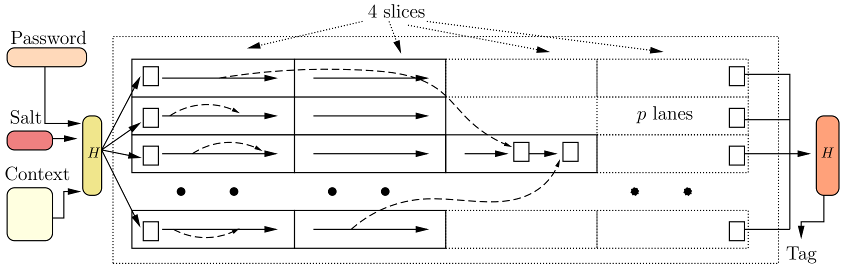
- Maximum cracking cost per password on all platforms;
- Tunable time, memory parameters.
- Security against time-space tradeoffs;
- Transparent design;
- Flexibility.
- Ideally, side-channel protection (missing in scrypt) and tunable parallelism.

Timeline

- 2013: Call for submissions.
- Feb 2014: 24 submissions.
- Dec 2014: 9 second-phase candidates.
- Jul 2015: 1 winner (Argon2), 4 special recognitions: Catena, Lyra2, yescrypt and Makwa (delegation hashing).

Specification of Argon2

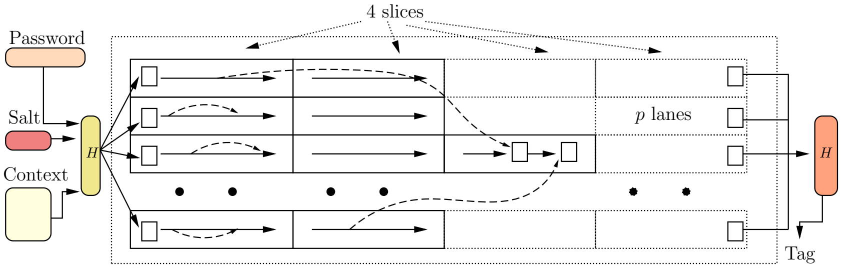
Specification



Two variants: Argon2d and Argon2i.

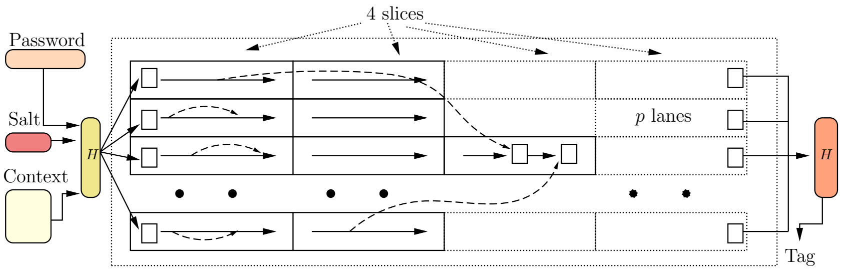
- Argon2d uses data-dependent addressing ($\phi(j) = X[j-1]$);
- Argon2i uses data-independent addressing ($\phi(j) = \text{Blake2b}(j)$);
- The block size is 8192 bits;
- The compression function is based on the Blake2b permutation, enriched with 32-bit multiplications;
- Arbitrarily level of parallelism.

Technical details



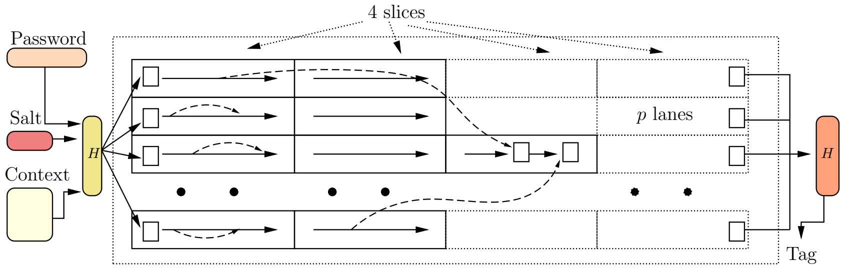
- Password – any length from 0 to $2^{32} - 1$ bytes;
- Salt – any length from 8 to $2^{32} - 1$ bytes, MUST be unique;
- Context (associated data) any length from 0 to $2^{32} - 1$ bytes;
- Key (another input) – from 0 to 32 bytes;
- Parallelism from 1 to $2^{24} - 1$ lanes.

Technical details II



- *H* is a mode over Blake2b, which allows arbitrary output in sponge-like fashion.
- Output tag – from 4 to $2^{32} - 1$ bytes.
- (not in the draft, but can be) Base64-encoded standard password file string for Tag+Salt+Username+Parameters.

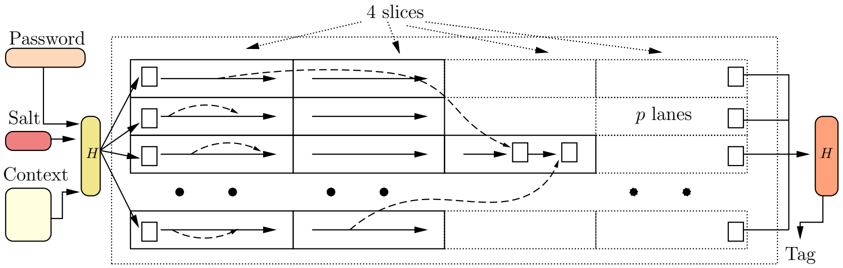
Tweak: from 1.2.1 to 1.3



Several enhancements from the version that won the PHC:

- Total memory up to 4 TB;
- Different way to take pseudo-random data for the reference block index from the previous block (Argon2i);
- In second and later passes over the memory, new blocks are XORed into old ones, not overwrite (rules out some attacks, see the last slide).

Feedback requested



- Should there be any H other than Blake2b;
- Should we specify context further?
- Should we allow salts shorter than 8 bytes?
- Should we restrict password hashing to Argon2i only?

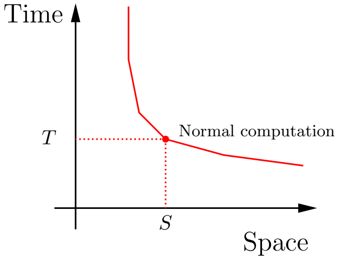
Extra material

Memory-hardness

Time-space tradeoffs and memory-hardness

Clearly, there should be no memoryless equivalent (thus *memory-hardness*).

Time-space tradeoff: how time grows if space is reduced.



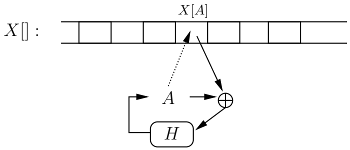
$$T = f(1/S).$$

Linear f means equal trading of space for time. We want f to be superpolynomial.

Time-space tradeoffs and memory-hardness

Provable memory-hard schemes are known since 1970s (FFT and pebble games), but are too slow for password time processing limits.

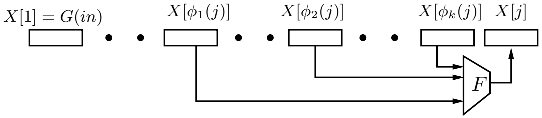
Fast but ad-hoc scrypt [Percival'09] has been the most advanced, but it is very sophisticated (stack of heterogeneus components) and admits a trivial time-memory tradeoff (thus allowing smaller chips).



Design rationale

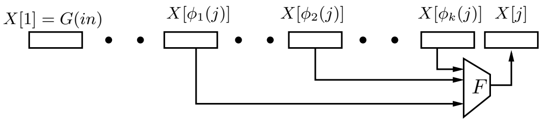
Generic memory-hard function

The following configuration is easy to tune and analyze:



- Filling memory array $X[]$;
- ϕ_i are some indexing functions;
- Computation is sequential.

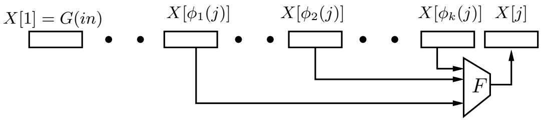
Questions we solved



1. Should the memory addressing be data-dependent or data-independent?
2. How large the block should be?
3. How to exploit multi-threading?
4. How to design F?

Memory addressing

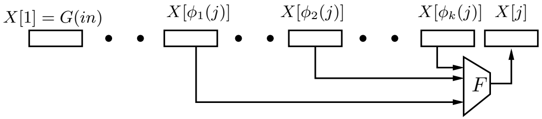
Input-independent addressing



Input-independent addressing:

- The adversary knows all addresses;
- Necessary inputs are prefetched or precomputed (if a tradeoff is applied).
- Memory reduction by 3-4-5 factor with no time increase (unless the bandwidth limits us).

Input-dependent addressing



Input-dependent addressing:

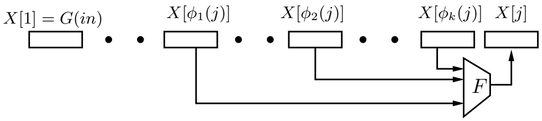
$$\phi(j) = X[j-1] \pmod{j}$$

- The input $X[\phi(j)]$ can not be prefetched/precomputed.
- Execution time increases with memory reduction.
- The scheme becomes vulnerable to cache timing attacks.

Memory fraction α: the time penalty is lower bounded by the *depth* D of the recomputation tree, so the adversary wins as long as

$$\alpha D(\alpha) \leq 1.$$

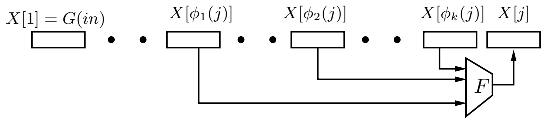
How many blocks



How many blocks should form an input?

- Modern CPU (e.g., Haswell) have two read ports and one write port;
- Assuming the last block is in registers, read of more than 2 random-address (RA) blocks would be slow.

How many blocks



How many blocks should form an input?

- Modern CPU (e.g., Haswell) have two read ports and one write port;
- Assuming the last block is in registers, read of more than 2 random-address (RA) blocks would be slow.

Our experiments with 8192-bit blocks:

	Cycles/block	Bandwidth (GB/sec)
One RA block	1194	4.3
Two RA blocks	1503	5.1

Thus extra block decreases the speed but increases the bandwidth.

Parallelism and multiple passes

Parallelism

If the defender has t cores, he may want to load them all, also increasing bandwidth.

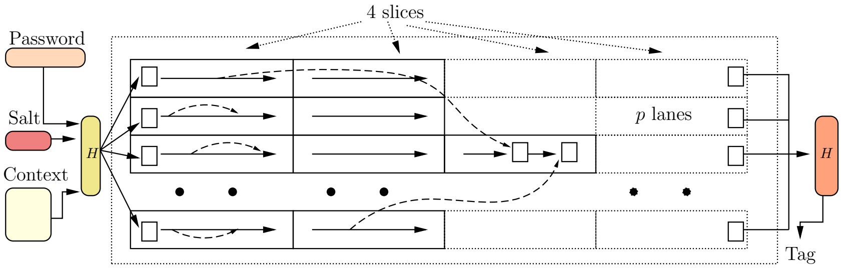
Bad idea:

$$H(X) = G(X||1) \oplus G(X||2) \oplus \cdots \oplus G(X||t).$$

Here memory can be easily traded for time.

Parallelism

Better approach:



- Filling memory with p threads (usually 2 threads per core). Memory is synchronized 4 times.
- A block in a slice can refer to the same slice or to any slice in a previous column.
- Only the last column can be computed sequentially.

Block size and compression function

Block size

There is a tradeoff in the block size:

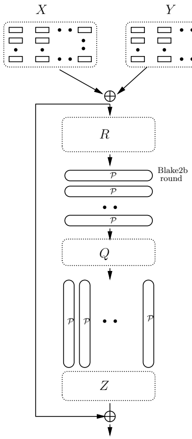
- Larger blocks are loaded faster (per byte) as CPU caches large memory sections;
- Compression functions for smaller blocks are faster with the same security.

Our experiments with the block size, compression function is 1-round Blake2b permutation:

Block size	Cycles per byte
1024	2.4
2048	1.39
4096	1.15
8192	0.96
8192 (2 rounds)	1.19

Our choice

State 8 times as big as Blake2b.



1-round Blake2b first rowwise, then columnwise.

Applications

Systems with no online threats:

- Cryptocurrency mining, that takes 0.1 seconds on a 2 Ghz CPU using 1 core — Argon2d with 2 lanes and 250 MB of RAM;
- Backend server authentication, that takes 0.5 seconds on a 2 GHz CPU using 4 cores — Argon2d with 8 lanes and 4 GB of RAM.

Systems with online attackers:

- Key derivation for hard-drive encryption, that takes 3 seconds on a 2 GHz CPU using 2 cores — Argon2i with 4 lanes and 6 GB of RAM;
- Frontend server authentication, that takes 0.5 seconds on a 2 GHz CPU using 2 cores — Argon2i with 4 lanes and 1 GB of RAM.

Security

Argon2d (1 pass, data-dependent):

- No generic attacks;
- Tradeoff attack: area-time product may be reduced by the factor of 1.5 (ranking method).

Argon2i (1 or 2 passes, never recommended):

- Low storage attack [Corrigan-Gibbs et al. 2016], 1/5 of memory with no penalty.

Argon2i (3 or more passes):

- Low storage attack [Corrigan-Gibbs et al. 2016], 1/3 of memory with no penalty, patched in version 1.3.
- Sandwich attack [Alwen-Blocki'16]: no benefit due to large constant in asymptotic.
- Our tradeoff attack: AT may be reduced by the factor of 3 (ranking method).