Argon2 and Egalitarian Computing

Alex Biryukov Dmitry Khovratovich

University of Luxembourg

January 7th, 2016

I. Unfair battle

Attack-defence paradigm

Attackers have always been more powerful than defenders:

- Large and variable resources;
- One weakness is sufficient;
- Can spend much time.

Defenders can

- Harden the protection (e.g. increase the key length);
- Sometimes restrict the attack vector (e.g. limit the exposure time).

Secure cryptographic algorithm with sufficient key length – solution for many confidentiality, integrity, signatures, etc..

Keyless setting

Sometimes, however, we do not have (long) keys.

- Reliance on human memory (password-based data protection, password-based authentication, PINs, etc.);

Brute-force attacks become possible (e.g., guess a PIN).

Moreover, integrity might become a problem in

- Unencrypted networks (P2P, blockchain).

Understanding brute-force

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

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.

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.

We need to slow down such attackers without burdening the defenders.

II. Argon2 for passwords

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.

Keyless password authentication:

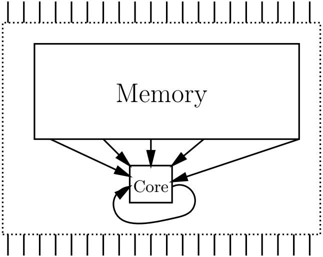
- User registers with name I and password p;
- Server selects hash function H, generates salt s, and stores $(I, H(s, p))$;
- User sends (I, p') during the login;
- Server matches $(I, H(s, p'))$ with its password file.

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

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

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

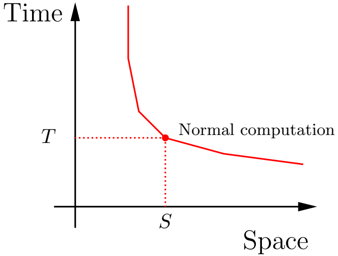


With large memory on-chip, the ASIC advantage vanishes.

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.

Password Hashing Competition (2013-2015)

Requirements for a new scheme:

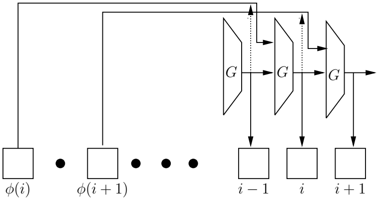
- Maximum cracking cost per password on all platforms;
- Tunable time, memory parameters.
- Security against time-space tradeoffs;
- Transparent design;
- Flexibility.
- Ideally, side-channel protection (missing in scrypt) and tunable parallelism.

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).

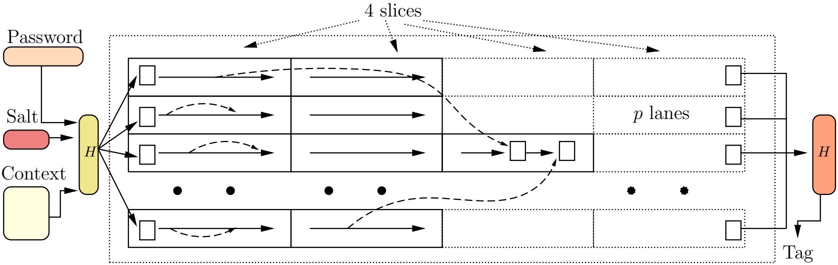
How we designed Argon2

To facilitate analysis, we selected the simplest secure mode of operation:



- Fill memory blockwise;
- Each block is a function of a previous and some older (*reference*) block;
- The reference index may (better tradeoff protection) or may not (side-channel protection) depend on the input;
- Weak and wide compression function G.

Argon2 unleashed



Properties:

- Preimage and collision resistance;
- Adjustable and inseparable parallelism;
- Core: larger and shorter variant (1/5) of Blake2b;
- Exponential time-space tradeoff.

Any part of the Argon2 chain is memory-hard.

Performance

We took a number of steps to speed up the memory filling on the x64 architecture:

- Wide registers and SIMD instructions;
- 1 KB blocks;
- Number of Blake2 rounds balanced with the memory latency.

Multithreaded Argon2 securely fills memory at 0.65 cycles/byte.

Memory bandwidth up to 5.5 GB/sec.

Try

`https://github.com/P-H-C/phc-winner-argon2` [C89]

`https://github.com/khovratovich/Argon2` [C++11]

Apparently, this method of slowing down password crackers has other applications...

III. Egalitarian computing

Why egalitarian

Bitcoin dream

- An egalitarian currency where every user could mine money on his own laptop...

Bitcoin dream

- An egalitarian currency where every user could mine money on his own laptop...

...and reality:

- A bunch of users with factory-size rigs and their own power plants control $> 50\%$ of network in a single pool.

Slowing brute-force

Argon2 ensures that both defenders and attackers hash passwords on the same platform (x86).

Slowing brute-force

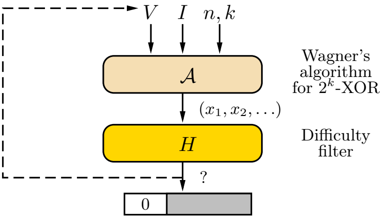
Argon2 ensures that both defenders and attackers hash passwords on the same platform (x86).

This is desirable for some other tasks to slow down brute force on custom hardware:

- Password-based protocols (key agreement, secret sharing);
- Password-based encryption;
- Proofs of work for cryptocurrencies/blockchain;
- Client puzzles for denial-of-service protection.

Proof-of-work – a certificate that confirms that the prover made a certain amount of computations (typically to slow him down for certain time). Clearly the cost of the work must not fluctuate across platforms.

Memory-hard proof-of-work based on Generalized Birthday (k-XOR) problem [NDSS 2016]



x86/GPU-oriented 700-MB proof is 120 bytes long.

Good for ASIC-resistant client puzzles.

Apparently, any NP-complete problem is a natural candidate for a memory-hard proof-of-work...

Egalitarian computing ensures that legitimate users and attackers are equal as they are forced to use the same platform.

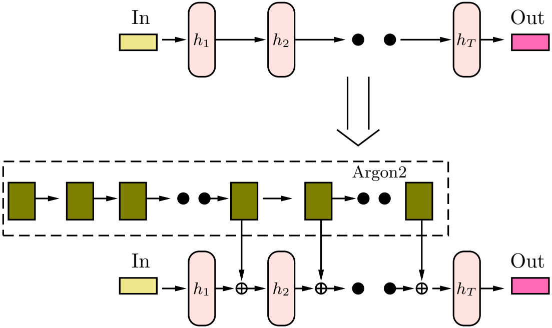
How?

Suppose you develop a scheme where the exact output value is not important (encryption, signature, etc.).

Amalgamate the computation with a memory-hard function such as Argon2.

If you already use some CPU time, why not using the available memory for that period?

Alter the computing: inject memory-hard blocks in between the subfunction calls.



Maybe even feed them back to Argon2 (may need stregthening the compression function).

Memory-hard encryption

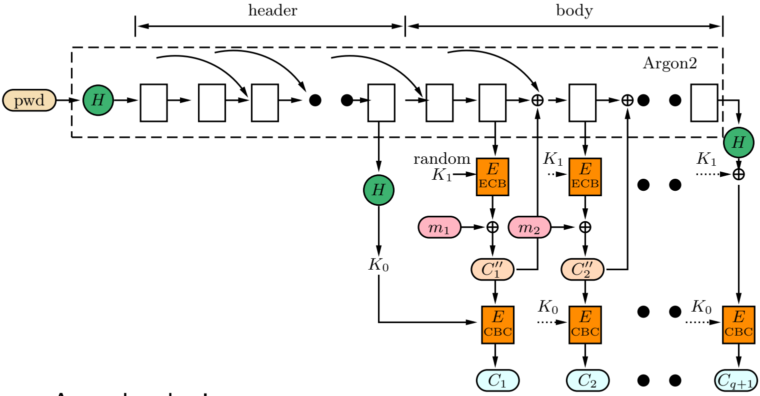
Memory-hard encryption

Password-based disk encryption:

- Encryption by chunks with password-based key;
- Trial decryption requires only a few blockcipher calls;
- Passwords can be tried offline.

We propose to bind it to a memory-hard function to make encryption and decryption run on the same hardware and non-outsourceable.

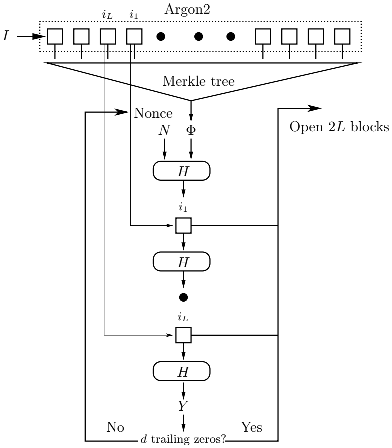
Disk encryption with memory-hard function based on Zaverucha's idea of using All-or-Nothing transform (or another scheme without online decryption):



- Any chunk size;
- Any memory size;
- No way to precompute either part.

MTP: memory-hard proof-of-work based on Argon2

The PC-oriented 2 GB-proof is 180 KB long (faster but longer than Equihash).



Parallelism inevitable so bandwidth-hardness.

Egalitarian computing

- deems 6-letter passwords secure;
- brings back 80-bit keys;
- renders DoS attacks harder;
- suffrages blockchain users.

It is a chance to revert Moore's law.

Cryptography

Some dirty crypto in

- Tradeoff (time-space) cryptanalysis (Asiacrypt 2015);
- Memory-hardness proofs (ePrint on scrypt);
- Memory-hard modes of operation (Argon2 – Euro S&P 2016);
- Asymmetric proof-of-work based on (NP-)hard problems (NDSS 2016);
- Memory-hard obfuscation and white-box cryptography (Asiacrypt 2014).

God may have made men, but Samuel Colt made them equal

Use **Egalitarian Computing**