Egalitarian Computing

Alex Biryukov Dmitry Khovratovich

University of Luxembourg

August 10th, 2016

Attack-defence paradigm

Attackers have always been more powerful than defenders:

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



Attack-defence paradigm

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..

Attack-defence paradigm

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..

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).

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.

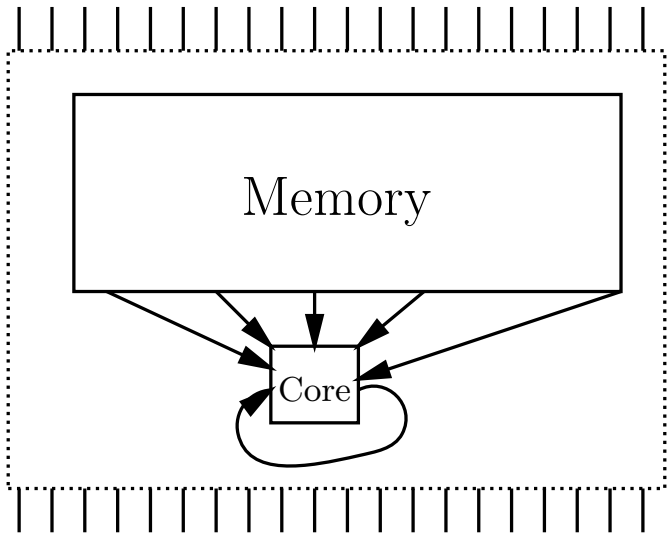
ASICs have high entry costs, but FPGA and GPU are employed too.

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

Solution

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

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

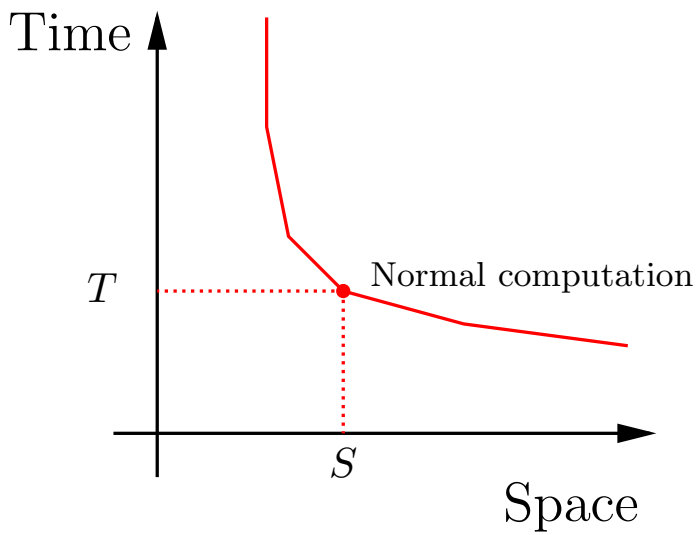


With large memory on-chip, the ASIC advantage vanishes.

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.

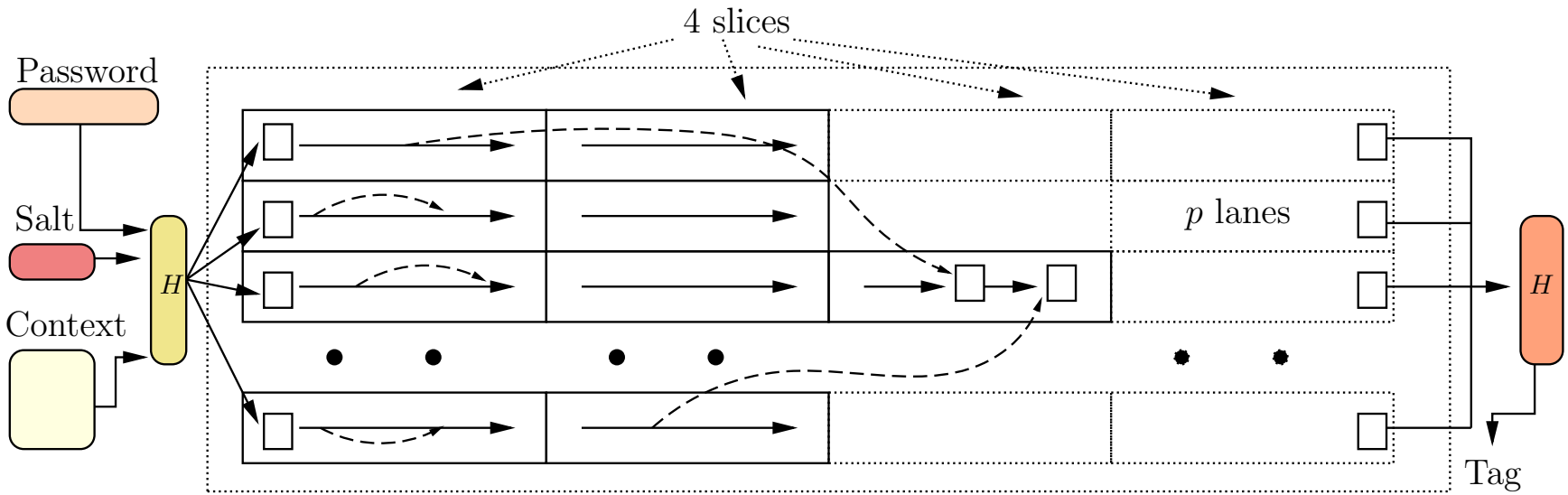


$$T = f(1/S).$$

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

Earlier work: Argon2

Making the password cracking more expensive.
Argon2 [BK'16] – memory-hard hash function, winner of the Password Hashing Competition:



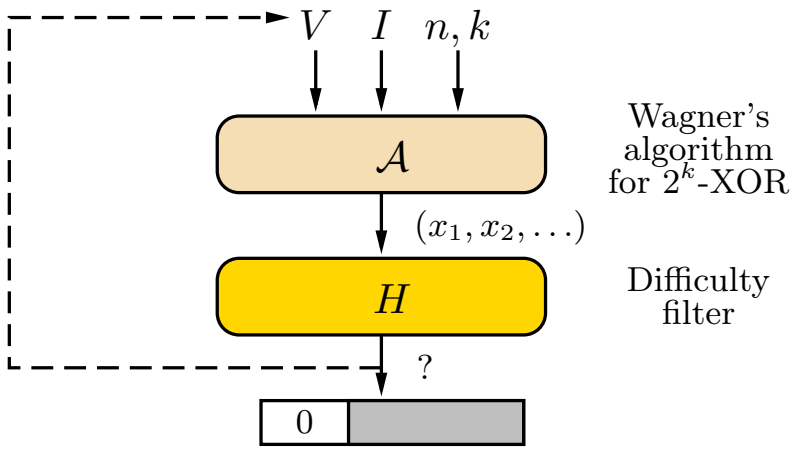
Exponential time growth if space reduced: $\log T = \frac{1}{S}$.

Earlier work: Equihash

Memory-hard proof-of-work with fast verification [BK'16], used in Zcash. *Generalized birthday problem*[Wagner'02]: given seed I, find V and $\{x_j\}$ such that

$$H(I||V||x_1) \oplus H(I||V||x_2) \oplus \cdots \oplus H(I||V||x_{2^k}) = 0. \qquad (1)$$

where H is Blake2b



x86/GPU-oriented 700-MB proof is 120 bytes long and 2 seconds to find.

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

Egalitarian computing

Why egalitarian

Bitcoin dream

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

Why egalitarian

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 made password hashing more secure by ensuring that both defenders and attackers hash passwords on the same platform (x86).

It is desirable to slow down brute force on custom hardware in:

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

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

How?

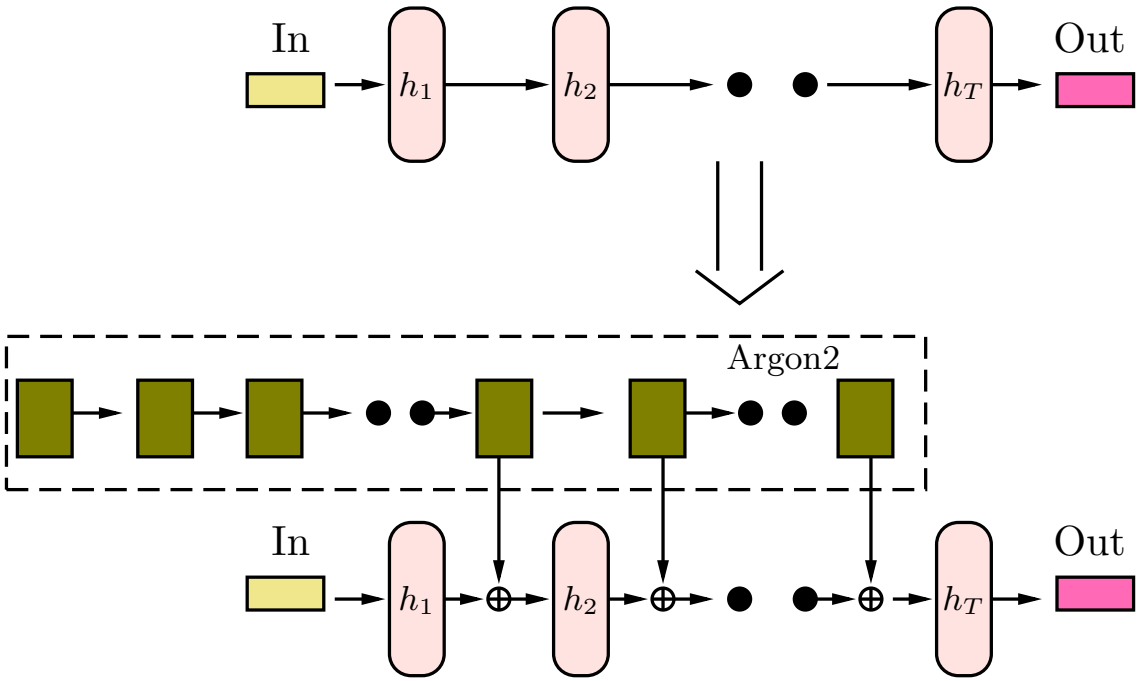
Suppose you develop a scheme where the exact output value is not important (encryption, signature, etc.).

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?

Alteration

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).

Approach 2: obfuscation

Obfuscate the computing with b_i — *memory-hard bit* (extracted from the Argon2-filled memory):

Original		**if** x	**then** A **else** B
Obfuscated	for $b_i = 0$	**if** $x \oplus b_i$	**then** A **else** B
	for $b_i = 1$	**if** $x \oplus b_i$	**then** B **else** A

Memory-hard bits can be built into opaque predicates and other obfuscation tools.

Memory-hard encryption

Disk encryption

Password-based disk encryption: process chunks of data with a password-derived key.
Attack:

- Obtain an encryption of a chunk.
- Decrypt a small block (16 bytes for AES) of the chunk;
- Check if the plaintext has sufficient redundancy.

Problem: trial decryption requires only a few blockcipher calls. Even if you use Argon2 for key derivation, the keys can still be precomputed on a memory-rich device. The trial decryption can be done memoryless on custom hardware using only a small amount of ciphertext.

Stregthening disk encryption

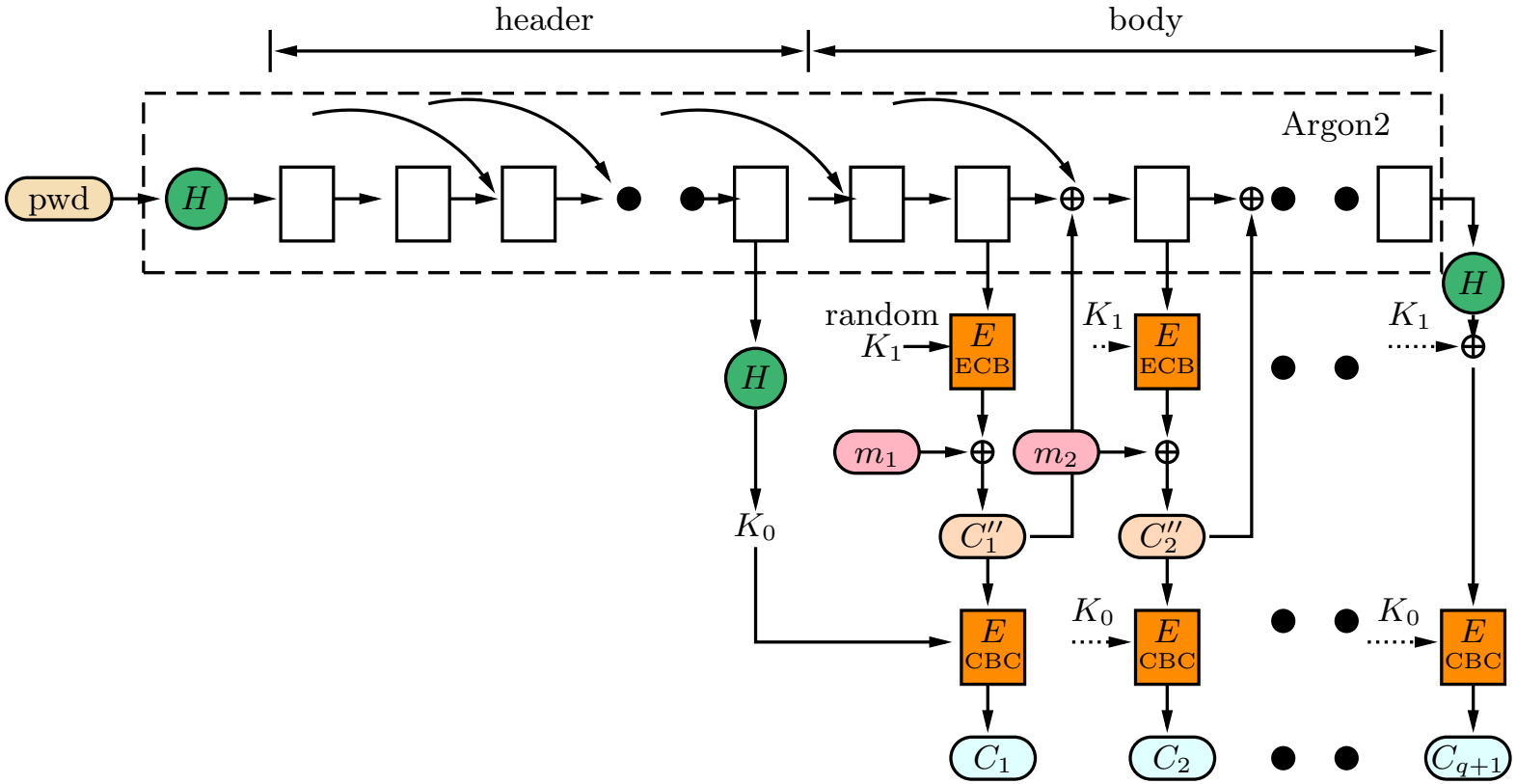
Earlier proposal [Zaverucha'15] with All-or-Nothing transform [Rivest'97]:

- Chunk is decryptable in two passes;
- It must be fully decrypted and kept in memory;
- Memory size is the chunk length.

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

Memory-hard encryption

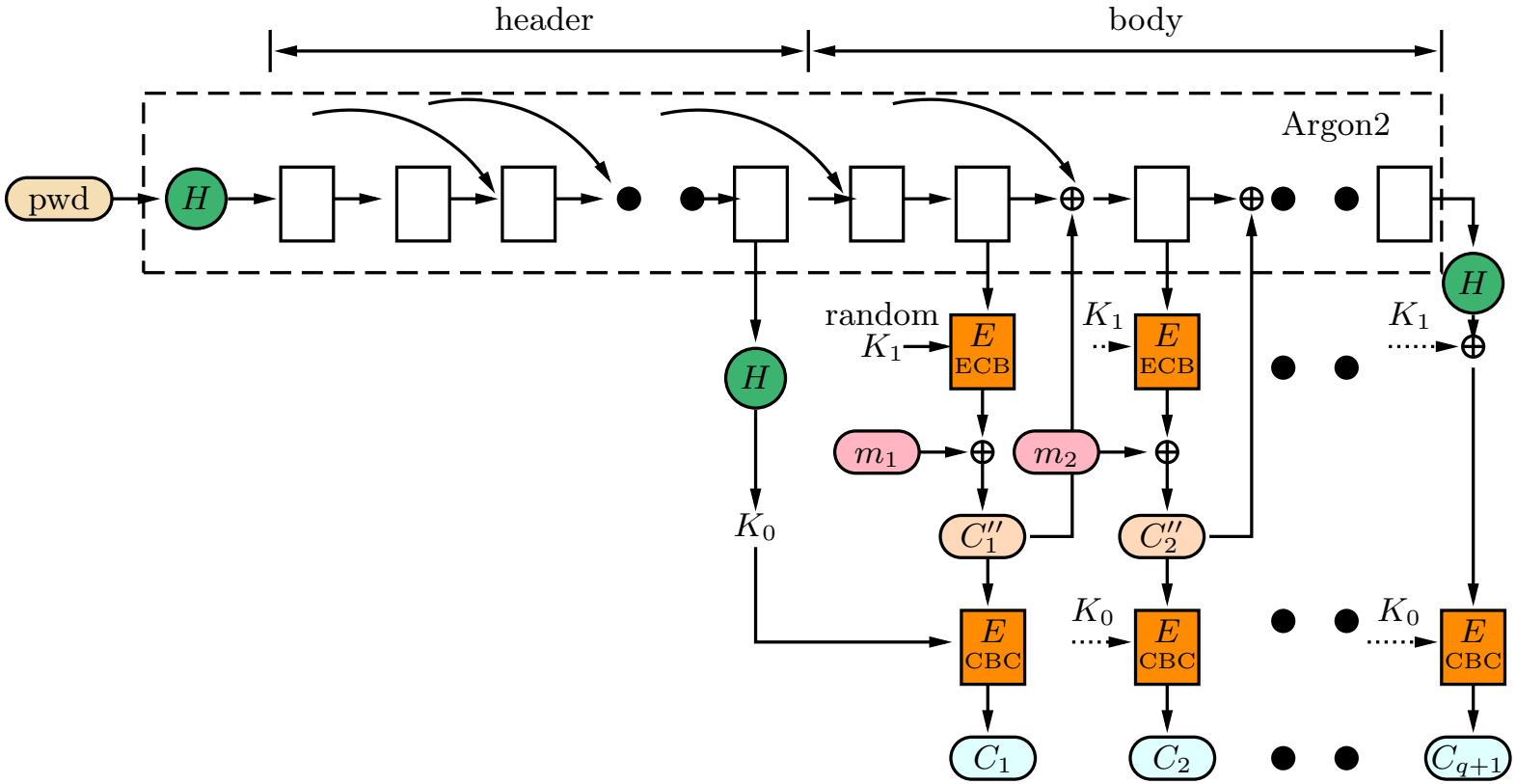
Disk encryption with memory-hard function also based on ANT or another scheme without online decryption:



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

Memory-hard encryption

Disk encryption with memory-hard function also based on ANT or another scheme without online decryption:



If E is AES, we can fill 4-6x more memory at the same time. Even more: encryption of 1-MB chunk in 0.1 second can use 256 MB of RAM.

Egalitarian proofs-of-work

Proof-of-work

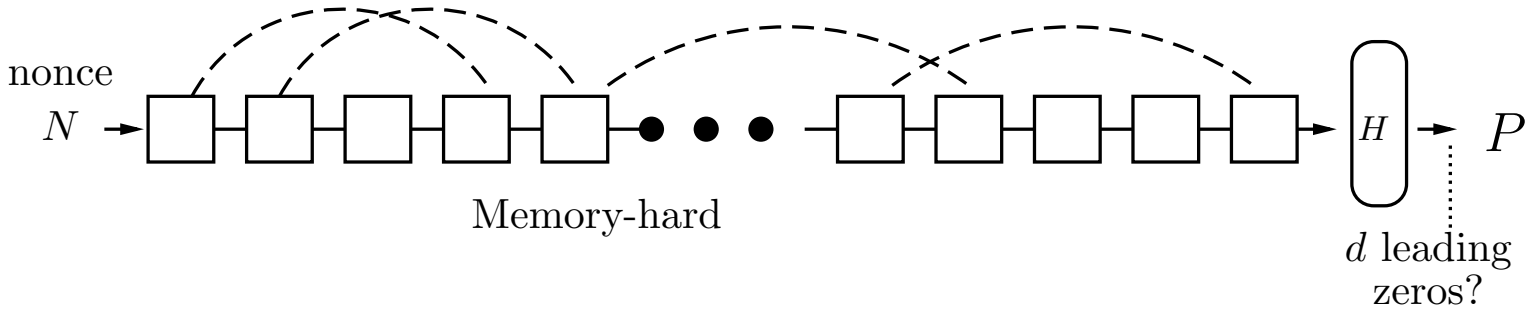
PoW – certificate of having certain computation performed. Used in competitive consensus protocols (Bitcoin) and as a DoS protection (e.g., TLS).
Necessary properties:

1. Asymmetry: hard to compute, easy to verify;
2. Amortization-free: every next solution is as much expensive;
3. If memory-hard – steep time-space tradeoffs;
4. Independent time and memory tune;
5. Optimization-free: best algorithms have been found;
6. Progress-free: chances to find a solution do not accumulate;
7. Parallel-unfriendly: multiple cores bring no cost cut.

Bitcoin's Hashcash misses 3,4; Equihash misses 6 and have problems with 4,7. Proofs-of-Space lack 2,3.

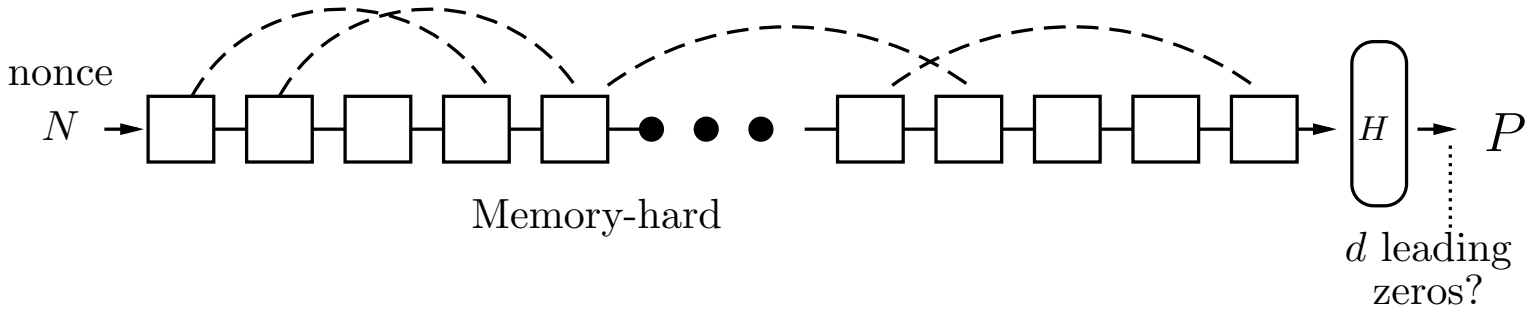
Proof-of-space + difficulty test

Memory-hash function does not work alone: too hard verification.

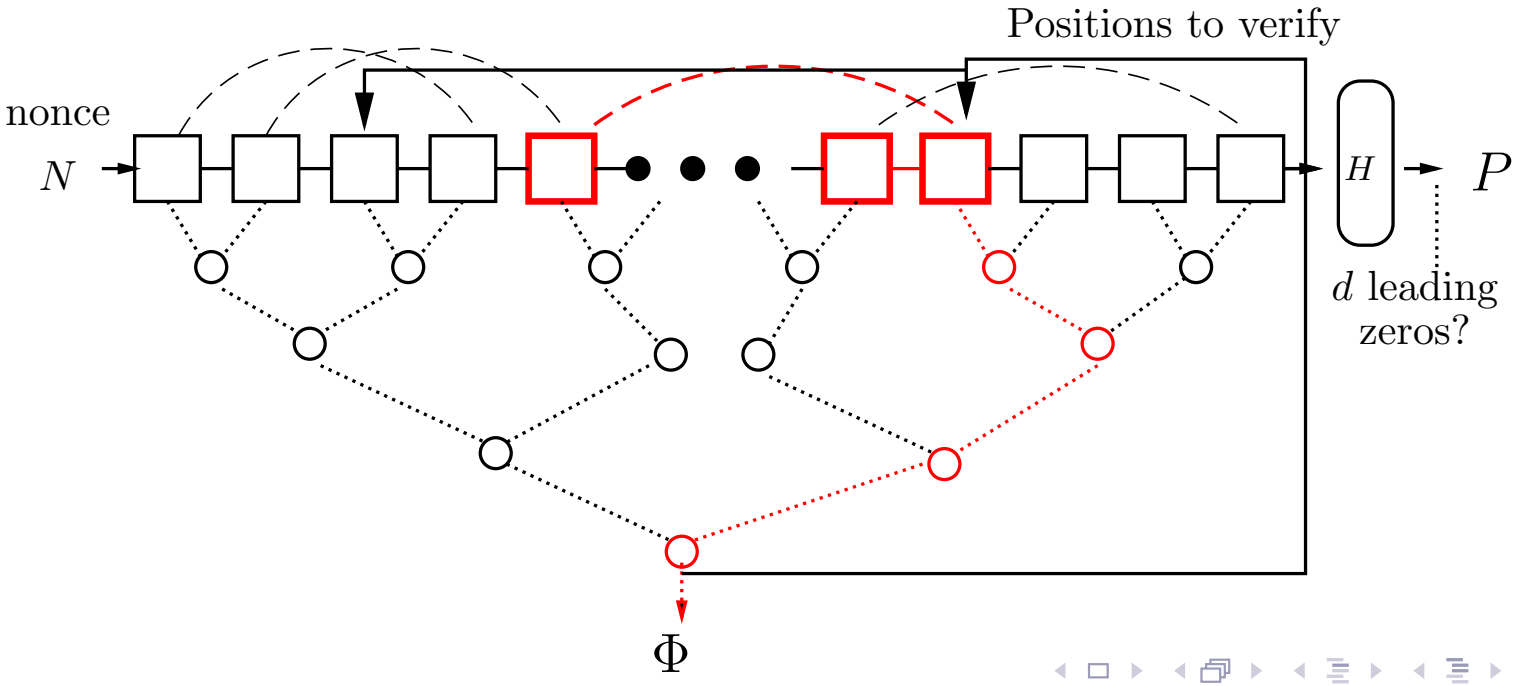


Proof-of-space + difficulty test

Memory-hash function does not work alone: too hard verification.

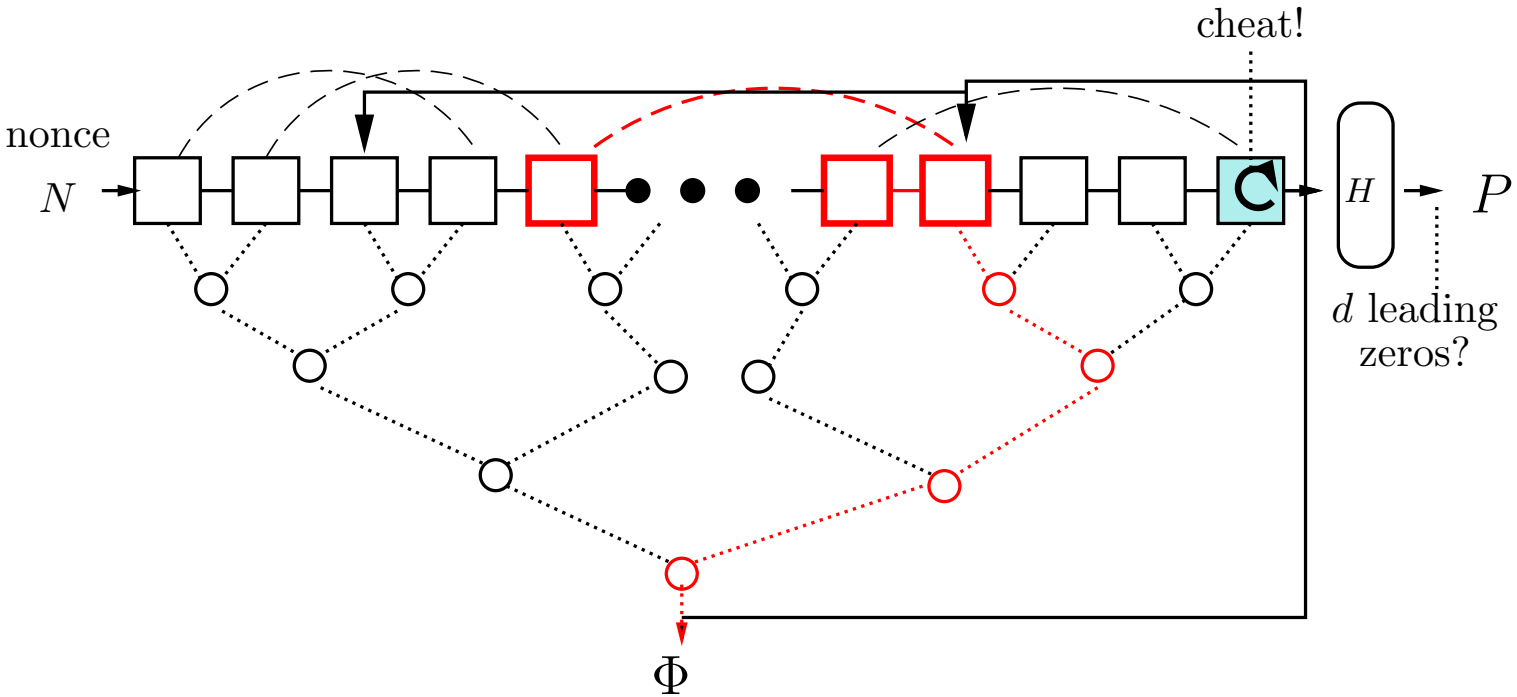


Proof-of-space + difficulty test? Proof: $(P, \Phi,$ block openings$)$.



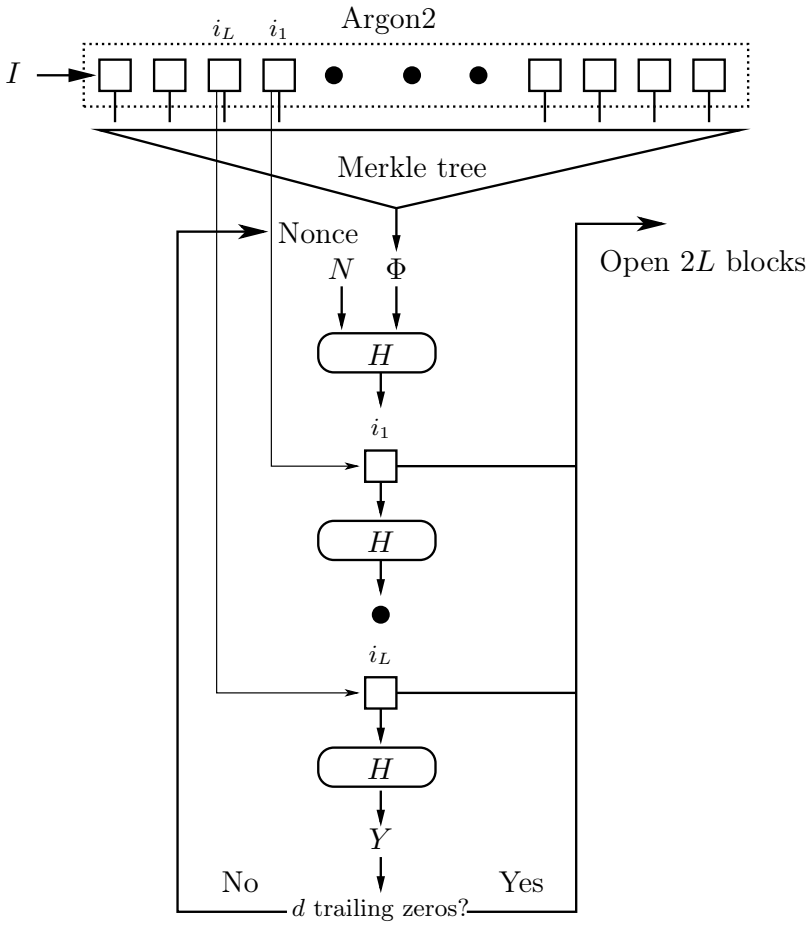
Grinding attack

In proof-of-space + difficulty test modify the last block until the test is passed – so no amortization-free.



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

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



- Compute Argon2;
- Select a nonce;
- Derive opening block indices from the nonce and the Merkle root.
- Check for difficulty. If fail, select a new nonce.

Parallelism inevitable but memory use is intensive and random.

Using Argon2 tradeoffs, MTP on ASICs can have AT cost reduced by at most 12.

New world

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.

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



Use **Egalitarian Computing**