New proofs of work for cryptocurrencies and crypto applications

Alex Biryukov Dmitry Khovratovich

University of Luxembourg

June 2nd, 2015

Proof-of-work

Problem:

- There is some limited public resource (e-mail, web service, crypto money);
- Adversary can emulate multiple (say, billions) users and grab the resource.

Problem:

- There is some limited public resource (e-mail, web service, crypto money);
- Adversary can emulate multiple (say, billions) users and grab the resource.

Solution:

- Require every user to compute some

$$\underbrace{P}_{\text{proof}} = \underbrace{f(\text{Time}, \text{User})}_{\text{work}};$$

- f should be fast enough: a regular user does not spend too much time to compute f (e.g., seconds);
- f should be slow enough: an adversary can not compute too many copies of f;
- It should be easy to verify that P is output of f.

Example

Bitcoin's proof-of-work

$$P = H(\text{Miner}, \text{AttemptNumber})$$

such that

$$P = \underbrace{00\cdots0}_{k \text{ zero bits}} * * * \ldots * .$$

Needs 2^k calls to H on average to find a proof.

Additional properties of PoW

$$\underbrace{P}_{\text{proof}} = \underbrace{f(\text{Time}, \text{User})}_{\text{work}};$$

- *Flexibility.* Computational difficulty can be adjusted easily.
- *Optimization-free.* All possible optimizations and amortizations must be already incorporated in the algorithm.

Proof-of-work in use:

- **Cryptocurrencies**. Currency user earns money if he does certain work and presents a proof:

$$P = H(\mathrm{Miner}, \mathrm{AttemptNumber})$$

H is fast, but it takes time to produce *right* P.

Proof-of-work in use:

- **Cryptocurrencies**. Currency user earns money if he does certain work and presents a proof:

$$P = H(\mathrm{Miner}, \mathrm{AttemptNumber})$$

  H is fast, but it takes time to produce *right* P.

- **Authentication**. User passwords are hashed with moderately hard H and stored:

$$X = H(\mathrm{Password}, \mathrm{Username}).$$

  If X is leaked, brute-forcing passwords takes too much time.

Proof-of-work in use:

- **Cryptocurrencies**. Currency user earns money if he does certain work and presents a proof:

$$P = H(\text{Miner}, \text{AttemptNumber})$$

  H is fast, but it takes time to produce *right* P.

- **Authentication**. User passwords are hashed with moderately hard H and stored:

$$X = H(\text{Password}, \text{Username}).$$

  If X is leaked, brute-forcing passwords takes too much time.

- **Privacy protection**. Secret key for hard-drive encryption is derived from the password:

$$K = H(\text{Password}).$$

New directions

$$\underbrace{P}_{\text{proof}} = \underbrace{f_{\text{Memorysize}}(\text{Time}, \text{User})}_{\text{cost}};$$

Major problem:

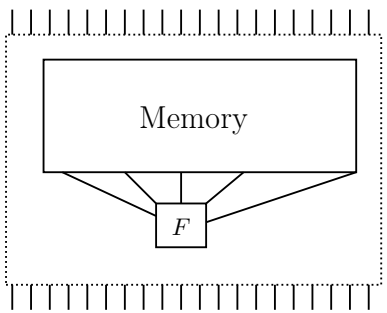
- Work on special hardware is much cheaper than on a desktop;
- Bitcoin PoW: hardware miners need 30,000 times as less energy (in Joules) as desktop miners;
- Password hashing: hardware reduces the password cracking costs by a huge factor.

Proof of cost

$$\underbrace{P}_{\text{proof}} = \underbrace{f(\text{Time}, \text{User})}_{\text{work}};$$

Our approach:

- Switch from work (computation) to generic costs (time-area product);
- Use *memory-hard* functions for proof-of-work: much memory is needed to compute f, and memory reduction is expensive.
- Memory-rich hardware is expensive: no more efficiency gain for rich adversaries.



Password-cracking chip

Further requirements to PoW

Memory-hard proof-of-work

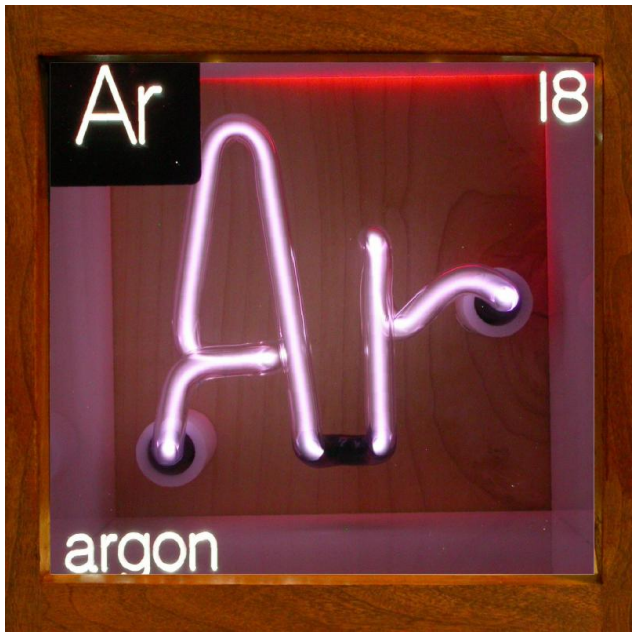
$$\underbrace{P}_{\text{proof}} = \underbrace{f(\text{Time}, \text{User})}_{\text{work}};$$

- *Flexibility.* Memory and time requirements can be adjusted independently.
- *Performance.* Large memory amounts (GBytes) must be handled by an average machine in reasonable time (seconds).
- *Parallelism-free.* Adding computational cores without memory does not speed up the computation.
- *Steep time-space tradeoffs.* Memory reductions are penalized.

Argon2: memory-hard multi tool

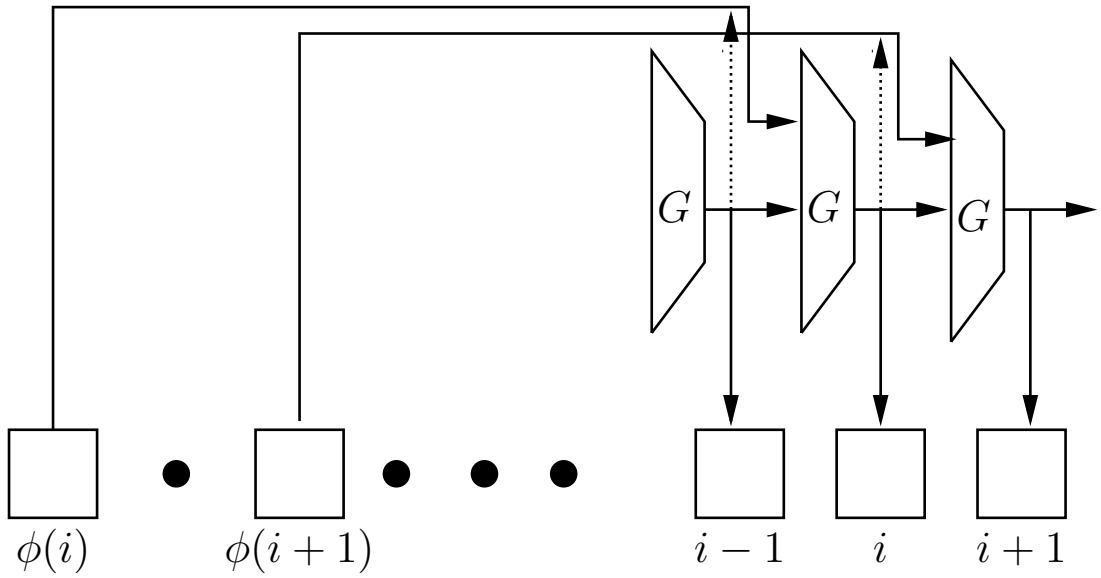
Argon2: memory-hard hash function

Argon — noble gas, which expands to fill all available volume (memory in our case) and can be easily compressed back to a small volume (short hash).



Argon2: memory-hard hash function

Argon2 fills memory in chunks 1 KB wide:



- Every block is a (hash) function of two previous ones;
- Arbitrary memory size;
- Every block depends on some previous one (location is pseudo-random).
- Two instances: Argon2d (faster) and Argon2i (secure against side-channel and memory reduction attacks).

Memory hardness

Computational penalties for memory reductions:

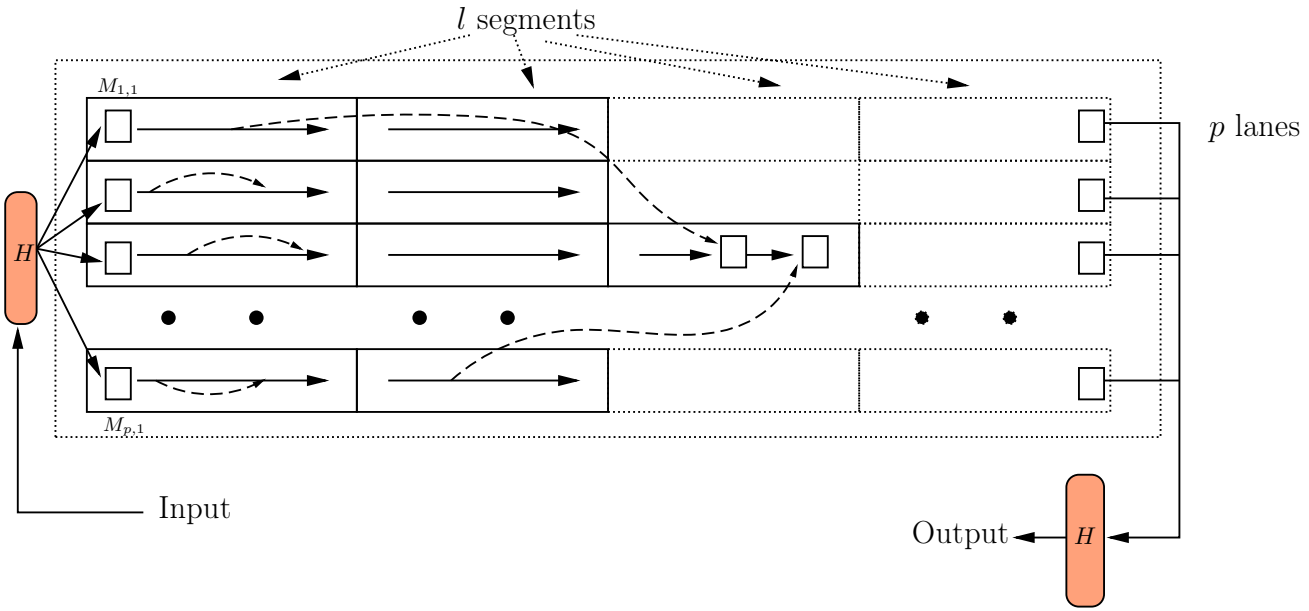
Memory fraction ($1/q$)	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{5}$	$\frac{1}{6}$
1 pass (3 GB/sec)	1.7	3	6.3	16.6	55
2 passes (1.5 GB/sec)	15	410	19300	2^{20}	2^{25}
3 passes (1 GB/sec)	3423	2^{22}	2^{32}		

What if I have just 500 MB instead of 1 GB? For 2-pass Argon2:

- 15x more computation;
- 7x more time (even with parallel cores).

Argon2: parallelism

Parallelism tunable for defender's machine:



Arbitrary number of lanes/threads with regular synchronisation.

Argon2: applications

Authentication backends:

- 0.5 seconds per password on a 2 GHz CPU using 4 cores and 4 GB of RAM.

Argon2: applications

Authentication backends:

- 0.5 seconds per password on a 2 GHz CPU using 4 cores and 4 GB of RAM.

Key derivation for hard-drive encryption on local machine:

- 3 seconds using 2 cores and 6 GB of RAM.

Cryptocurrency proof-of-work:

$$P = \mathrm{Argon2d}(\mathrm{Miner}, \mathrm{AttemptNumber}).$$

How much memory?

Cryptocurrency proof-of-work:

$$P = \mathrm{Argon2d}(\mathrm{Miner}, \mathrm{AttemptNumber}).$$

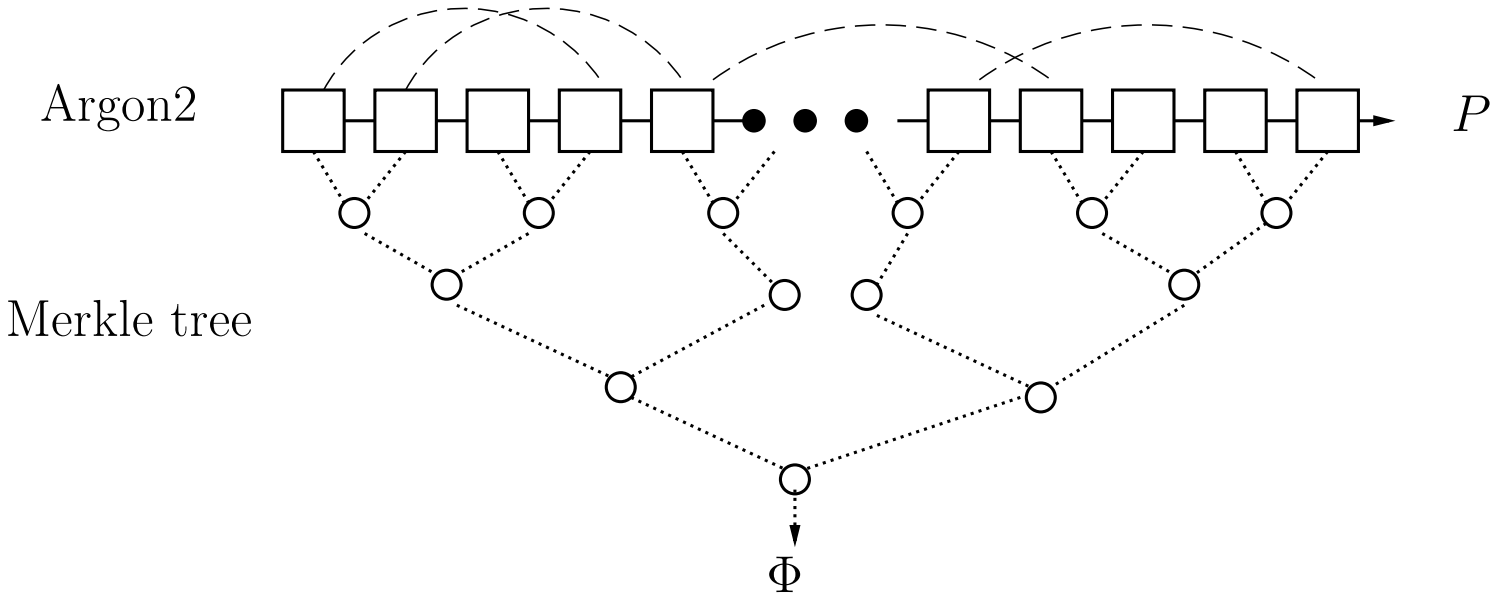
How much memory?

- 100 MB and more — verification takes more than 0.1 seconds, slows down the network;
- less than 100 MB — fits on chip.

Fast verification and memory-hard

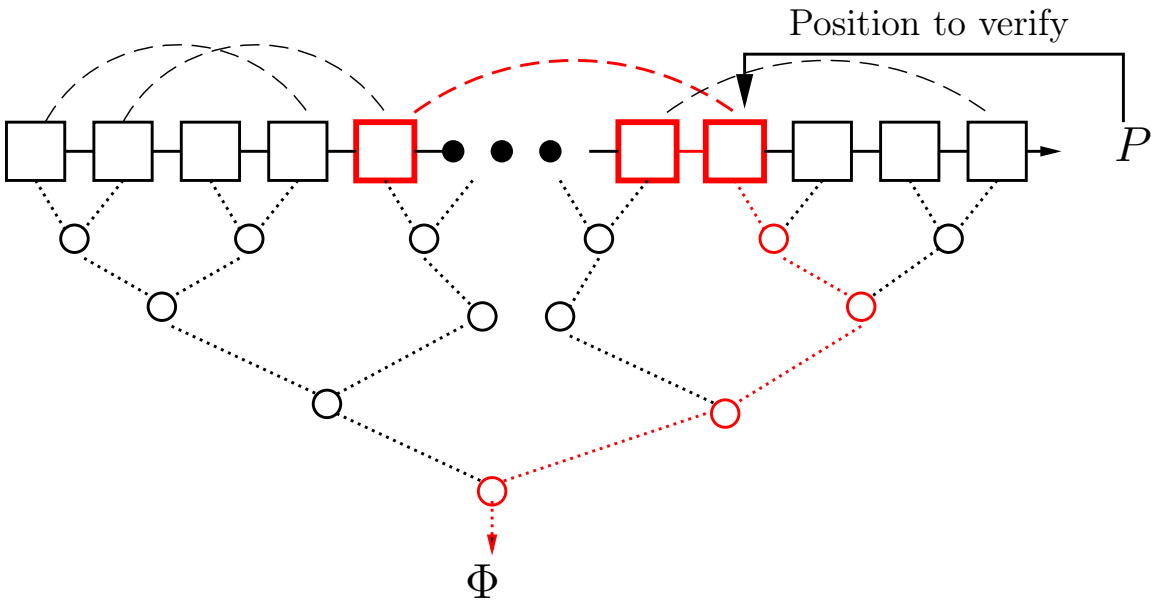
Fast verification feature

How to verify that Argon2 was called?



- P is the output;
- *Merkle tree* with root Φ is built over all memory blocks;
- Prover commits to P and Φ.

Fast verification feature



- P determines 110 positions in the memory to verify;
- Backward links and path in the Merkle tree are provided for each position;
- Cheating probability $< 1/8$;
- Verification is as easy as hashing 500 KB with a regular hash function (milliseconds, memoryless).

https://www.cryptolux.org/index.php/Argon2

https://github.com/khovratovich/Argon2

Questions?