Argon: tradeoff-resilient password hashing scheme

Alex Biryukov Dmitry Khovratovich

University of Luxembourg

Concept of password hashing

1. Client generates password P and sends it to the server;
2. Server generates salt S and computes hash

$$H(P||S),$$

   which is stored along the user's identification data.
3. When the client attempts to login, the supplied password is hashed and checked.

Password can not be recovered if the hash is preimage-resistant, and can not be escrowed if there is no trapdoor.

Primary threat model

We protect from the following attack:

- The hashed passwords are leaked.
- Adversary tries to bruteforce passwords with the help of dictionaries.

Primary threat model

We protect from the following attack:

- The hashed passwords are leaked.
- Adversary tries to bruteforce passwords with the help of dictionaries.

However, we explicitly do not protect from:

- Adversaries that have access to the server during hashing (this includes cache-timing, power analysis, acoustic and other side-channel attacks).
- Adversaries that can affect the server's hardware and software behaviour (fault attacks, salt generation attacks, etc.).

In rare cases when these threats are relevant, stored passwords are not the biggest concern.

Primary threat model

Typical attack:

- The hashed passwords are leaked.
- Adversary tries to bruteforce passwords with the help of dictionaries etc.

Primary threat model

Typical attack:

- The hashed passwords are leaked.
- Adversary tries to bruteforce passwords with the help of dictionaries etc.

Countermeasures:

- Unique salts;
- Increased computational cost of the hash function (analogous to *proof-of-work*).

Switching to new architectures

Adversaries are tempted to brute-force on the most efficient hardware (not CPU, but GPUs, or FPGA, or dedicated ASICs). Electricity and hardware are the dominating costs.

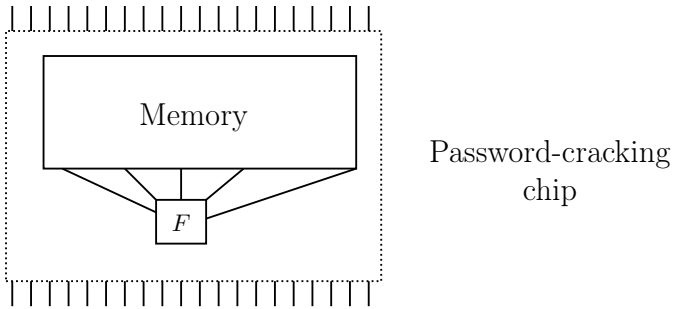
To understand the efficiency of other architectures, we turn to cryptocurrency hardware `https://en.bitcoin.it/wiki/Mining_hardware_comparison`:

- Bitcoin mining on Intel Core computes 2^{17} hashes per joule (=watt*sec).
- Bitcoin mining on the best ASICs does 2^{32} hashes per joule.

Memoryless computations are about 30000 times as cheap on ASICs as on typical server's hardware.

Memory-demanding computations

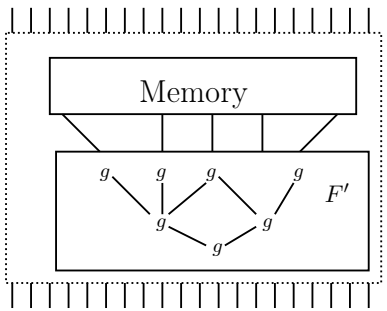
Situation is different when some memory is required:



In a straightforward ASIC implementation of a memory-demanding scheme the memory part consumes most electricity.

Computation-memory tradeoff

An adversary is tempted to trade the memory area for the computation area.



Password-cracking chip

The enlarged computational cores can be pipelined and do not affect the overall throughput.

Therefore, a tradeoff

$$\mathrm{Time} \cdot \mathrm{Memory} = \mathrm{const.}$$

allows an attacker to reduce the memory 100/1000-fold and still win.

Therefore, a tradeoff

$$\mathrm{Time} \cdot \mathrm{Memory} = \mathrm{const.}$$

allows an attacker to reduce the memory 100/1000-fold and still win.

Scrypt allows for such tradeoffs.

Another problem: complexity

Scrypt:

$$H(\cdot) = MFcrypt_{HMAC_{SHA256}, ROMix_{BlockMix_{Salsa20/8}}}(\cdot)$$

Clearly, too many components.

Need for a new scheme

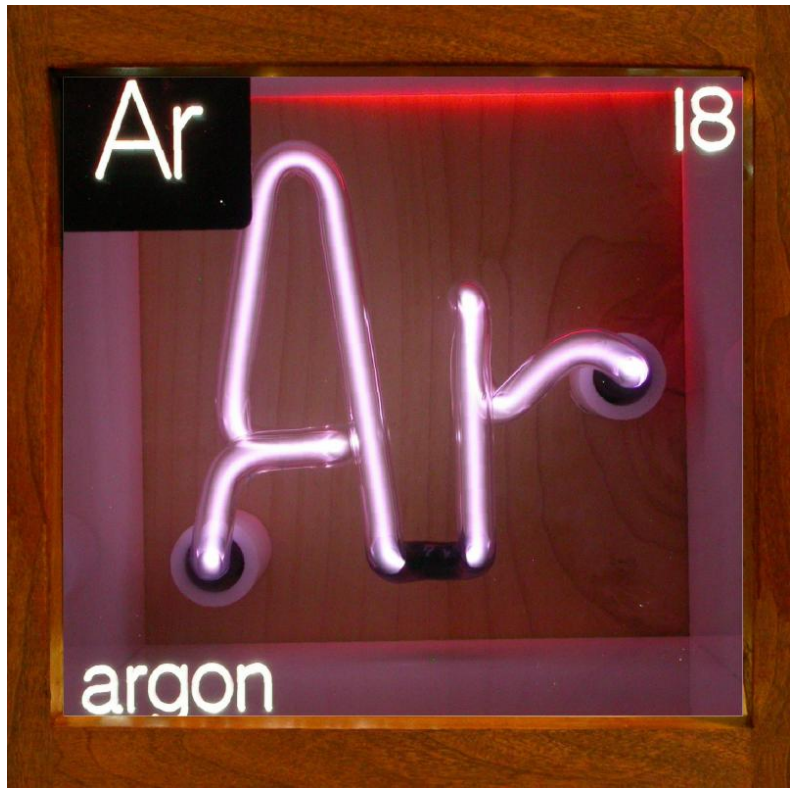
Major goals

Goals:

- **Tradeoff resilience:** prohibitive penalties for memory-reducing attackers.
- **Speed:** faster than scrypt, securely filling hundreds of MBytes of RAM per second.
- **Simplicity:** Minimum of external components, rational design, easy analysis. Scheme should fit a single picture.

Design of Argon

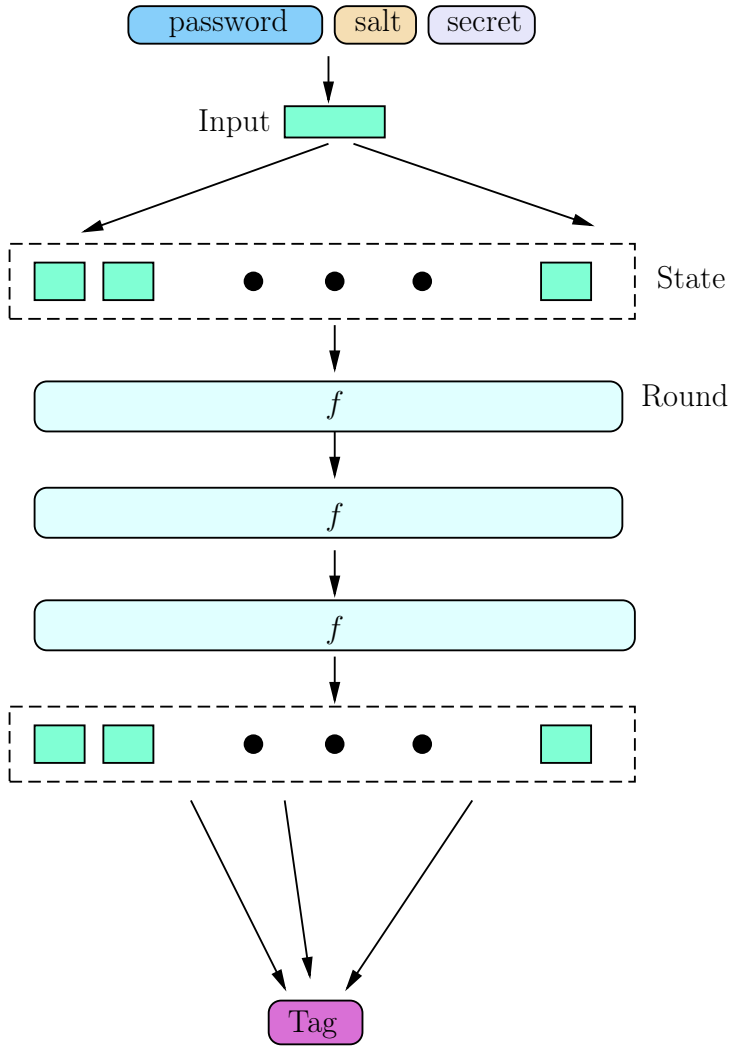
Argon — noble gas, which expands to fill all available volume (memory in our case) and can be easily compressed back to a small volume (short hash).



Design: overview

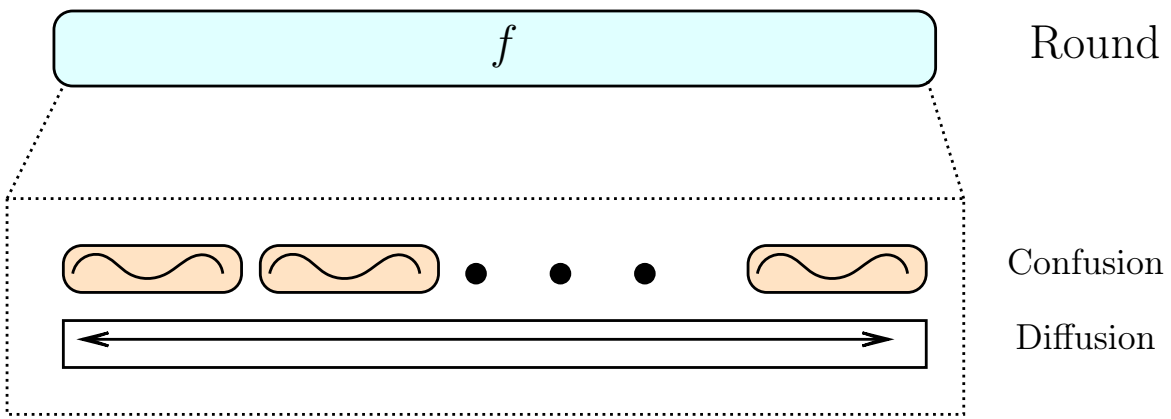
Input: salt, password, secret, all lengths, all costs. Fits into a short string.

1. Expand to the entire memory available. No cryptography involved in this step.
2. Apply a sequence of memory-hard transformations (rounds).
3. Absorb the entire state into a small tag.



Ideas:

1. Memory block = Input block + counter.
2. L rounds:
   - Confusion part: apply cryptographic transformations to a small group of blocks;
   - Diffusion part: data-dependent block shuffling among the groups.



3. XOR the entire state into a small tag.

Ideas for confusion part

In the confusion part we first need a building block — fast transformation $\mathcal{F}$.

Candidates:

- ARX (Addition-Rotation-XOR). Good but existing designs are ad-hoc and complicated. Fastest one runs at 4 cycles per byte.
- AES with AES-NI instructions. Very fast (0.6 cpb if pipelined), sustained decades of cryptanalysis, simple.

Ideas for confusion part

In the confusion part we first need a building block — fast transformation $\mathcal{F}$.

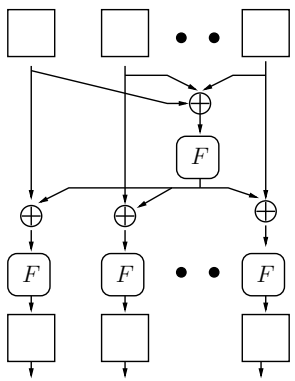
Candidates:

- ARX (Addition-Rotation-XOR). Good but existing designs are ad-hoc and complicated. Fastest one runs at 4 cycles per byte.
- AES with AES-NI instructions. Very fast (0.6 cpb if pipelined), sustained decades of cryptanalysis, simple.

Decision: reduced 5-round AES-128 with a fixed key.

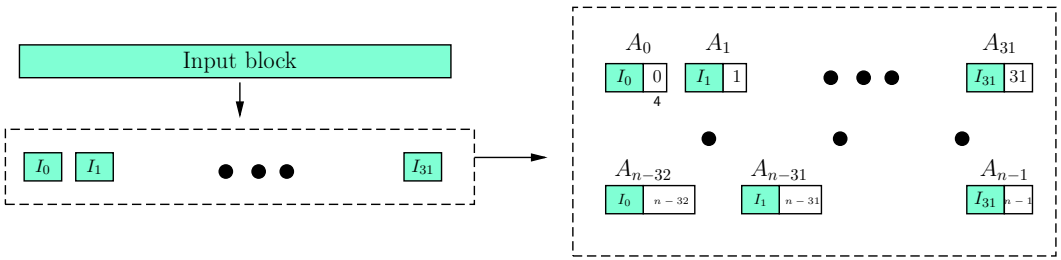
- Twice as fast as regular AES-128;
- Permutation with good cryptographic properties.

Updating several blocks:

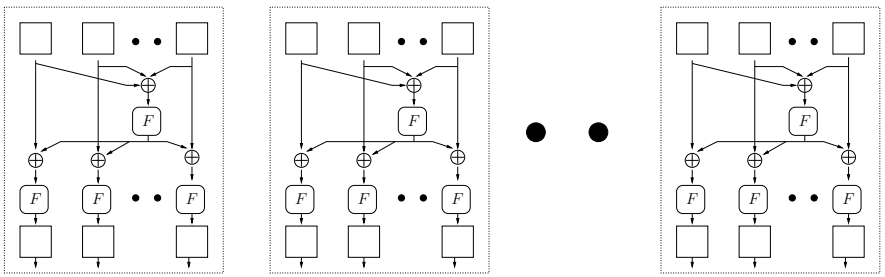


First attempt:

1. Memory block = Input block + counter:



2. L rounds:



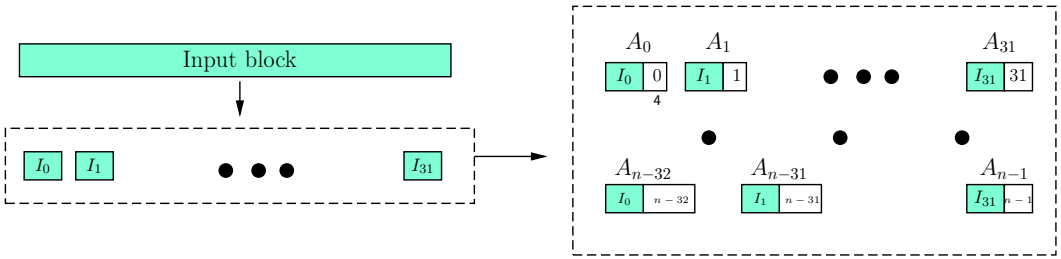
   - SubGroups:
   - Diffusion part: *sorting*.

3. XOR the entire state into a small tag.

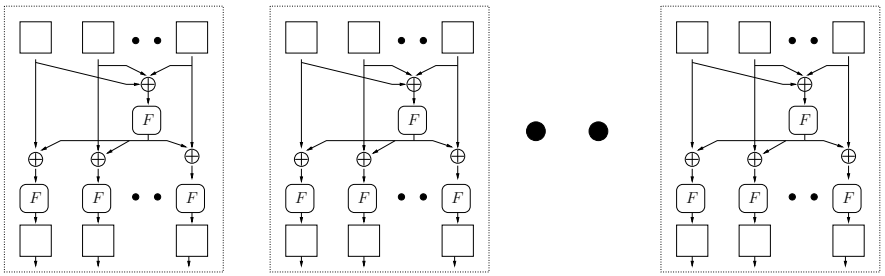
First attempt

First attempt:

1. Memory block = Input block + counter:



2. L rounds:



   - SubGroups:
   - Diffusion part: *sorting*.

3. XOR the entire state into a small tag.

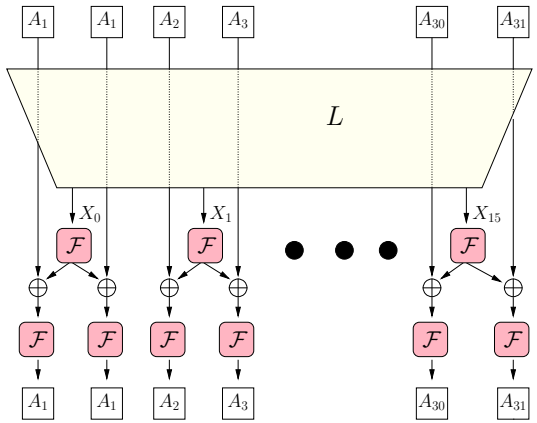
Problems:

- Output block of a small group to depend on few input blocks;
- Large groups allow to store $\mathcal{F}(\bigoplus_i A_i))$ in memory;
- Sorting is too slow for 2^{20} blocks or more.

Second attempt

Second attempt:

1. Memory block = Input block + counter.
2. L rounds:
   - SubGroups: more blocks are inputs to $\mathcal{F}$



   - Shuffle: the RC4 permutation

```
j=0
for each i
   j+=S[i]
   swap(S[i],S[j])
```

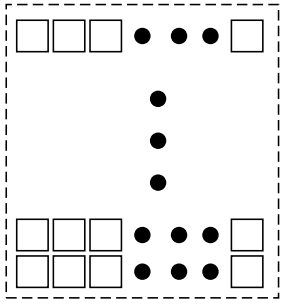
3. XOR the entire state into a small tag.

Problems:

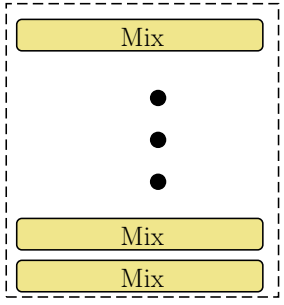
- Shuffle is not parallelizable.

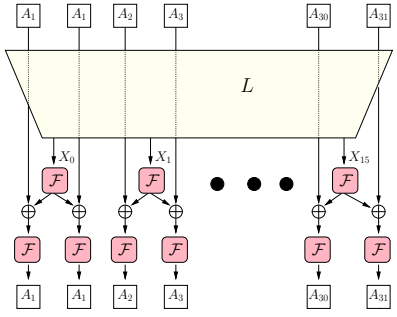
Final attempt

State is a rectangle with rows (groups) and columns (slices):

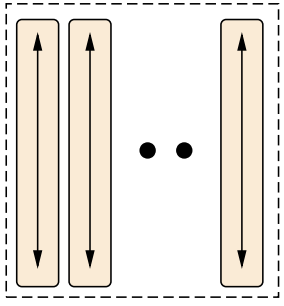


SubGroups:





ShuffleSlices: permutation on slices



```
j=0
for each i
    j+=S[i]
    swap(S[i],S[j])
```

Both SubGroups and ShuffleSlices can be parallelized (up to 32 threads).

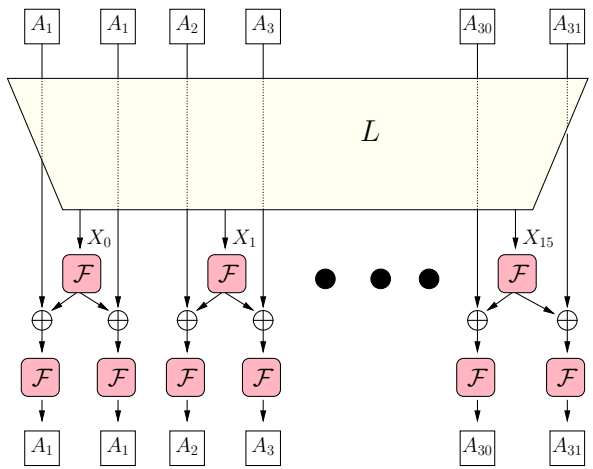
Design of SubGroups

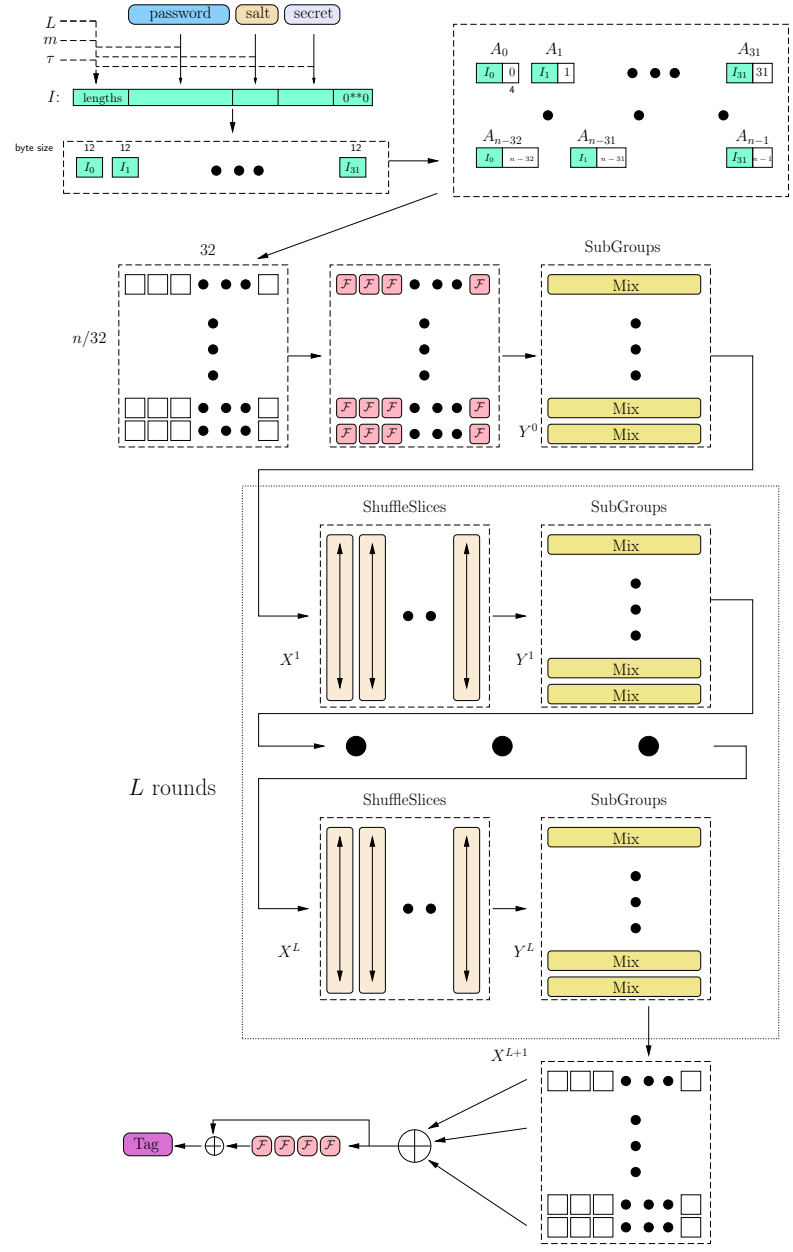
Requirements:

- One input block should affect several output blocks;
- Recomputing an output block should require storing/recomputing some d blocks or internal variables.
- Fast on typical server hardware;
- Parallellizm.

Solution:

- Inputs to intermediate $\mathcal{F}$'s are linear functions L_i;
- When viewed as boolean vectors, L_i form a linear code with distance 8 (Reed-Muller code RM(2,5)).



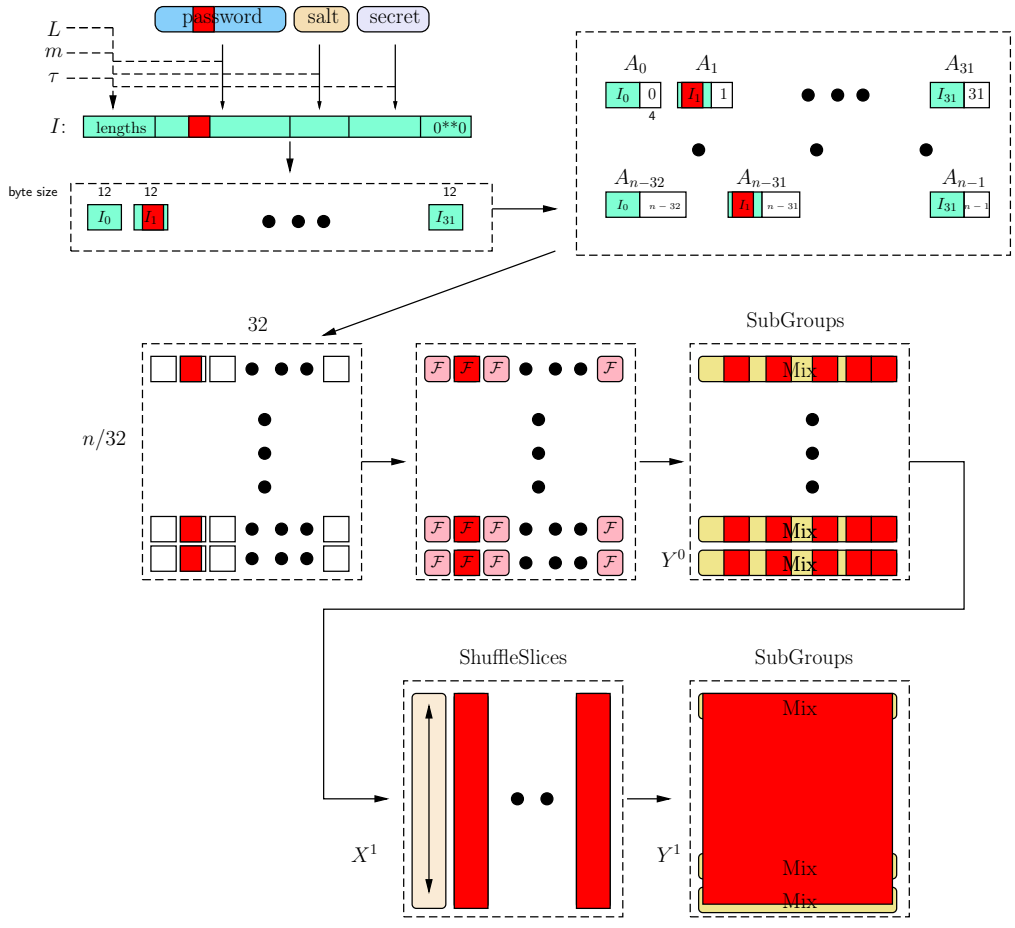

password
salt
secret
L
m
τ
I:
lengths
0**0
byte size
12
12
12
I_0
I_1
I_{31}
A_0
A_1
A_{31}
A_{n-32}
A_{n-31}
A_{n-1}
32
$n/32$
SubGroups
Mix
Y^0
ShuffleSlices
X^1
Y^1
L rounds
X^L
Y^L
X^{L+1}
Tag

Analysis of Argon

Diffusion properties

When a single password byte changes:

1. One block is changed;
2. At least 6 blocks in each group are affected;
3. Second SubGroups transformation activates all the blocks.



Tradeoff analysis

When an attacker uses less memory, he has to recompute some elements.

What can be stored:

- ShuffleSlices permutations ($\frac{m-9}{128}$ for 2^m bytes of memory per level: from $\frac{1}{6}$ to $\frac{1}{2}$ of all memory for $L = 3$);
- Outputs of middle $\mathcal{F}$ in SubGroups ($\frac{1}{2}$ of total memory per level).

One can store a subset of outputs/permutations as well.

When only permutations are stored ($L = 3$):

Memory total	64 KB	1 MB	16 MB	256 MB	1 GB
Memory used	10 KB	250 KB	5 MB	114 MB	500 MB
Penalty factor	190				

Penalty factors for larger amounts of memory ($L = 3$):

Regular memory Attacker's fraction \	128 KB	1 MB	16 MB	128 MB	1 GB
$\frac{1}{2}$	91	112	139	160	180
$\frac{1}{4}$	164	314	2^{18}	2^{26}	2^{34}
$\frac{1}{8}$	6085	2^{20}	2^{31}	2^{36}	2^{47}

Thus highest (claimed) tradeoff resilience among PHC candidates.

Performance

Argon runs fast on multi-core CPUs with AES instructions.

Pre-optimized version on Intel(R) Core(TM) i7-2760QM CPU @ 2.40GHz (Quad Core):

MBytes used	1	16	128	1024
Cycles per RAM byte	8.2	5.4	8.1	9
Threads	16	8	4	8

Possible extensions

Extensions:

- Reducing L to 2: 1.5x further increase in speed.
- Other permutations: Photon, Blake2, Spongent, Quark, Keccak, etc.
- Variable password/salt length.