Tradeoff cryptanalysis of password hashing schemes

Alex Biryukov Dmitry Khovratovich Johann Grozschaedl

University of Luxembourg

PasswordsCon'14, Las Vegas

August 5th, 2014

Passwords

Password evolution

Multi-user environments traditionally employ password-based authentication:

- User is given a login l and a password p (or he generates them);
- Salt s and $H(s, p))$ are generated and stored;
- When a user logs in, he sends (l, p) to the server;
- Server matches $(l, H(s, p))$ with its database $\{l_i, H(s_i, p_i)\}$.

Password evolution

Multi-user environments traditionally employ password-based authentication:

- User is given a login l and a password p (or he generates them);
- Salt s and $H(s, p))$ are generated and stored;
- When a user logs in, he sends (l, p) to the server;
- Server matches $(l, H(s, p))$ with its database $\{l_i, H(s_i, p_i)\}$.

Iterating H many times makes the password cracking harder, but how hard?

Switch architecture problem

Whereas hashing is done on server hardware (not much different from desktop), an adversary may employ graphic cards (GPU) and dedicated hardware (FPGA or even custom ASICs), where each password computation is much cheaper.

Switch architecture problem

Whereas hashing is done on server hardware (not much different from desktop), an adversary may employ graphic cards (GPU) and dedicated hardware (FPGA or even custom ASICs), where each password computation is much cheaper.

Solution: use a hash function, which takes similar time to compute on different architectures. This suggests *memory-hard functions*, which intensively use a large amount of memory.

ASIC advantage

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

- Intel Core computes 2^{17} hashes per joule (=watt*sec).
- Best AMD GPU compute 2^{22} hashes per joule (=watt*sec).
- Best FPGA compute 2^{25} hashes per joule.

ASIC advantage

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

- Intel Core computes 2^{17} hashes per joule (=watt*sec).
- Best AMD GPU compute 2^{22} hashes per joule (=watt*sec).
- Best FPGA compute 2^{25} hashes per joule.
- Best ASICs compute 2^{32} hashes per joule.

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

What about memory-intensive schemes?

Theory of memory-hard functions

Back in 1970s, computer scientists explored functions computed on multitape Turing machines.

Theory of memory-hard functions

Back in 1970s, computer scientists explored functions computed on multitape Turing machines.

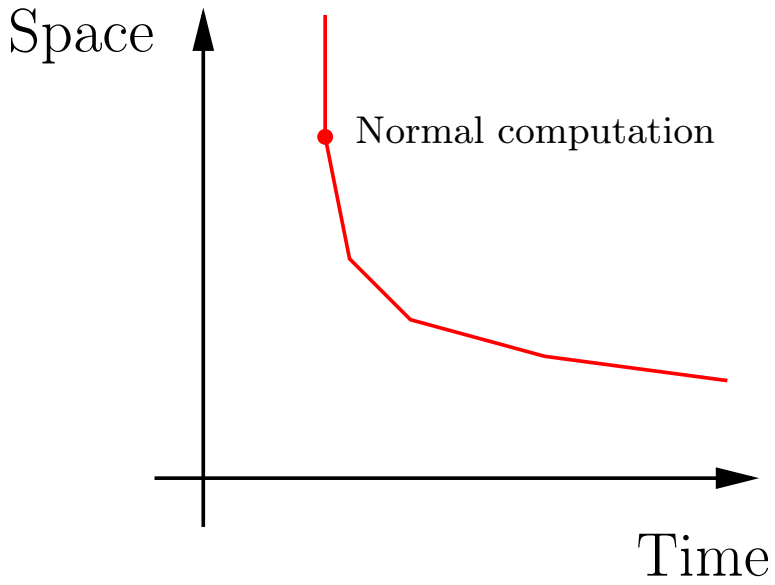
What can be computed in time T, can be computed in space $\frac{T}{\log T}$, and this bound is tight [HPV'77, PTC'77].

There are provably hard functions (for $\frac{1}{\log T}$ memory reduction and asymptotically).
How high can be computational penalties for $\frac{1}{4}$ memory reduction? Unknown.

Tradeoff cryptanalysis

Time-memory tradeoff

Tradeoff: *how much time is needed to compute the same function with less memory.*



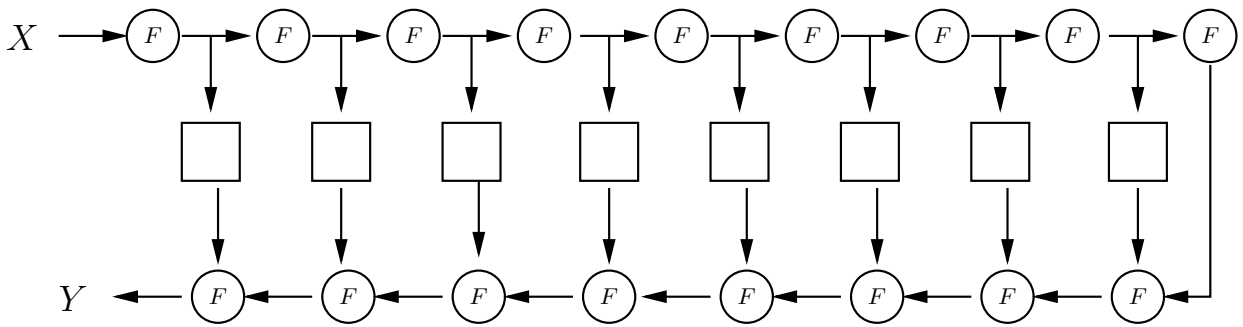
- There could be other tradeoffs: computation-memory, energy-memory (for implementations), time-area, etc.

Toy example

Toy example

Hash function with two iterations over memory.

- $V_i = F(V_{i-1})$;
- $V'_N = V_N$;
- $V'_i = F(V'_{i+1}||V_i)$.

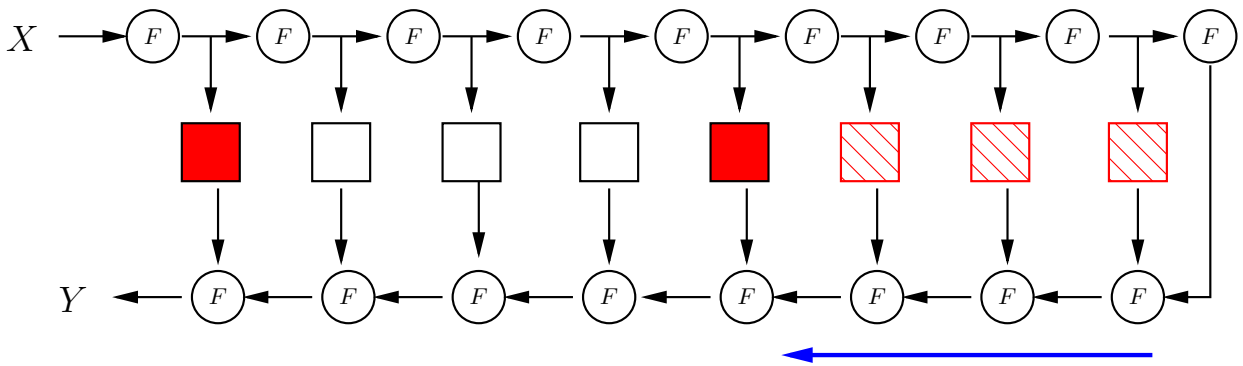


Trivial tradeoff

Compute the hash using $\frac{N}{m} + m$ memory units and calling the hash function $3N$ times instead of $2N$:

- Store every m-th block;
- When entering a new interval, precompute its m inputs.

Optimal point is $m = \sqrt{N}$.

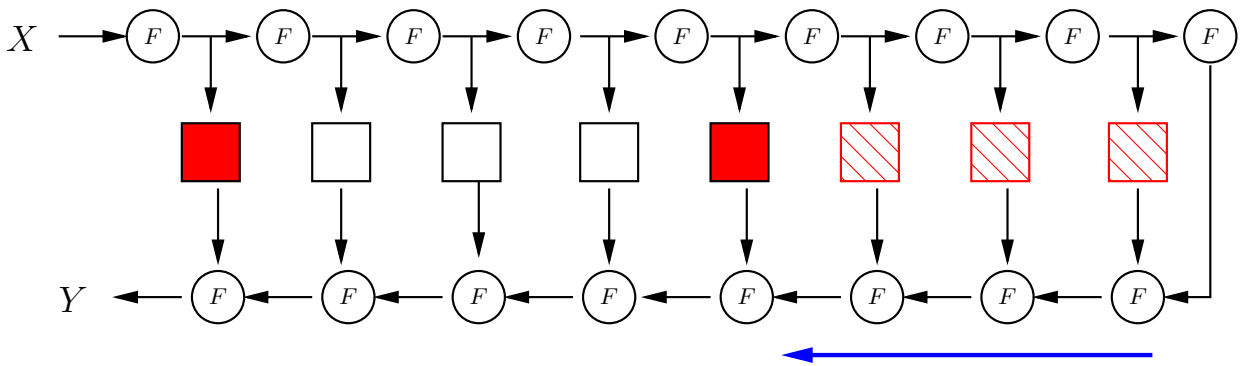


Trivial tradeoff

Compute the hash using $\frac{N}{m} + m$ memory units and calling the hash function $3N$ times instead of $2N$:

- Store every m-th block;
- When entering a new interval, precompute its m inputs.

Optimal point is $m = \sqrt{N}$.



If multiple hash cores are available, the scheme latency is still $2N$ hash calls, since all the addresses are known and data can be prefetched.

Scrypt

Scrypt [Percival'09]:

- Sequential initialization: $X_i \leftarrow H(X_{i-1})$;
- Random processing:

$$\text{for } 1 \leq i \leq N$$
$$A \leftarrow H(A \oplus X_A).$$

Problems:

- Too many parameters and subfunctions;
- Allows trivial tradeoff:

$$\text{Memory} \times \text{Time} = O(N^2);$$

PHC

Password Hashing Competition (2014-2015): struggle to find faster, more secure, more universal schemes.

- 22 schemes in competition;
- Vast majority claim resilience to GPU/ASIC cracking;
- Only a few really tried to attack their schemes (standard practice in cryptography designs);

PHC

Password Hashing Competition (2014-2015): struggle to find faster, more secure, more universal schemes.

- 22 schemes in competition;
- Vast majority claim resilience to GPU/ASIC cracking;
- Only a few really tried to attack their schemes (standard practice in cryptography designs);
- These attacks can be much improved, as we see later;
- And we will see how ASIC-equipped adversaries can exploit them.

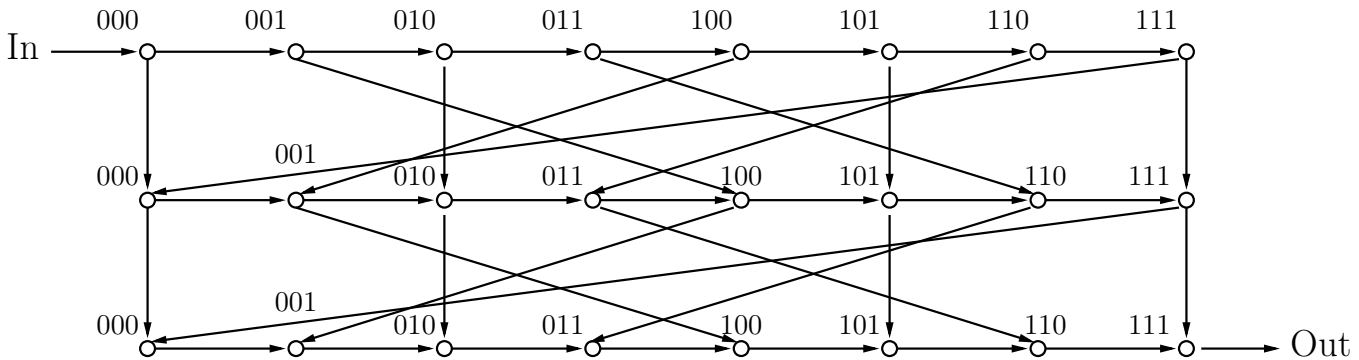
We considered three schemes, which have come out of academic crypto-community and have clear documentation.

Catena

Catena [Forler-Lucks-Wenzel'14]:

- Stack of λ *bit-reversal* permutations ($\lambda = 3, 4$):

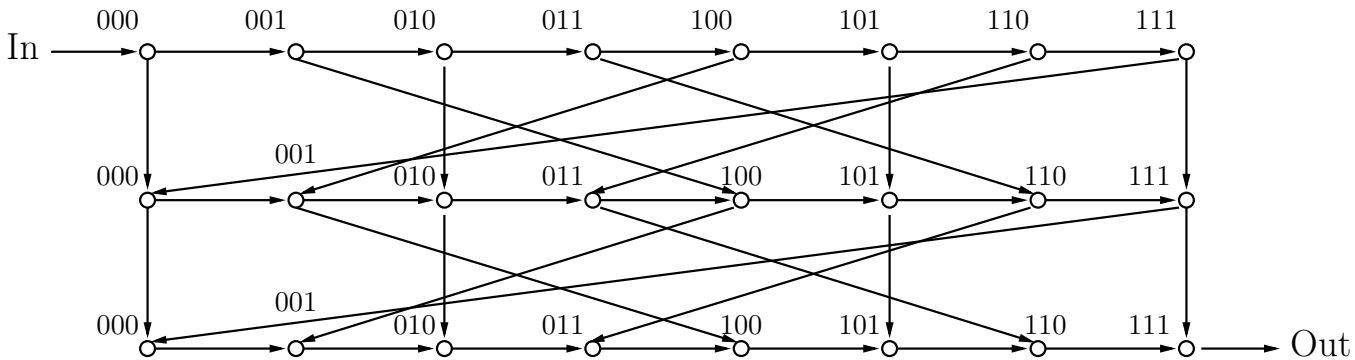
$$V^L[ABC] = H(V^L[ABC - 1], V^{L-1}[\overline{C}\,\overline{B}\,\overline{A}]).$$



Catena [Forler-Lucks-Wenzel'14]:

- Stack of λ *bit-reversal* permutations ($\lambda = 3, 4$):

$$V^L[ABC] = H(V^L[ABC - 1], V^{L-1}[\overline{C}\,\overline{B}\,\overline{A}]).$$



- Full-round hash function (Blake2);
- Proof of tradeoff resilience (extension of Lengauer-Tarjan proof for $\lambda = 1$):

$$S^\lambda T = \Omega(N^{\lambda+1})$$

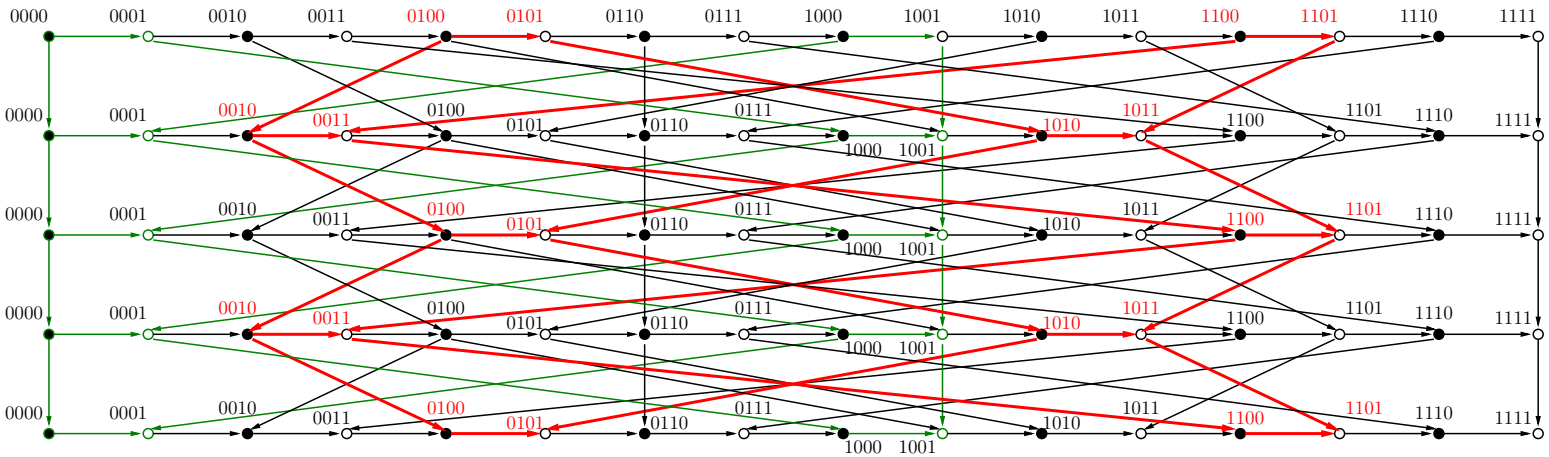
Memory fraction $\frac{1}{q}$ should imply penalty q^λ.

Observation

Apparently, the proof has a flaw.

Observation

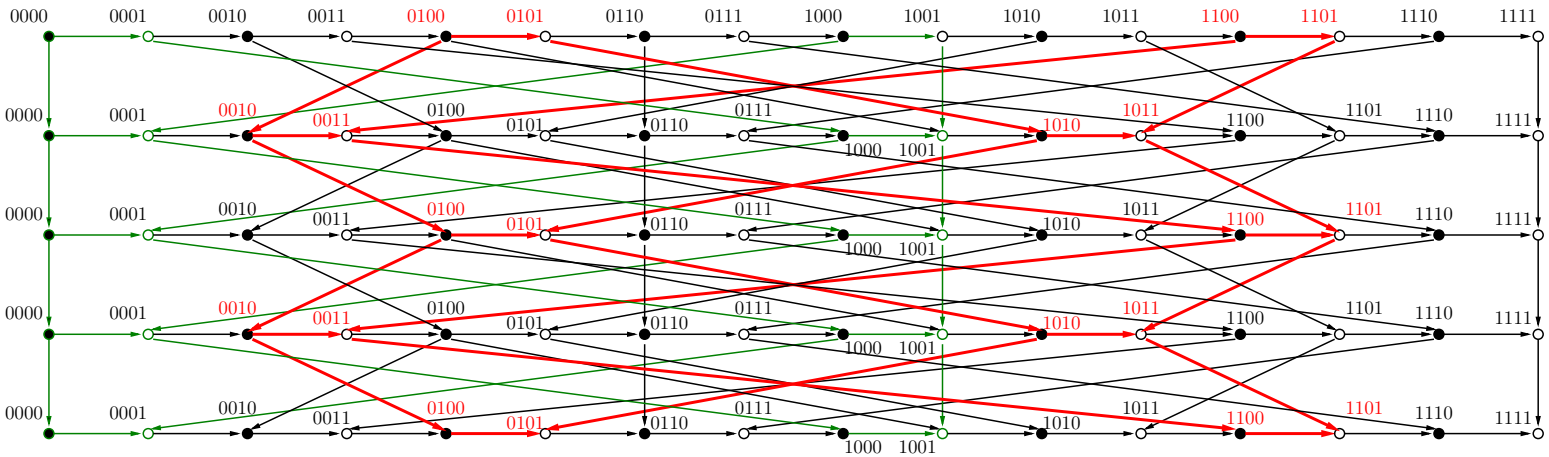
Apparently, the proof has a flaw.



- Consider vertices $[AB0], [AB1], [AB2], \ldots$, where A and $0, 1, 2$ are k-bit values;

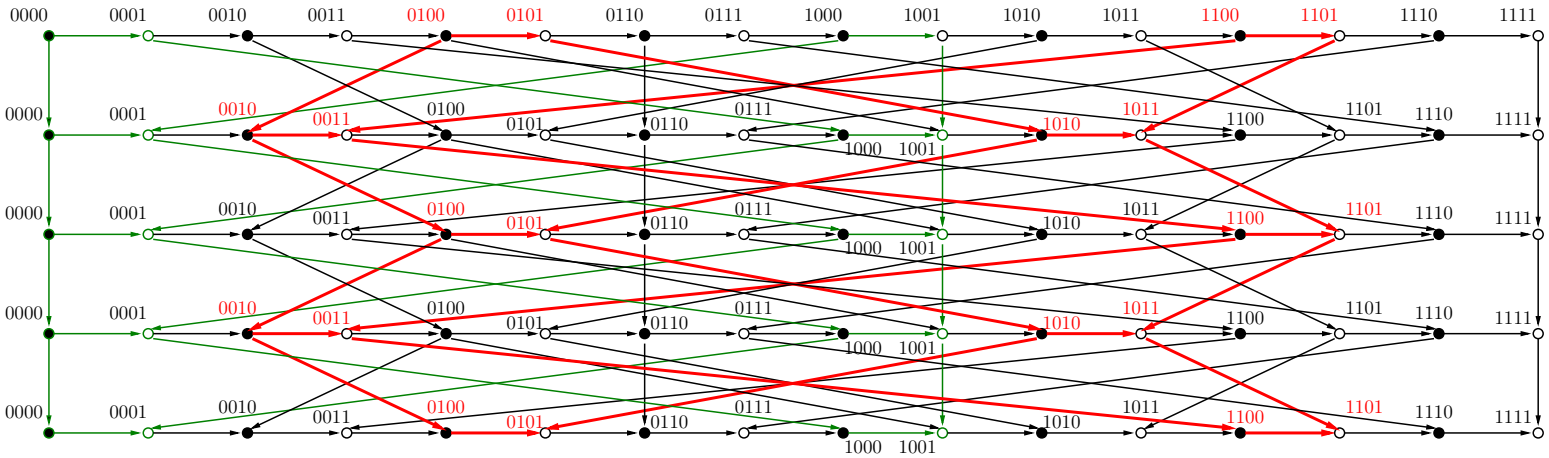
Observation

Apparently, the proof has a flaw.



- Consider vertices $[AB0], [AB1], [AB2], \ldots$, where A and $0, 1, 2$ are k-bit values;
- To compute $[ABC]$ at level T (C is also k-bit), we need $[\overline{C}\,\overline{B}\,\overline{A}]$ at level $T-1$;
- $[\overline{C}\,\overline{B}\,\overline{A}]$ refers to $[ABC]$ at level $T-2$.
- Note that the middle part is either B or $\overline{B}$.

Tradeoff cryptanalysis



Efficient computation of $[AB*]$ at level 4:

- Suppose that we have stored all vertices $[**0]$ at all levels (2^{n-k} vertices per level);
- Compute $[*B*]$ at level 0 (2^{2k} steps);
- Use these values to compute $[*\overline{B}*]$ at level 1 (2^{2k} steps);
- Use these values to compute $[*B*]$ at level 2 (2^{2k} steps);
- Use these values to compute $[*\overline{B}\,\overline{A}]$ at level 3 ($\overline{A}2^{k}$ steps);
- Use these values to compute $[AB*]$ at level 4 (2^{k} steps).

In total $3.5 \cdot 2^{2k} + 2^{k}$ hashes for 2^{k} vertices.

Eventually we have the following penalties for $l < n/3 - 2$:

Memory fraction	Catena-3	Catena-4
	Penalty	
$\frac{1}{2}$	7.4	13.1
$\frac{1}{4}$	15.5	26.1
$\frac{1}{8}$	30.1	51.5
$\frac{1}{2^l}$	$2^{l+1.9}$	$2^{l+2.7}$

Eventually we have the following penalties for $l < n/3 - 2$:

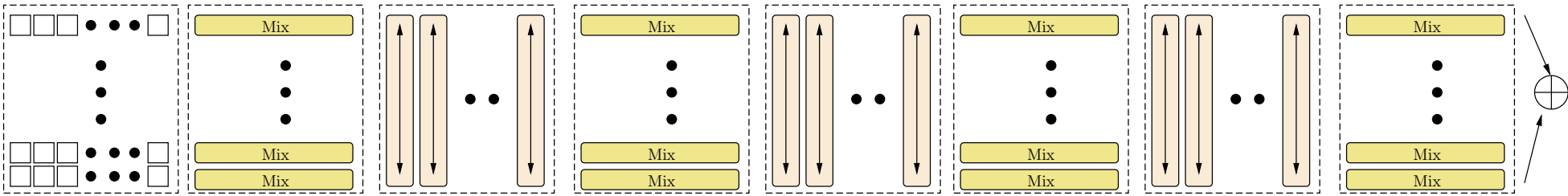
Memory fraction	Catena-3	Catena-4
	Penalty	
$\frac{1}{2}$	7.4	13.1
$\frac{1}{4}$	15.5	26.1
$\frac{1}{8}$	30.1	51.5
$\frac{1}{2^l}$	$2^{l+1.9}$	$2^{l+2.7}$

So the penalty is $4q$ for memory fraction $\frac{1}{q}$. Tradeoff for Catena-3:

$$MT = 16N^2;$$

Argon

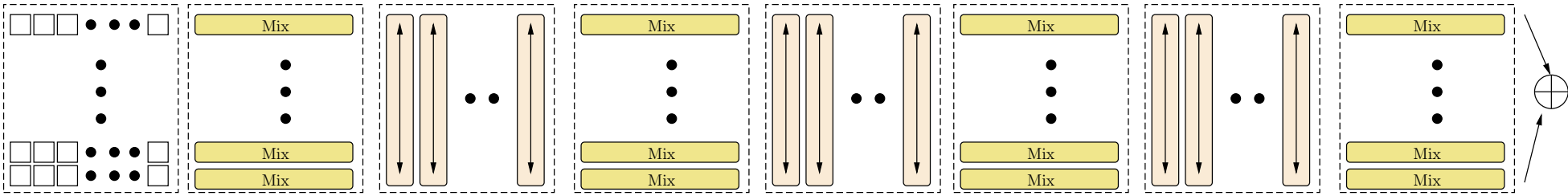
Argon [Biryukov-Khovratovich'14]:



Blockcipher-based design:

- $n \times 32$-matrix of 16-byte blocks;
- Row-wise nonlinear transformation (48 reduced AES cores and a linear layer) with guaranteed branch number (at least 8 inputs for 1 output);
- Column-wise permutation (n data-dependent swaps based on the RC4 permutation).

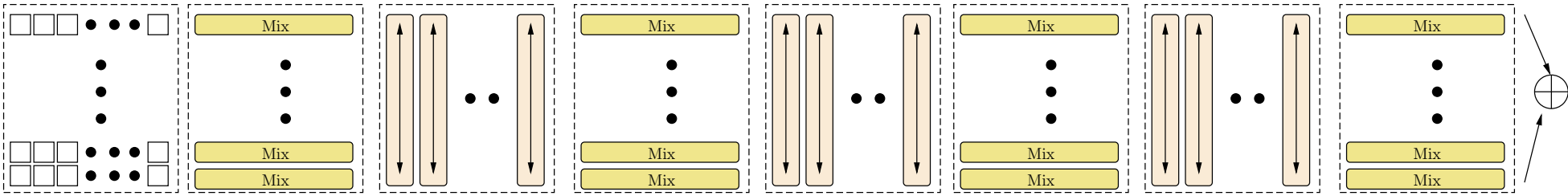
Tradeoff



When trying to attack apply the following strategy:

- Store permutations, not blocks (about $\frac{1}{2}$ of total memory);

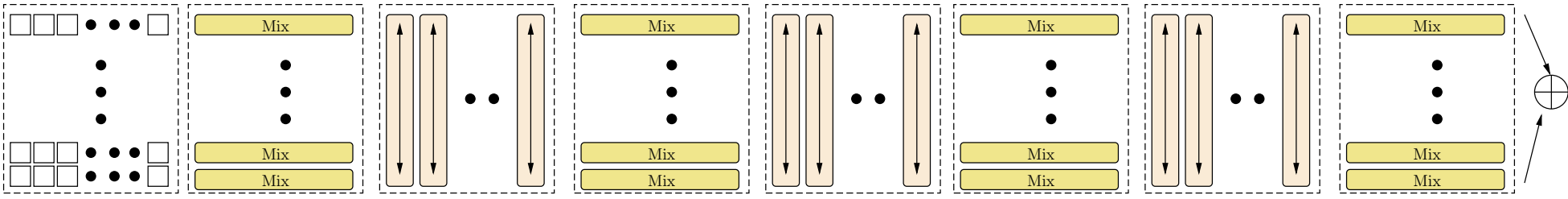
Tradeoff



When trying to attack apply the following strategy:

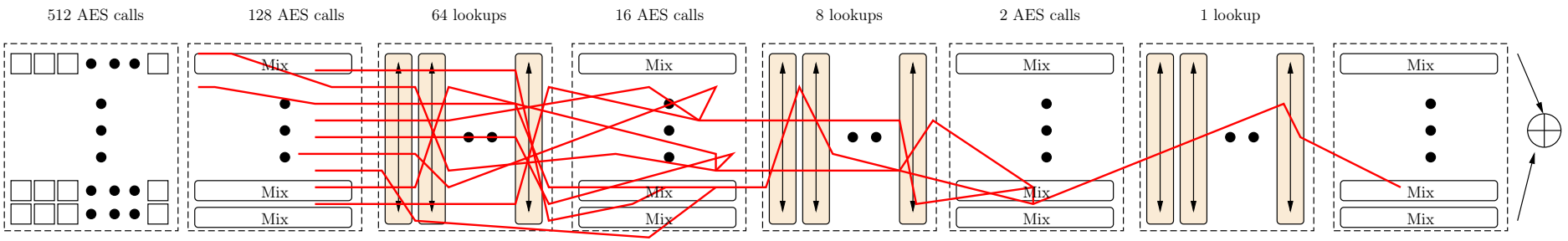
- Store permutations, not blocks (about $\frac{1}{2}$ of total memory);
- When an element is needed, recompute it;

Tradeoff



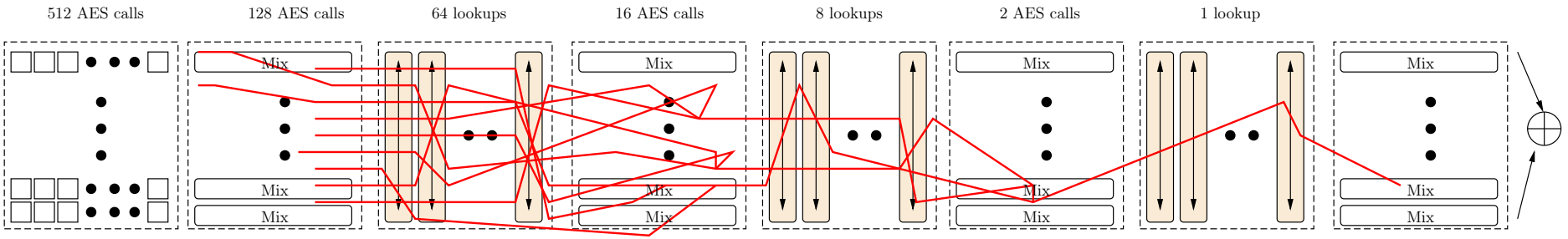
When trying to attack apply the following strategy:

- Store permutations, not blocks (about $\frac{1}{2}$ of total memory);
- When an element is needed, recompute it;
- Parallelize the RC4 permutation: ≈ 250 elements can be read in parallel without bank collisions.



Last level: one memory access is replaced with a tree of depth 7 of 5-round AES, which increases latency by a few times.

Further tradeoff



If the last permutation can not be saved, it has to be recomputed each time we need an element: 2^{18}-increase in latency.

Computational penalties

Penalties slightly depend on the memory size:

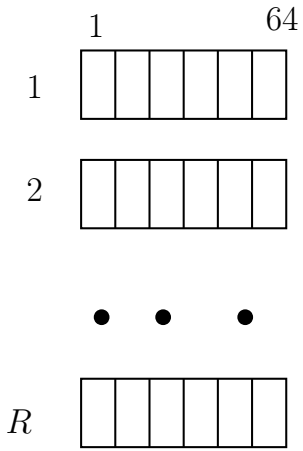
Fraction \ Memory	16 MB	128 MB	1 GB
$\frac{1}{2}$	139	160	180
$\frac{1}{4}$	2^{18}	2^{26}	2^{34}
$\frac{1}{8}$	2^{31}	2^{36}	2^{47}

Tradeoff:

$$T = \frac{N^4}{(cN)^{3N/M}}$$

Lyra2

Lyra2 [Simplicio-Almeida-Andrade-dos Santos-Barreto'13]:



$R \times 64$ matrix of 64-byte blocks. Two phases:

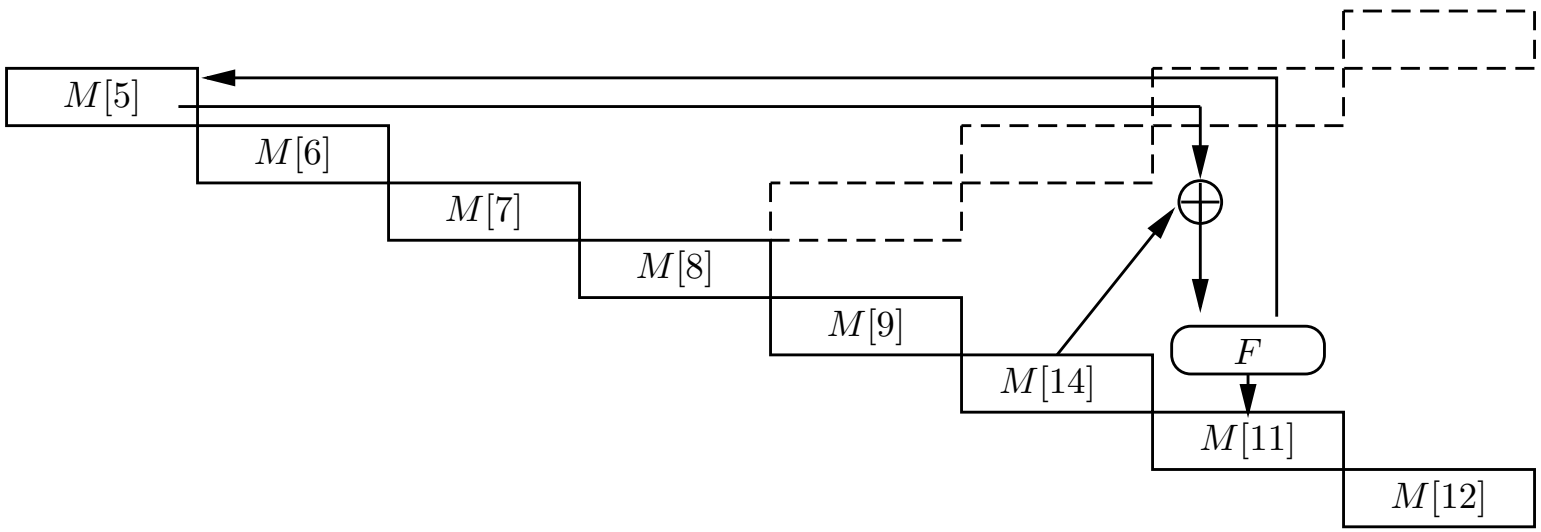
- Setup phase: deterministic generation and update of rows;
- Wandering phase ($T \geq 1$ iterations): sequential and pseudorandom update in parallel.

Claims high speed (1.2GB/sec for $T = 1$).

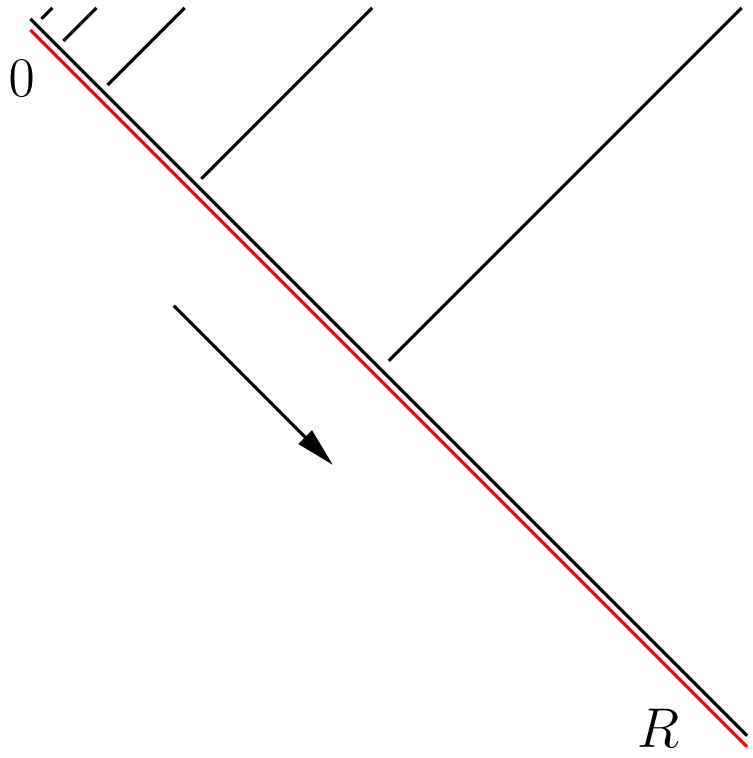
Setup phase

F — stateful function with 128-byte state (*sponge construction* based on hash function Blake2b).

$$M[i] \leftarrow F(M[i-1], M[2^k - i]),$$
$$M[2^k - i] \oplus = M[i];$$



Overall picture:

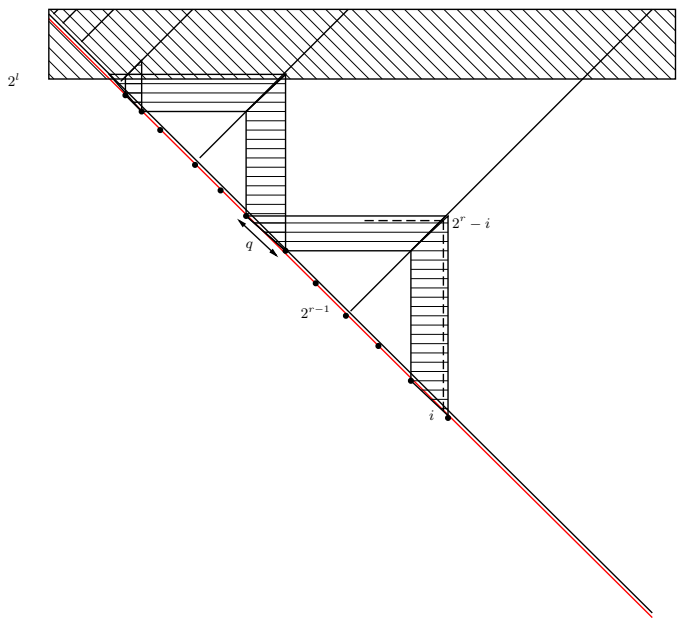


Tradeoff analysis: Setup phase

Strategy:

- Store first 2^l rows;
- Store every q-th row;

Then q consecutive rows are determined from $q(r-l)$ previous rows, which are precomputed.



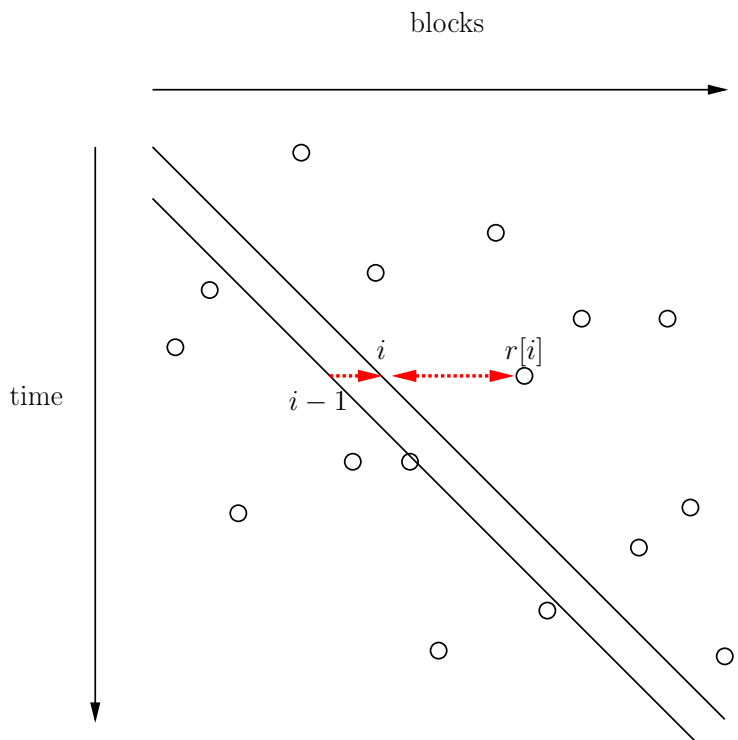
Setup phase can be computed with little penalty and memory.

Wandering phase

Wandering phase:

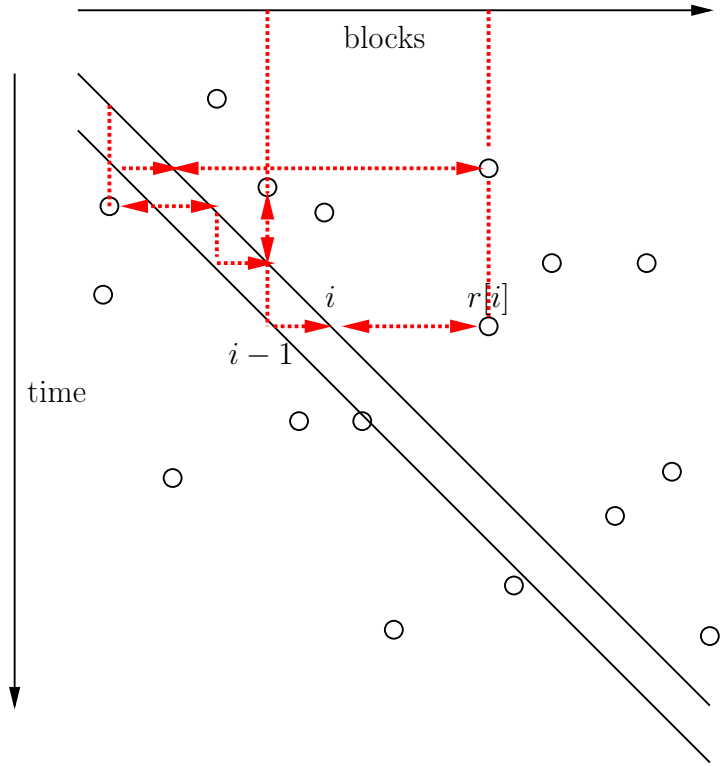
$$M[i] \leftarrow F(M[i-1], M[r_i],$$
$$M[r_i] \oplus = M[i];$$

Here r_i – pseudorandom function of $M[i-1]$ (i.e. determined at the time of computation).



Tradeoff analysis: Wandering phase

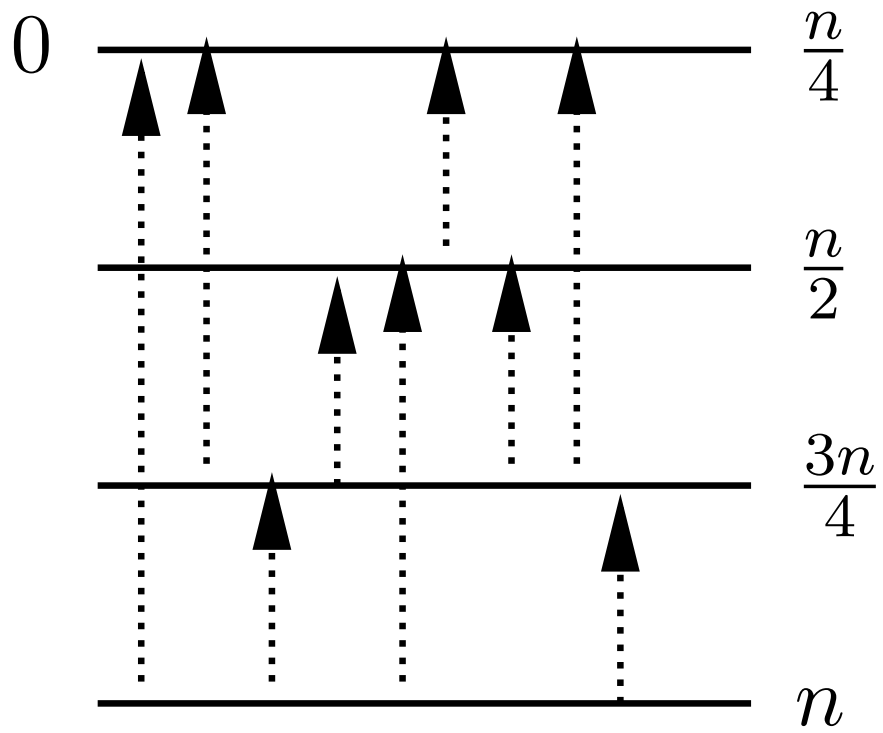
Pseudo-random dependency seem to impose prohibitive penalties:



Trees may cover the entire matrix.

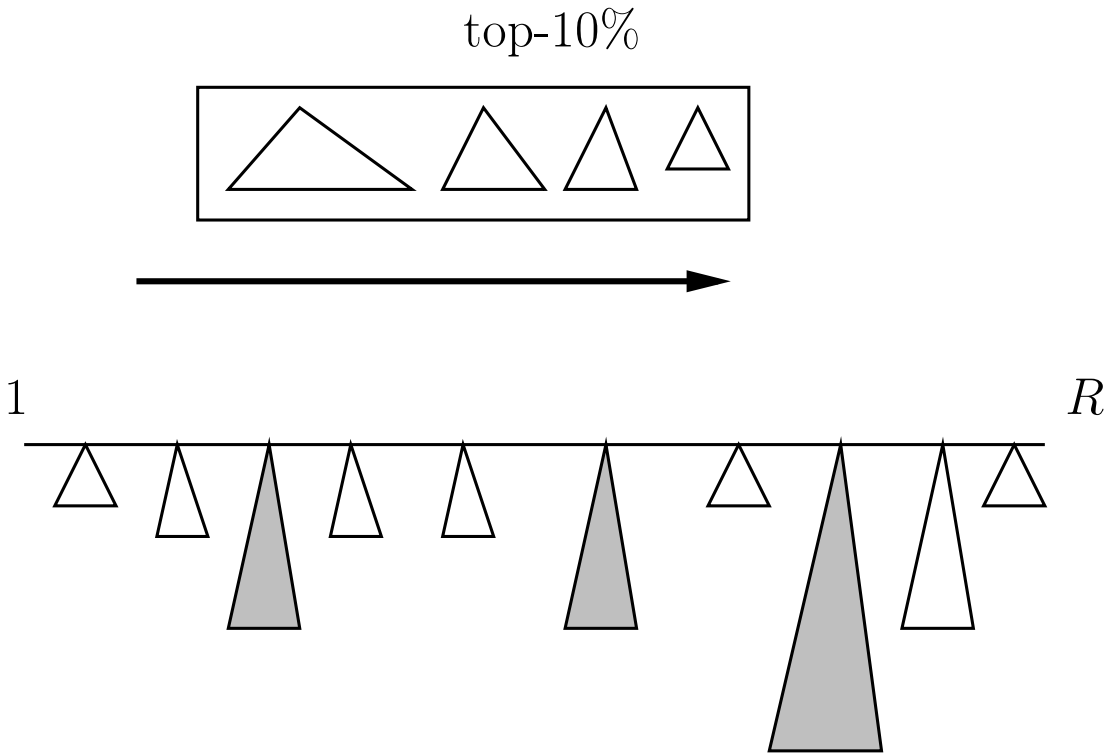
Tradeoff analysis: Wandering phase

First idea: split the computation into levels, store all links within the level.



Tradeoff analysis: Wandering phase

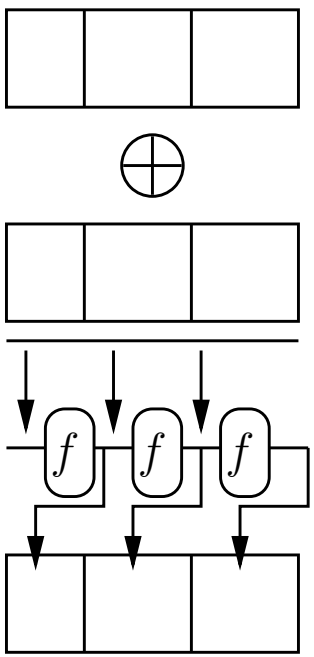
Second idea: store everything that refers to the most expensive rows (keep a list).

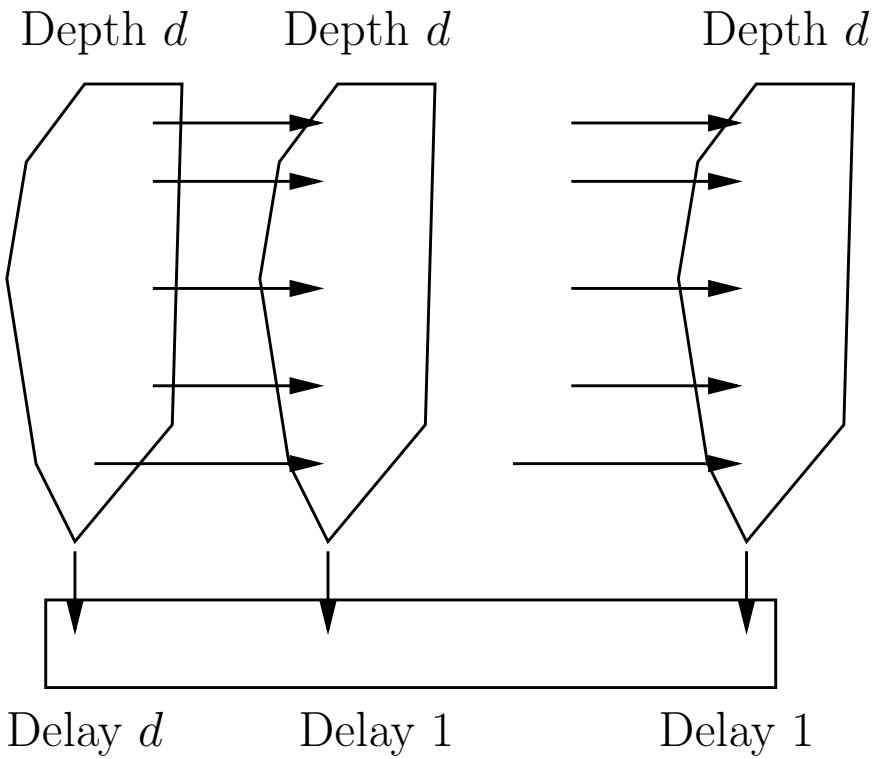


Tradeoff analysis: Wandering phase

Third idea: note that rows are updated column-wise. Good for CPU cache, but even better for ASIC-equipped adversaries.

- Store initial state of each row;
- Compute new row columnwise;
- So the extra latency is introduced before the first column only.





Penalties

Penalties:

Setup phase	
Memory fraction	Penalty
$\frac{1}{2}$	1.5
$\frac{1}{4}$	2
$\frac{1}{8}$	3
$\frac{1}{16}$	4

Wandering phase ($T = 1$)	
Memory fraction	Penalty
$\frac{1}{2}$	2
$\frac{1}{4}$	6.6
$\frac{1}{8}$	111.7
$\frac{1}{16}$	2^{16}

When we combine two phases, we count how many intervals of length q are accessed at the Wandering phase.

Total:

Memory fraction	Penalty
$\frac{1}{2}$	118
$\frac{1}{3}$	602
$\frac{1}{4}$	2241
$\frac{1}{6}$	14801

Catena, Argon, and Lyra2 tradeoffs for 1 GB:

Memory fraction	Penalty		
	Catena-3	Argon	Lyra2 ($T = 1$)
$\frac{1}{2}$	7.4	180	118
$\frac{1}{3}$	11.2	$2^{29.5}$	602
$\frac{1}{4}$	15.5	2^{34}	2241
$\frac{1}{8}$	30.1	2^{47}	2^{18}

Optimal ASIC implementations

Password crackers

History of password crackers:

- 70-90s: regular desktops;
- 00s: GPUs and FPGAs;
- 10s: dedicated hardware?

Password crackers

History of password crackers:

- 70-90s: regular desktops;
- 00s: GPUs and FPGAs;
- 10s: dedicated hardware?

Let us figure out how a rich adversary would build his password cracker.

ASIC

ASIC (application-specific integrated chip) — dedicated hardware.

- Large design costs (mln \$);
- Production costs high in small quantity;
- The most energy-efficient systems.

ASIC

ASIC (application-specific integrated chip) — dedicated hardware.

- Large design costs (mln \$);
- Production costs high in small quantity;
- The most energy-efficient systems.

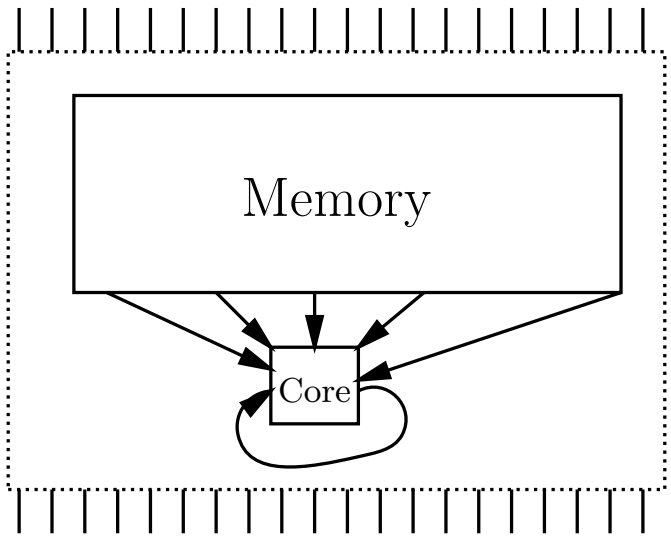
When passwords are of high value, an adversary may want to design a password-cracking scheme.

- Parallelism in computations;
- Parallelism in memory access (very difficult for all other architectures);
- In the long term electricity will dominate the costs.

So let us minimize the energy needed to compute a single password.

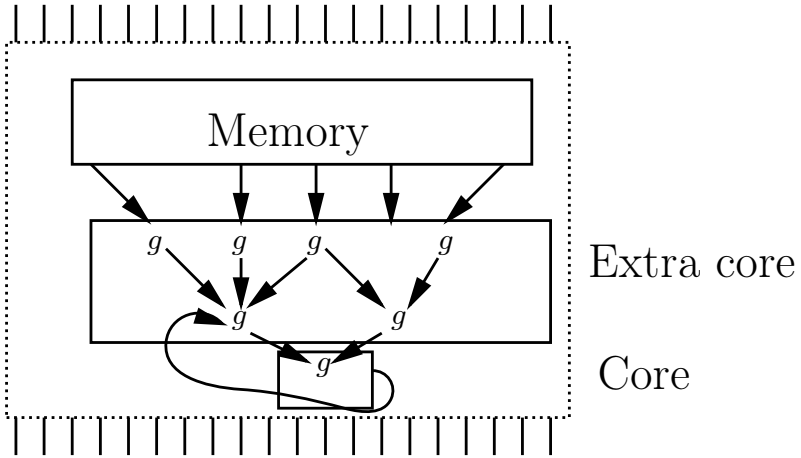
Straightforward implementation

A straightforward implementation of a password hashing scheme typically has a huge *memory block* and a small *computational core block*.



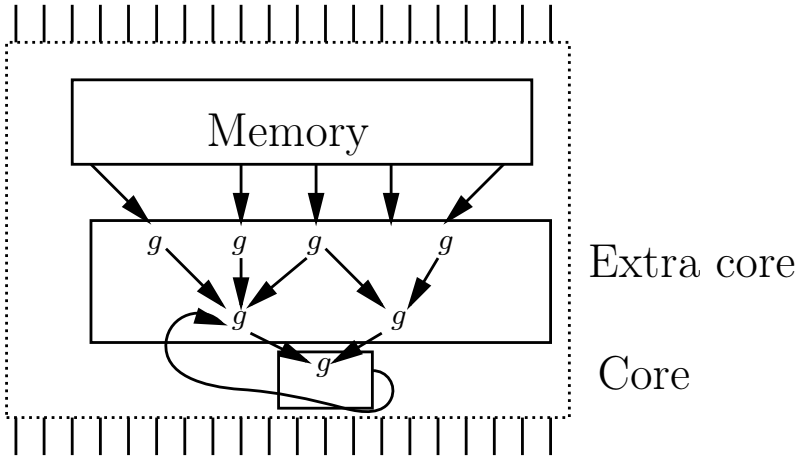
Tradeoff implementation

Less memory, more computations:



Tradeoff implementation

Less memory, more computations:



Time may not grow:

- If transformations are data-independent, they can be precomputed. Protection against cache-based timing attacks makes the scheme more vulnerable to tradeoff attacks.
- Data-dependent transformations introduce some latency.

However, at the other tree levels all data dependencies are known.

Tradeoff evaluation

What determines the cracking cost? The following metrics can be used:

- *Computational complexity* (total number of operations). Rather easy to compute, but inaccurate for memory-hard functions.
- *Time×area.* Good approximation to energy consumption if all the elements consume the same energy. Needs to know latencies and area requirements of all operations.
- *Energy cost*. More relevant when idle memory, active memory, and logic consume different power (actual for static RAM). Needs to know energy requirements of all elements.

Our assumptions

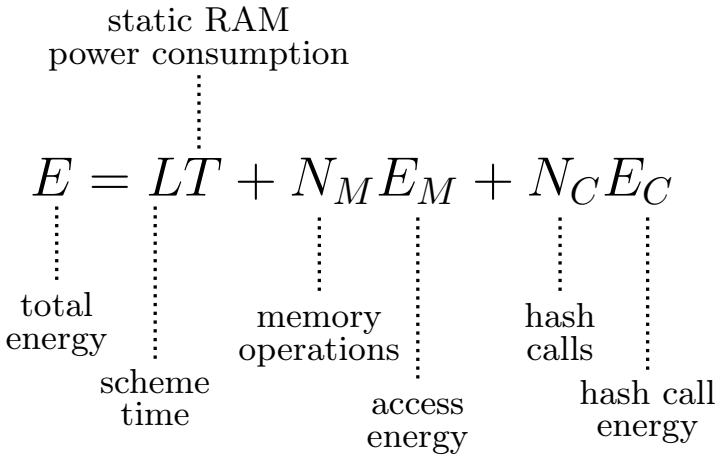
So far no one has placed that much memory on a single ASIC, so the exact behaviour of such chip is unknown. We make the following assumptions:

- Static RAM is more energy-efficient;
- The memory can be partitioned into 2^{16} banks (two levels of hierarchy);
- All banks can be read and written in parallel with average latency of 3 cycles;
- We ignore the area of communication wires between memory and computational cores. OK for our 2^{16} memory banks and not so many cores, but can be a problem for much more dense structure.

Energy model

Energy model

Energy model:



Three main contributors to the energy cost:

- Leakage power of static RAM;
- Memory access energy;
- Hash computation energy.

Reference platform

We take the best implementations scale them to the reference platform: 65nm CMOS technology, 1.2V supply voltage, 400 MHz frequency.

- AES: scaling down 22 nm, 1GHz implementation [Intel'2014], 1 cycle per round;
- Blake2b: scaling up and doubling 90nm, 286 MHz implementation of Blake-32 [Knezevic et al.'2011], 2 cycles per round;
- Static RAM: 65nm, 850 MHz implementation [Rooseleer et al.'2014];

Reference platform

Primitive	Power	Area	Latency
AES (full)	32 mW	17.5 kGE	10
Blake2b (full)	13.3 mW	19 kGE	20
16 KB – 32bit memory bank	12.6μW	192 kGE	3

Operation	Energy
1 Gcall (2^{30}) of AES	800 mJ
1 Gcall of Blake2b	867 mJ
1 GB memory reads/writes	1 mJ

Therefore, an AES core is equivalent to 700 bytes in area. One run of AES costs as much as reading 800 bytes.

ASIC implementations of tradeoffs

Catena

All Catena operations are independent of the data (to avoid cache-based side-channel attacks).

- This allows to precompute the hash tree by the time it is needed;
- If the memory is reduced by factor q, we add λq Blake2 cores on the chip.

1 GB Catena-3:

Total energy	Time	Memory			Blake	
		Fraction	Read	Energy	Gcalls	Energy
192 J	240 sec	1	6 GB	192 J	0.06	54 mJ
$\frac{192}{q} + 0.2q$ J	240 sec	$\frac{1}{q}$	12 GB	$\frac{192}{q}$ J	$0.25q$	$0.2q$ J

Optimal tradeoff point: $q = 32$, 12 J per password.

Argon

ASIC implementation

We use the following strategy:

- Always use 2^{10} AES cores for the Mix operation, this makes the latency of the AES part very low;
- When $\frac{1}{2}$ of memory is used, the latency grows by the factor of 6;
- When $\frac{1}{3}$ of memory is used, the latency further grows by the factor 2^{23}.

ASIC implementation

We use the following strategy:

- Always use 2^{10} AES cores for the Mix operation, this makes the latency of the AES part very low;
- When $\frac{1}{2}$ of memory is used, the latency grows by the factor of 6;
- When $\frac{1}{3}$ of memory is used, the latency further grows by the factor 2^{23}.

1 GB Argon:

Total energy	Time	Memory			AES (5 rounds)	
		Fraction	Read	Energy	Gcalls	Energy
209 mJ	0.02 sec	1	21 GB	34 mJ	0.43	175 mJ
33 J	0.1 sec	1/2	10.3 GB	52 mJ	83	33 J
139 MJ	8 hrs	1/3	2 PB	10 kJ	$2^{28.5}$	139 MJ

Efficiency drops very quickly.

Lyra2

ASIC implementation

Lyra:

- Store the initial values of the state (sponge);
- Compute the dependency tree columnwise;
- Quite large penalty for the setup phase, subject to improve.

Energy	Time	Memory			Blake (1 round)	
		Fraction	Read	Energy	Gcalls	Energy
71 mJ	0.08 sec	1	6 GB	68 mJ	0.03	3 mJ
318 mJ	0.10 sec	1/2	7.4 GB	50 mJ	3.7	269 mJ
1.4 J	0.15 sec	1/3	37.6 GB	77 mJ	18.8	1.4 J
5.2 J	0.17 sec	1/4	140 GB	173 mJ	70	5.1 J

Memory-full implementation is energy-efficient. Growth is not that high though.

Is Lyra2 secure?

Is Lyra2 secure?
Depends on the metric.

Time× area	Memory		Cores	
	Fraction	Size (MGE)	Number	Size (MGE)
120	1	1536	1	0.02
80	1/2	768	179	3.2
76	1/3	512	642	11.6
71	1/4	384	2079	37.6
91	1/5	307	2992	54.2

Optimal point at 1/4 of memory, as below that point the scheme execution time grows too fast.

Final remarks

- All numbers are preliminary and subject to improve;
- Much more time is needed to evaluate the security of most promising candidates (at least 2 months per submission);
- ASIC implementations and related tricks (data-independency, parallelization) are underestimated;
- Full report to be published soon (raw data can be given to anyone to verify);

Final remarks

- All numbers are preliminary and subject to improve;
- Much more time is needed to evaluate the security of most promising candidates (at least 2 months per submission);
- ASIC implementations and related tricks (data-independency, parallelization) are underestimated;
- Full report to be published soon (raw data can be given to anyone to verify);

Other candidates follow, stay tuned.

Final remarks

- All numbers are preliminary and subject to improve;
- Much more time is needed to evaluate the security of most promising candidates (at least 2 months per submission);
- ASIC implementations and related tricks (data-independency, parallelization) are underestimated;
- Full report to be published soon (raw data can be given to anyone to verify);

Other candidates follow, stay tuned.

UPDATE: RIG can be attacked with the same techniques.

Questions?