New algorithms for quantum (symmetric) cryptanalysis

María Naya-Plasencia, André Schrottenloher, André Chailloux, Lorenzo Grassi

New Algorithms for Quantum (Symmetric) Cryptanalysis

María Naya-Plasencia[2], André Schrottenloher[2]
Joint work with André Chailloux[2] and Lorenzo Grassi[1]

[1] IAIK, Graz University of Technology, Austria
[2] Inria, France

Outline

Quantum-safe (Symmetric) Cryptography

(Pre-quantum) cryptography

Enable secure communications even in the presence of malicious adversaries.

Asymmetric (e.g. RSA)

- No shared secret / computationally costly
- Security based on well-known hard mathematical problems (e.g. factorization)

Symmetric (e.g. AES)

- Shared secret / computationally efficient
- Ideal security defined by generic attacks (e.g. $2^{|K|}$)
- Need of continuous security evaluation (cryptanalysis)

A typical symmetric primitive

Ideal block cipher

E_K is a family of permutations of $\{0,1\}^n$ parameterized by K.

Real block cipher:

- Typically built by iterating a round function
- Select a key K
- Decompose the message into n-bit blocks and use E_K with a mode of operation

Generic attacks on ciphers

- The security provided by an ideal block cipher is defined by the best generic attack: exhaustive search for the key in $2^{|K|}$
- Recovering the key from a secure cipher must be infeasible.

Typical key sizes range from $|K| = 128$ to 256 bits.

Symmetric cryptanalysis

- The ideal security is defined by generic attacks ($2^{|K|}$)
- Does real security meet this ideal security?
- We won't know . . . without a continuous security evaluation.

Any attack better than the generic one is considered a "break".

Cryptanalysis is an empirical measure of security.

The security margin

The security of a cipher is not a 1-bit information:

- *e.g.* round-reduced attacks.

$\Rightarrow$ determine and adapt the security margin.

- The best attacks find the highest number of rounds reached (regardless of the complexity)
- Allows to compare primitives

Quantum-safe (Symmetric) Cryptography

Post-quantum cryptography

Asymmetric (e.g. RSA)

- Shor's algorithm factorizes in polynomial time: this is not secure anymore.
- Actively looking for replacements (NIST call)

Symmetric (e.g. AES)

Exhaustive search in $2^{|K|/2}$ with Grover's algorithm.

- Double the key length for equivalent ideal security.

In both cases, lots of work regarding quantum attacks.

Many new results

- Breaking some classically secure constructions in some quantum adversary models
- Extending cryptanalysis studies to quantum adversaries
- Solving recurrent generic problems

Quantum search

Find in S (of size 2^n) an element x (2^t solutions) such that x satisfies some condition.

$$\underbrace{2^{(n-t)/2}}_{\substack{2^t \text{ solutions} \\ \text{among } 2^n}} \left(\underbrace{\text{Sampling}}_{\substack{\text{Produce the} \\ \text{search space } S \text{ in} \\ \text{superposition}}} + \underbrace{\text{Checking}}_{\substack{\text{Test a} \\ \text{superposition of} \\ x \in S}} \right)$$

Two settings

"Low-qubits"

Only $\mathcal{O}(n)$ qubits, no qRAM access.

$\Rightarrow$ A quantum adversary from tomorrow.

Exponential qRAM

Read and write access in quantum superposition:

$$\sum_i |i\rangle |0\rangle \rightarrow \sum_i |i\rangle |a_i\rangle$$

Quantum Collision Search

with A. Chailloux, M. Naya-Plasencia

The birthday problem

Collision search

Let $H : \{0,1\}^n \to \{0,1\}^n$ be a random function, find a collision of H, *i.e.* a pair $x_1, x_2 \in \{0,1\}^n$ such that $H(x_1) = H(x_2)$.

Numerous applications, *e.g.* generic attacks on hash functions.

- Classical time and queries: $\Theta(2^{n/2})$
- With $2^{n/2}$ queries, we can form 2^n pairs, an n-bit collision occurs w.h.p.
- We can do this in $\mathcal{O}(n)$ memory (Pollard's rho)

Quantum algorithms for collisions

	Time	Queries	Qubits / qRAM	Classical memory
Pollard	$2^{n/2}$	$2^{n/2}$	0	$\mathcal{O}(n)$
Grover	$2^{n/2}$	$2^{n/2}$	$\mathcal{O}(n)$	0
Brassard, Høyer, Tapp	$2^{n/3}$	$2^{n/3}$	$2^{n/3}$	$2^{n/3}$
BHT (*)	$2^{2n/3}$	$2^{n/3}$	$\mathcal{O}(n)$	$2^{n/3}$

Collision search in a low-qubits setting

- Single-processor
- Only $\mathcal{O}(n)$ qubits
- No qRAM lookups

A naive collision algorithm

- Perform ℓ arbitrary classical queries to H: $H(x_1), \ldots, H(x_\ell)$.
- Search $x \in \{0,1\}^n$ such that:

$$H(x) \in \{H(x_1), \ldots, H(x_\ell)\}$$

Optimal $\ell = 2^{n/2}$:

$$2^{n/2} + \frac{2^n}{2^{n/2}}$$

A quantum collision algorithm

Naive classical:

- Perform ℓ arbitrary classical queries to H: $H(x_1), \ldots, H(x_\ell)$.
- Search $x \in \{0,1\}^n$ such that:

$$H(x) \in \{H(x_1), \ldots, H(x_\ell)\}$$

Optimal $\ell = 2^{n/2}$**:**

$$2^{n/2} + \frac{2^n}{2^{n/2}}$$

Quantum (BHT):

- Perform ℓ arbitrary classical queries to H : $H(x_1), \ldots, H(x_\ell)$.
- **With Grover**, search $x \in \{0,1\}^n$ such that $H(x) \in \{H(x_1), \ldots, H(x_\ell)\}$.

Optimal $\ell = 2^{n/3}$**:**

$$\underbrace{2^{\frac{n}{3}}}_{\text{List}} + \underbrace{\sqrt{\frac{2^n}{2^{n/3}}}}_{\text{Iterations}} \left(1 + \underbrace{1}_{\substack{\text{qRAM} \\ \text{lookup}}}\right)$$

Removing qRAM

We have a list $L = \{H(x_1), \ldots, H(x_\ell)\}$, known classically, and want to compute:

$$|y\rangle |0\rangle \mapsto |y\rangle |y \in L\rangle \quad .$$

- With qRAM: build a data structure for L, compute membership in $\mathcal{O}(\log \ell)$ qRAM gates;
- Without qRAM: compare sequentially against elements of L.

We compute:

$$|y\rangle |0\rangle \mapsto |y\rangle |(y = H(x_1)) \vee (y = H(x_2)) \ldots \vee (y = H(x_\ell))\rangle$$

in time $\widetilde{\mathcal{O}}(\ell)$.

BHT without quantum memory

Queries:

$$2^{n/3} + \sqrt{2^n/2^{n/3}}\,(1+0)$$

Time:

$$2^{n/3} + 2^{n/3}\left(1 + 2^{n/3}\right)$$

Can we improve this?

Let's build a list of distinguished points, *e.g.* $H(x_i) = 0^u||z$ for $z \in \{0,1\}^{n-u}$.

- Building the list costs more: $2^{n/3+u/2}$
- We have a setup cost (for searching among DPs): $2^{u/2}$ per iteration
- The test still requires $2^{n/3}$ time
- BUT less iterations: $2^{n/3-u/2}$

$$\underbrace{\underbrace{2^{\frac{n}{3}}}_{\text{List size}} \times \underbrace{2^{\frac{u}{2}}}_{\substack{\text{Grover search} \\ \text{of a DP}}}}_{\text{First step: constructing the list}} + \underbrace{\underbrace{2^{\frac{n}{3}-\frac{u}{2}}}_{\text{Less iterations}} \left(\underbrace{2^{\frac{u}{2}}}_{\substack{\text{Building} \\ \text{all the DPs}}} + \underbrace{2^{\frac{n}{3}}}_{\text{Lookup}} \right)}_{\text{Second step: searching a collision}}$$

With optimal parameters

The cost becomes optimal for an intermediate list of size $2^v \neq 2^{n/3}$.

$$\underbrace{\underbrace{2^v}_{\text{List size}} \times \underbrace{2^{\frac{u}{2}}}_{\text{Grover search of a DP}}}_{\text{First step: constructing the list}} + \underbrace{\underbrace{2^{\frac{n-v-u}{2}}}_{\text{Less iterations}} \left(\underbrace{2^{\frac{u}{2}}}_{\text{Building all the DPs}} + \underbrace{2^v}_{\text{Lookup}} \right)}_{\text{Second step: searching a collision}}$$

With $v = \frac{n}{5}$, $u = \frac{2n}{5}$, time: $\widetilde{\mathcal{O}}\left(2^{2n/5}\right)$.
We also need $2^{n/5}$ *classical memory*.

Conclusion

- An asymptotic difference for collisions: time reduced from $2^{n/2}$ to $2^{2n/5}$
- Smallest number of computations when qRAM is not used
- More applications: multi-user settings, operation modes...

Example: $n = 128$, 2^{51} hash function queries instead of 2^{64}, with less than $1GB$ classical data.

State of the problem

	Time	Queries	Qubits	Classical memory
Pollard	$2^{n/2}$	$2^{n/2}$	0	$\mathcal{O}(n)$
Grover	$2^{n/2}$	$2^{n/2}$	$\mathcal{O}(n)$	0
BHT	$2^{n/3}$	$2^{n/3}$	$2^{n/3}$	$2^{n/3}$
New	$2^{2n/5}$	$2^{2n/5}$	$\mathcal{O}(n)$	$2^{n/5}$

Can we meet the lower bound $2^{n/3}$ with $\mathcal{O}(n)$ qubits?

Quantum k-xor Algorithms

with L. Grassi, M. Naya-Plasencia (AC' 18)

Generalized Birthday Problem(s)

Problem 1: The "original"

Given $L_1, \ldots L_{\mathbf{k}}$ classical lists of random n-bit strings, find $x_1, \ldots x_{\mathbf{k}} \in L_1 \times \ldots L_{\mathbf{k}}$ such that $x_1 \oplus \ldots \oplus x_{\mathbf{k}} = 0$.

Problem 2: The "oracle"

Given oracle access to a random n-bit to n-bit function H, find $x_1, \ldots x_{\mathbf{k}}$ such that $H(x_1) \oplus \ldots \oplus H(x_{\mathbf{k}}) = 0$.

Problem 3: The "unique solution"

Given oracle access to a random $n/\mathbf{k}$-bit to n-bit function H, find the single $\mathbf{k}$-tuple $x_1, \ldots x_{\mathbf{k}}$ such that $H(x_1) \oplus H(x_2) \oplus \ldots H(x_{\mathbf{k}}) = 0$.

Focus on Problem 2 (with oracle)

Problem 2: The "oracle" k-xor

Let $H : \{0,1\}^n \to \{0,1\}^n$ be a random function, find $x_1, \ldots, x_{\mathbf{k}}$ such that $H(x_1) \oplus \ldots \oplus H(x_{\mathbf{k}}) = 0$.

- Cryptanalysis: (R)FSB, SWIFFT...
- Applications for $\oplus$ (bitwise XOR) and modular $+$
- Related: approximate variants, subset-sums, decoding random linear codes, lattice problems...

Examples

We note $\widetilde{\mathcal{O}}\left(2^{\alpha_{\mathbf{k}} n}\right)$ the best time complexity of **k**-xor.

The 1-xor Problem: exhaustive search

Searching x such that $H(x) = 0$: a preimage of 0. Simply use Grover's algorithm: $\alpha_{\mathbf{1}} = 1/2$.

The 2-xor Problem: collision search

Previously: $\alpha_{\mathbf{2}} = 1/3$ with qRAM and $2/5$ without.

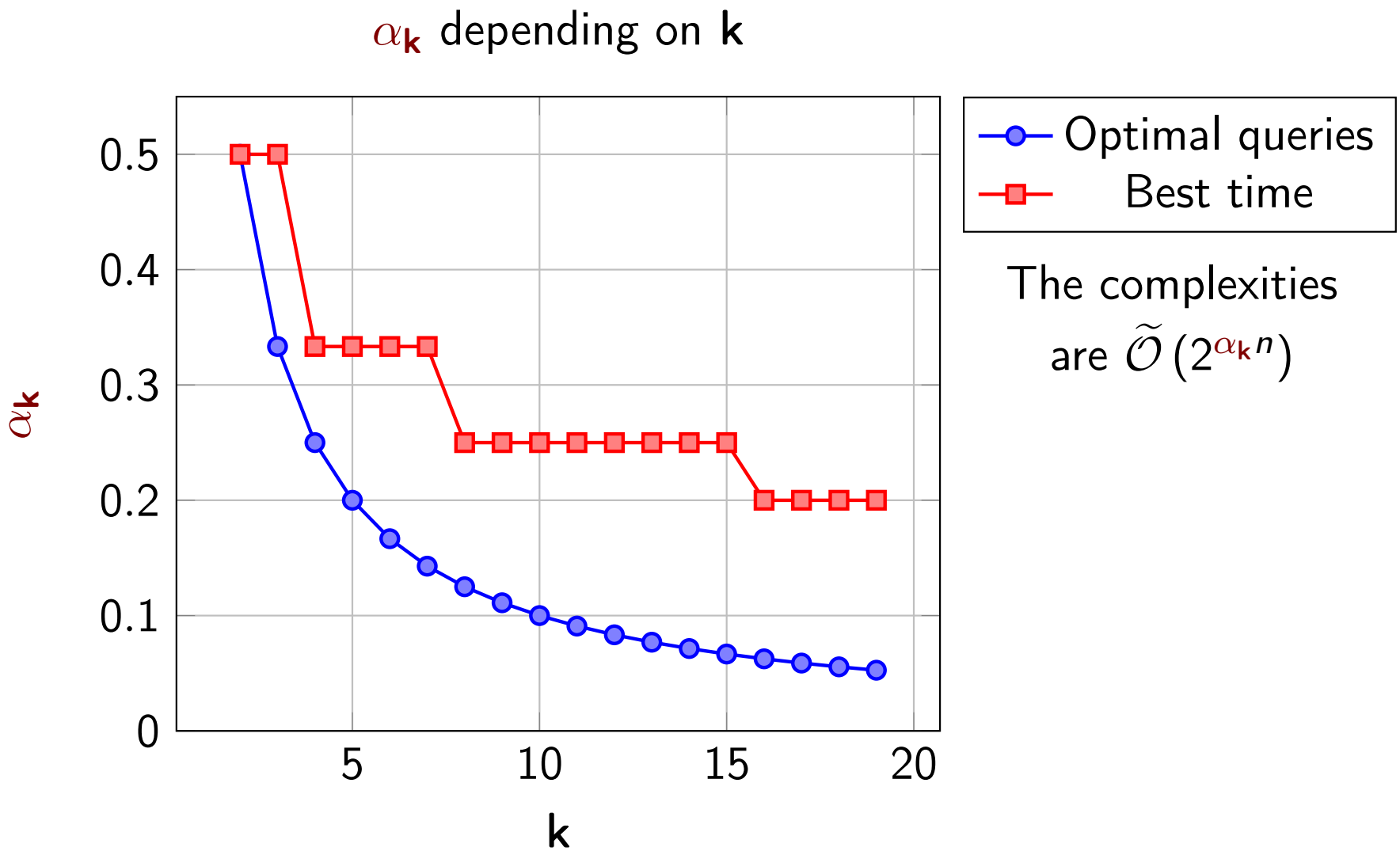
The problem becomes easier when **k** increases: $\alpha_{\mathbf{k}}$ is a decreasing function of **k**.

Classical results for general k

To get a **k**-xor on n bits:

- The optimal query complexity is $\Theta(2^{n/\mathbf{k}})$
- The time complexity is $\mathcal{O}\left(2^{n/(1+\lfloor \log_2(\mathbf{k}) \rfloor)}\right)$ (Wagner, 2002): $\alpha_{\mathbf{k}} = \frac{1}{1+\lfloor \log_2(\mathbf{k}) \rfloor}$
- Logarithmic improvements in time
- We focus on **exponents**

Classical results



Wagner's algorithm in a single slide

Let L_1 and L_2 be lists of 2^u random values of H. Build L: among all pairs $x_1, x_2 \in L_1 \times L_2$, we take the partial collisions on the first u bits.
Then:

- L contains 2^u elements (there are 2^{2u} pairs and a u-bit condition)
- L can be built in time 2^u if L_1 and L_2 are sorted

This works recursively: from two lists L_1, L_2 of partial **k**-xors, we can obtain a list of 2**k**-xors on more bits in time:

$$\mathrm{MAX}\left(\text{size of the output list}, \mathrm{MIN}\left(\text{size of } L_1, \text{size of } L_2\right)\right)$$

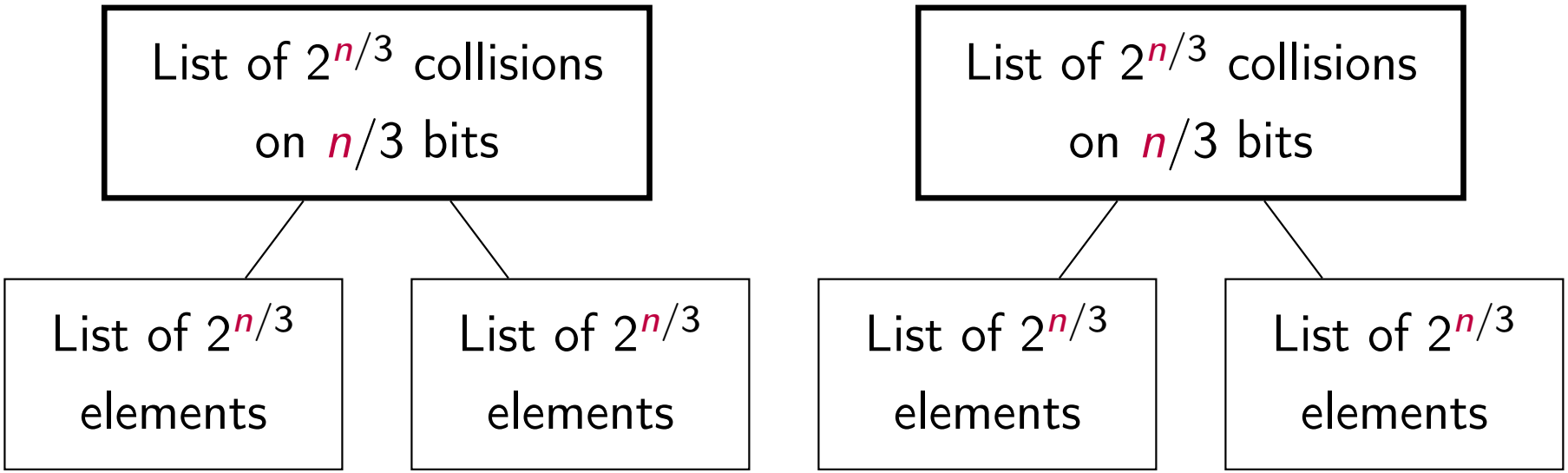
An example with $k = 4$

1. Query 4 lists of $2^{n/3}$ single elements (values of H): time $2^{n/3}$

List of $2^{n/3}$ elements	List of $2^{n/3}$ elements	List of $2^{n/3}$ elements	List of $2^{n/3}$ elements

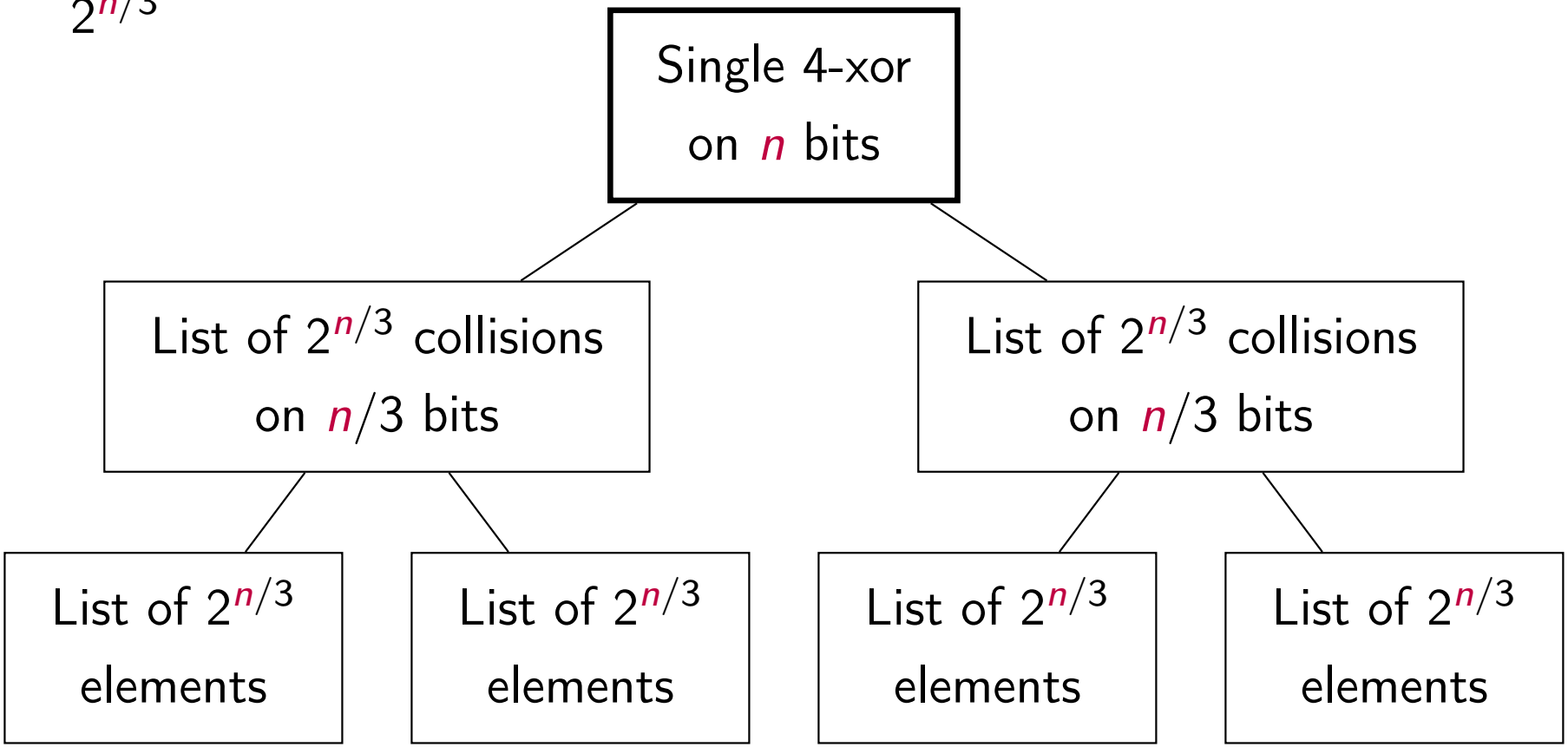
An example with $k = 4$

1. Query 4 lists of $2^{n/3}$ single elements (values of H): time $2^{n/3}$
2. Merge into two lists of $2^{n/3}$ collisions on $n/3$ bits: time $2^{n/3}$



An example with k = 4

1. Query 4 lists of $2^{n/3}$ single elements (values of H): time $2^{n/3}$
2. Merge into two lists of $2^{n/3}$ collisions on $n/3$ bits: time $2^{n/3}$
3. Find a collision between these lists: a single 4-xor of H: time $2^{n/3}$

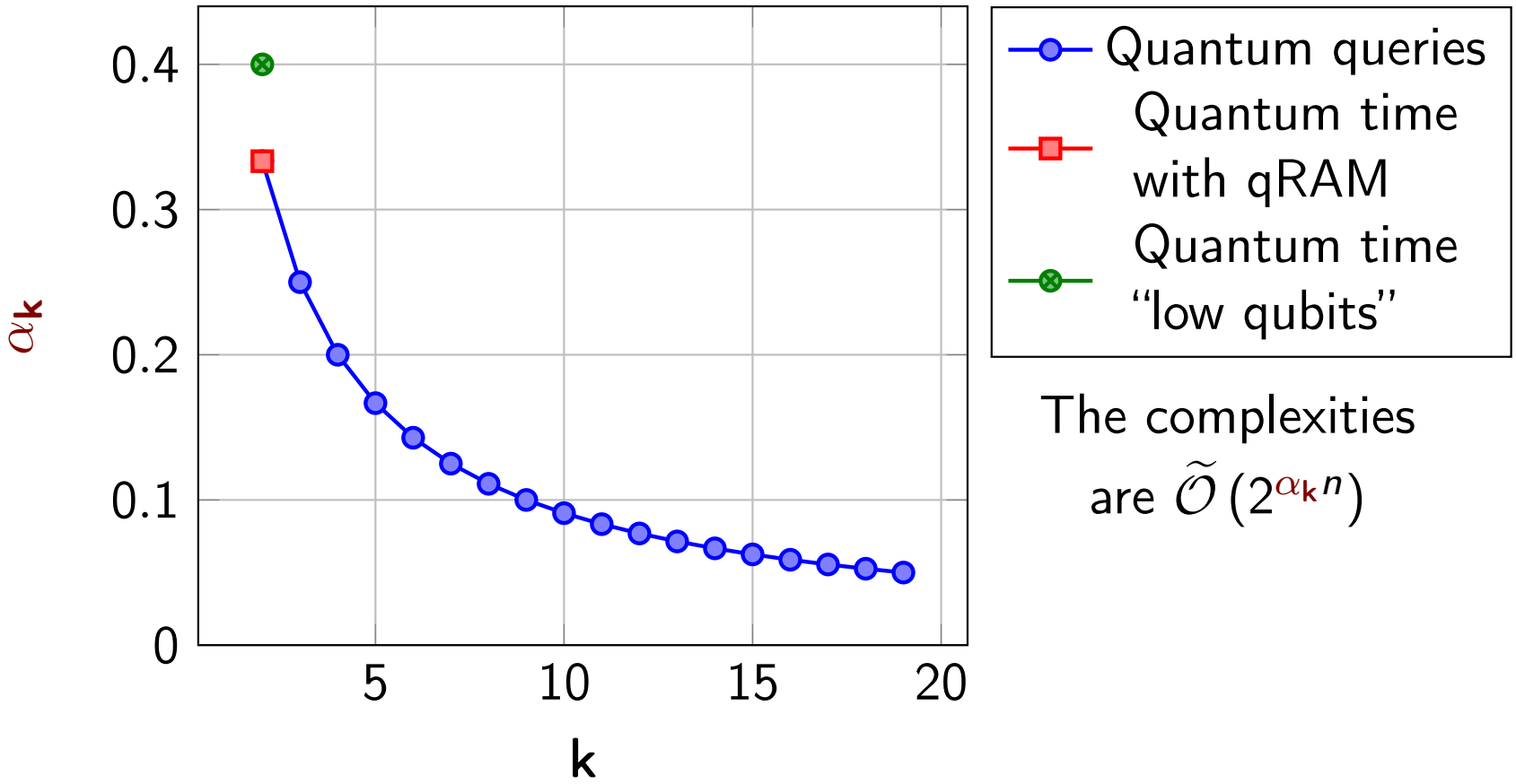


Previous quantum results on k-xor

To get a $\mathbf{k}$-xor on n bits:

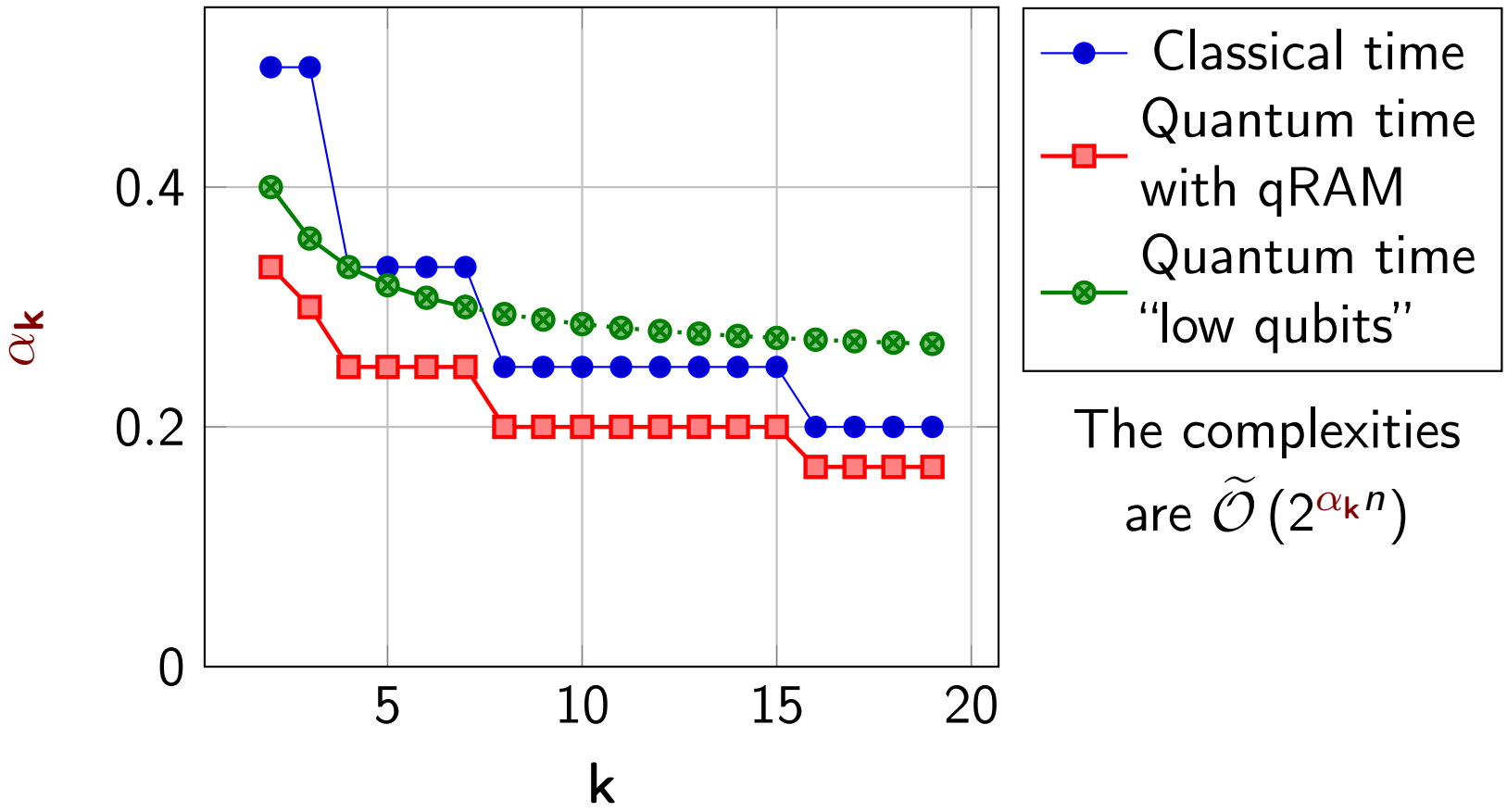
- The optimal query complexity is $\Theta\left(2^{n/(\mathbf{k}+1)}\right)$ (Belovs and Spalek)
- We know what happens for $\mathbf{k} = 2$.
- For $\mathbf{k} > 2$?

Previous quantum results



The complexities are $\widetilde{\mathcal{O}}\left(2^{\alpha_k n}\right)$

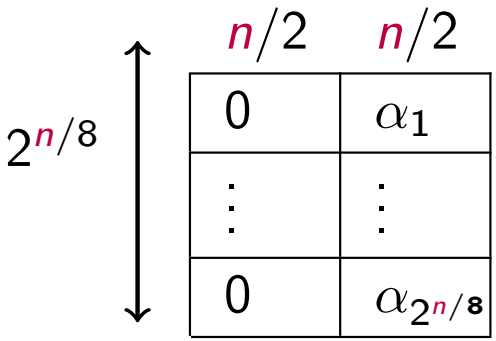
Results of AC' 18

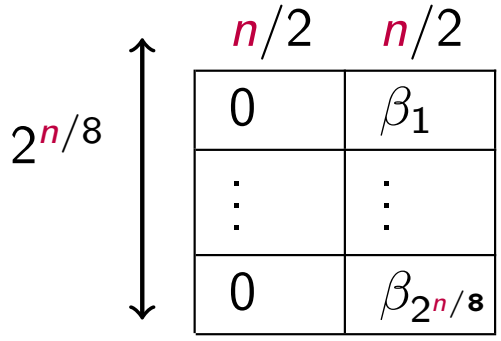


The complexities are $\widetilde{\mathcal{O}}\left(2^{\alpha_k n}\right)$

Low-qubits merging strategy for k = 3

We don't have a single intermediate list, but two of them ⇒ they can be smaller.





- Searching for a "distinguished solution": we compare against all $y, z \in L_1 \times L_2$
- Producing the lists costs $2^{n/4} \times 2^{n/8} = 2^{3n/8}$ time and as much for searching x.

Low-qubits merging strategy for k = 3 (ctd.)

$\ell = 2^{n/7}$

$2n/7$	$n/7$	$n/7$	$3n/7$
0	0	y_1	α_1
⋮	⋮	⋮	⋮
0	0	$y_{2^{n/7}}$	$\alpha_{2^{n/7}}$

$2^{n/7}$

$2n/7$	$n/7$	$n/7$	$3n/7$
0	z_1	0	β_1
⋮	⋮	⋮	⋮
0	$z_{2^{n/7}}$	0	$\beta_{2^{n/7}}$

We take more specific L_1 and L_2. Checking a distinguished point x:

- Match L_1 (find a partially colliding element); then match L_2;
- Compute the xor of the three values.

$$2^{n/7+3n/14} + \underbrace{2^{3n/14}}_{\substack{3n/7 \text{ remaining} \\ \text{bits}}} \left(\underbrace{2^{n/7}}_{\substack{\text{Setup} \\ \text{search} \\ \text{space}}} + \underbrace{\left(\underbrace{2^{n/7}}_{\text{Match } L_1} + \underbrace{2^{n/7}}_{\text{Match } L_2} \right)}_{\text{Instead of } 2^{n/7} \times 2^{n/7}} \right) = 2^{5n/14}$$

qRAM merging strategy for k = 3

$\ell = 2^{n/5}$

$n/5$	$n/5$	$3n/5$
0	y_1	α_1
$\vdots$	$\vdots$	$\vdots$
0	$y_{2^{n/5}}$	$\alpha_{2^{n/5}}$

$2^{n/5}$

$n/5$	$n/5$	$3n/5$
z_1	0	β_1
$\vdots$	$\vdots$	$\vdots$
$z_{2^{n/5}}$	0	$\beta_{2^{n/5}}$

$$2^{n/5+n/10} + \underbrace{2^{3n/10}}_{\substack{3n/5 \text{ bits} \\ \text{remaining}}} \left(\underbrace{1}_{\substack{\text{Matching} \\ L_1}} + \underbrace{1}_{\substack{\text{Matching} \\ L_2}} \right) = 2^{3n/10} < 2^{n/3}$$

$\Rightarrow$ quantum 3-xor is exponentially faster than quantum collision search.

Conclusion of AC' 18

Quantum 3-xor is exponentially faster than quantum collision search.

Low-qubits $\mathbf{k}$-xor improves over classical for $\mathbf{k} \leq 7$.

$\mathbf{k}$-xor with qRAM in time $\widetilde{\mathcal{O}}\left(2^{n/(\mathbf{2}+\lfloor \log_2(\mathbf{k}) \rfloor)}\right)$ (instead of $\mathcal{O}\left(2^{n/(\mathbf{1}+\lfloor \log_2(\mathbf{k}) \rfloor)}\right)$).

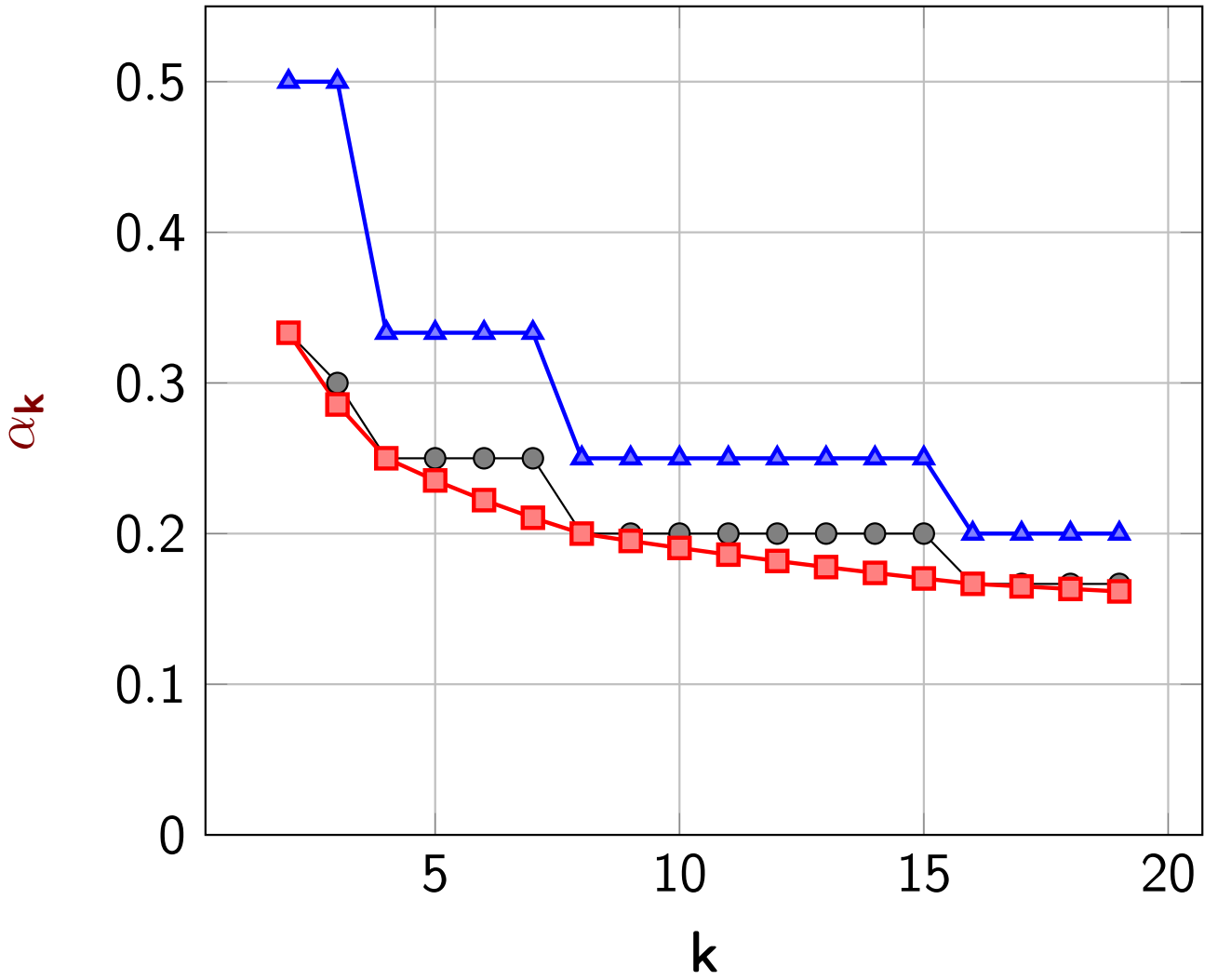
Open questions

- A low-qubits speedup for all $\mathbf{k}$?
- With qRAM, other improvements than $\mathbf{k} = 3$?

(Very) Recent Quantum Algorithms for k-xor

with María Naya-Plasencia

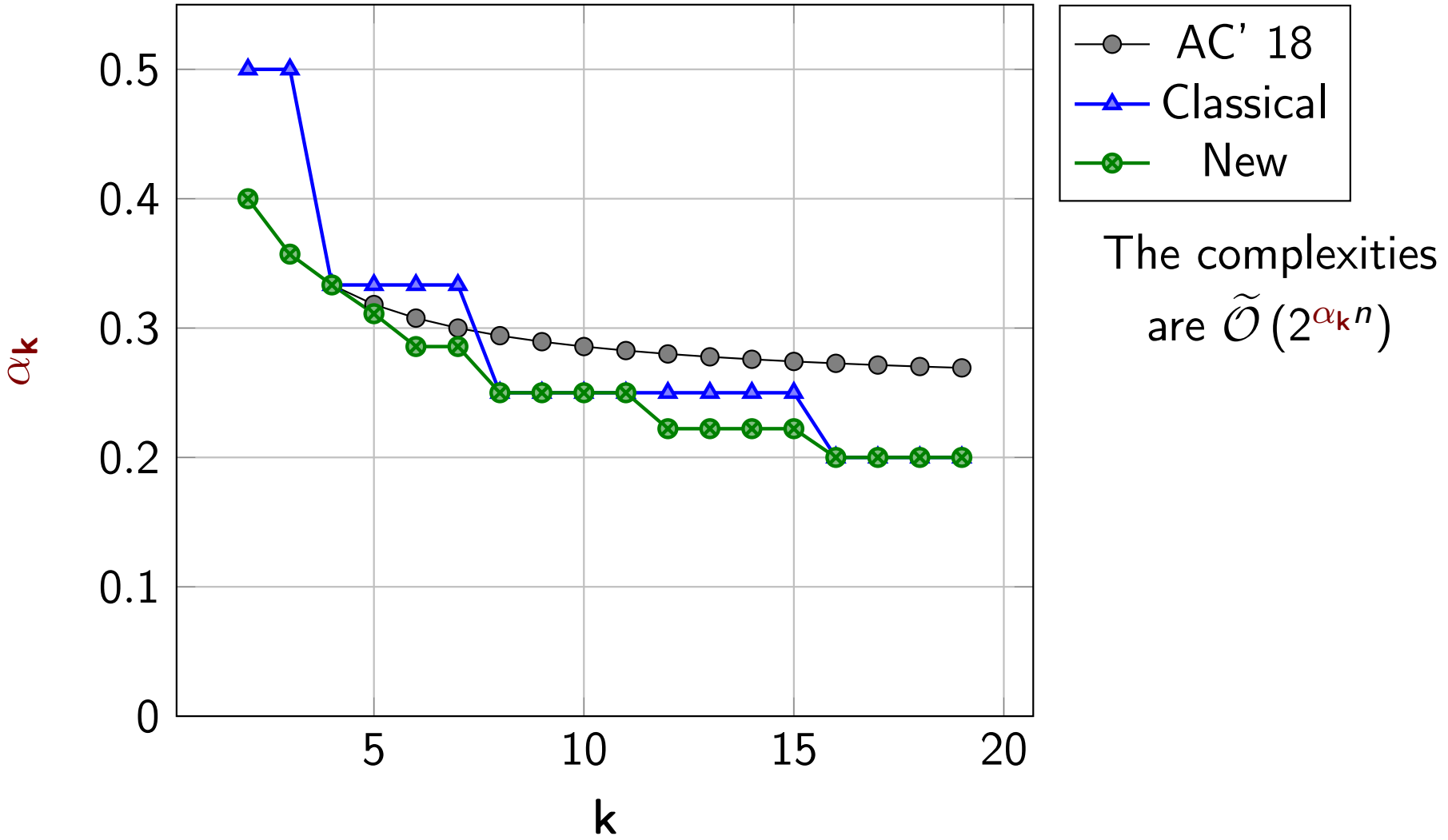
Recent results (with qRAM)



AC' 18
Classical
New

The complexities are $\widetilde{\mathcal{O}}\left(2^{\alpha_k n}\right)$

Recent results (low-qubits)



The complexities are $\widetilde{\mathcal{O}}\left(2^{\alpha_k n}\right)$

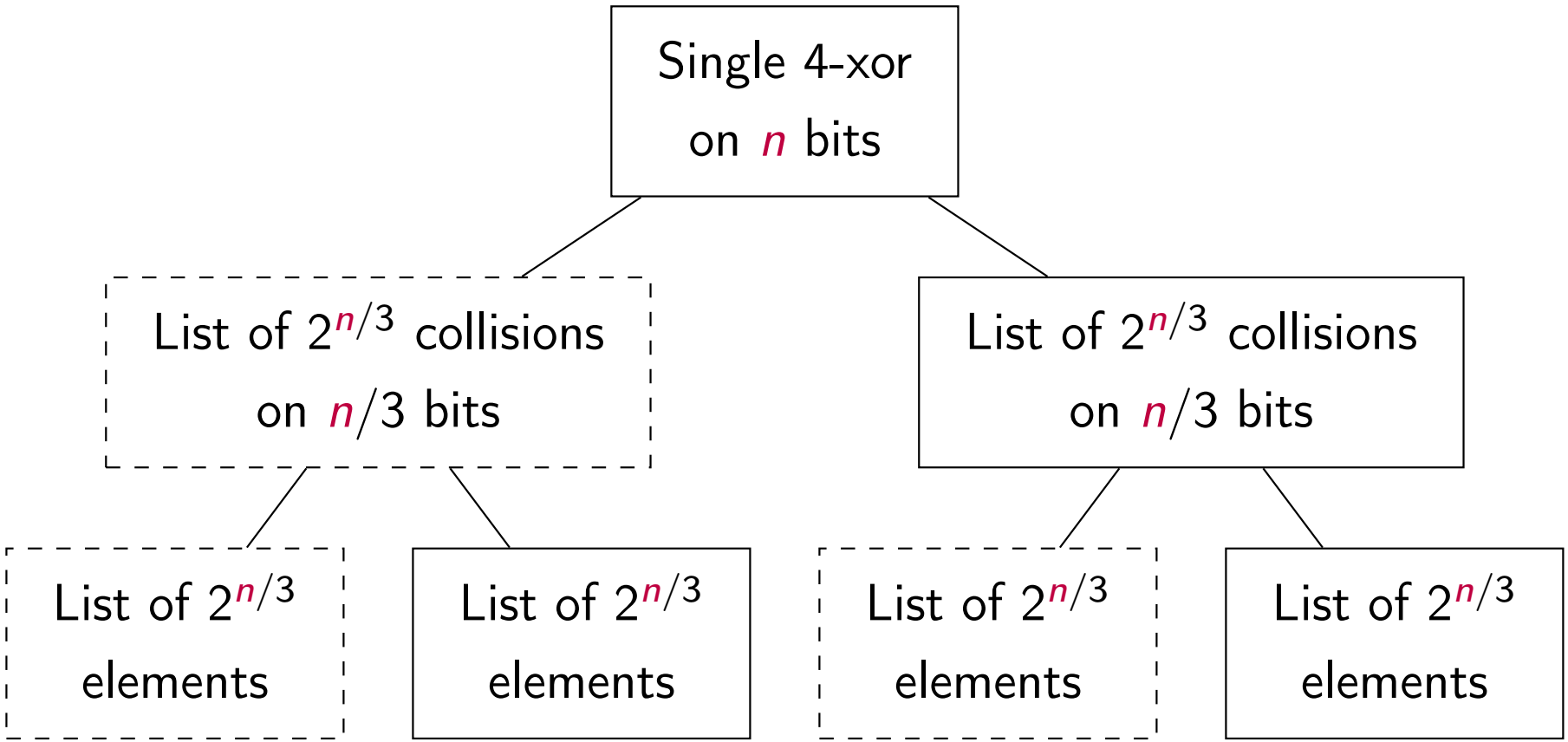
History

- We found some punctual improvements, for some values of **k**;
- We realized that all the possibilities could be included in a single framework: merging in a quantum-compliant way;
- We implemented an automatic search for the best merging strategies.

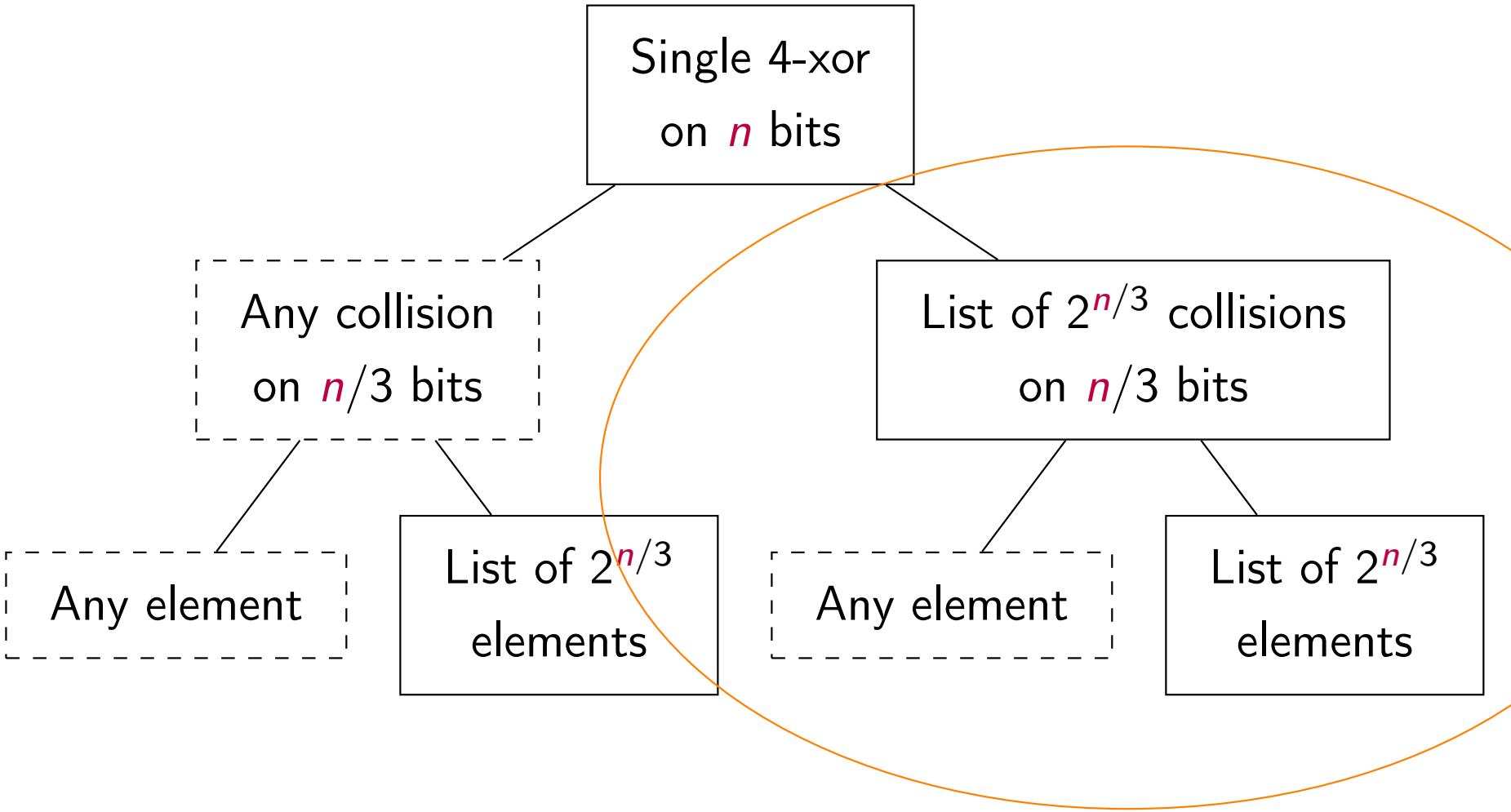
Merging strategies: build successive lists of partial ℓ-xor for increasing ℓ.

Back to classical merging

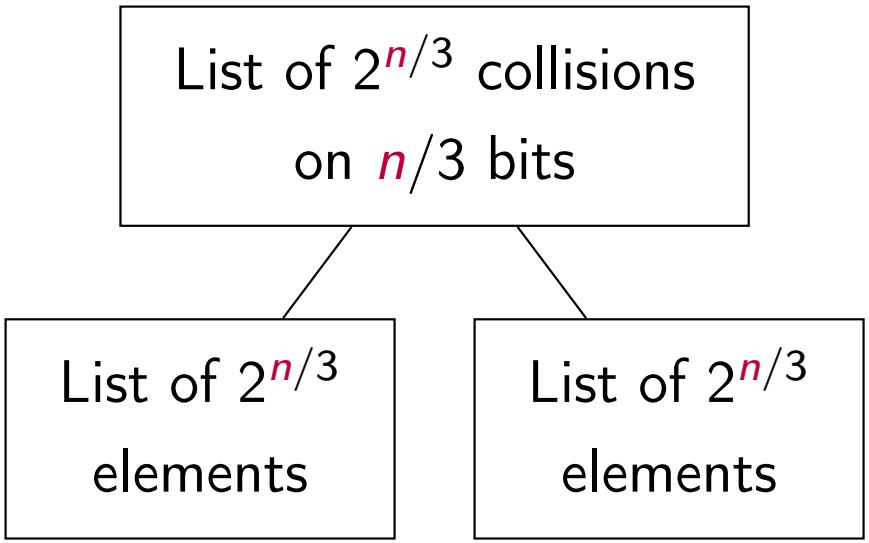
Traverse the tree of merges in a depth-first manner (Wagner, 2002): store $\lceil \log_2 \mathbf{k} \rceil$ lists instead of $\mathbf{k}$.



Rephrasing the classical 4-xor algorithm



From merging to matching



Before:

Two lists of $2^{n/3}$ elements

(random queries to H)

$\Downarrow$

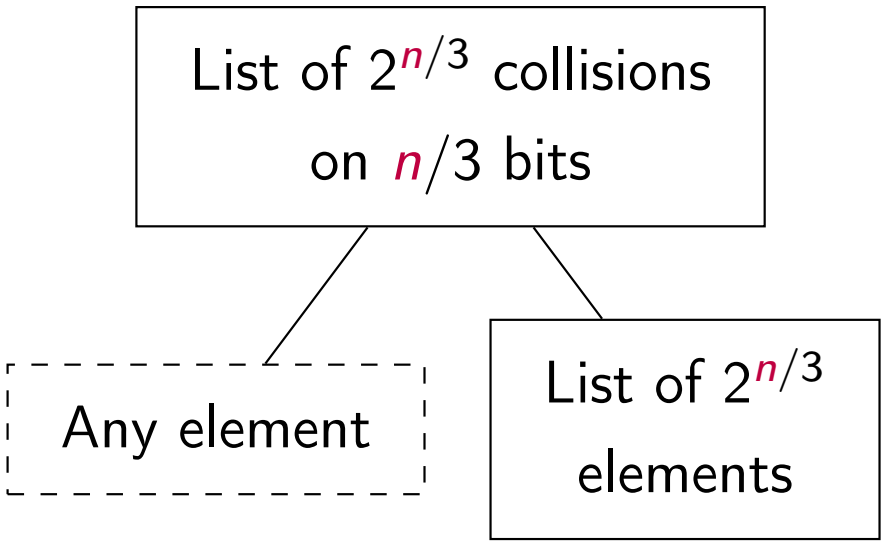
$2^{2n/3}$ pairs

$\Downarrow$

$2^{n/3}$ pairs with $n/3$-bit collision

In time $2^{n/3}$ (sorted lists).

From merging to matching



After:

A single list of $2^{n/3}$ elements

$\Downarrow$

Query H on the fly

$\Downarrow$

Each query yields $2^{n/3}$ pairs

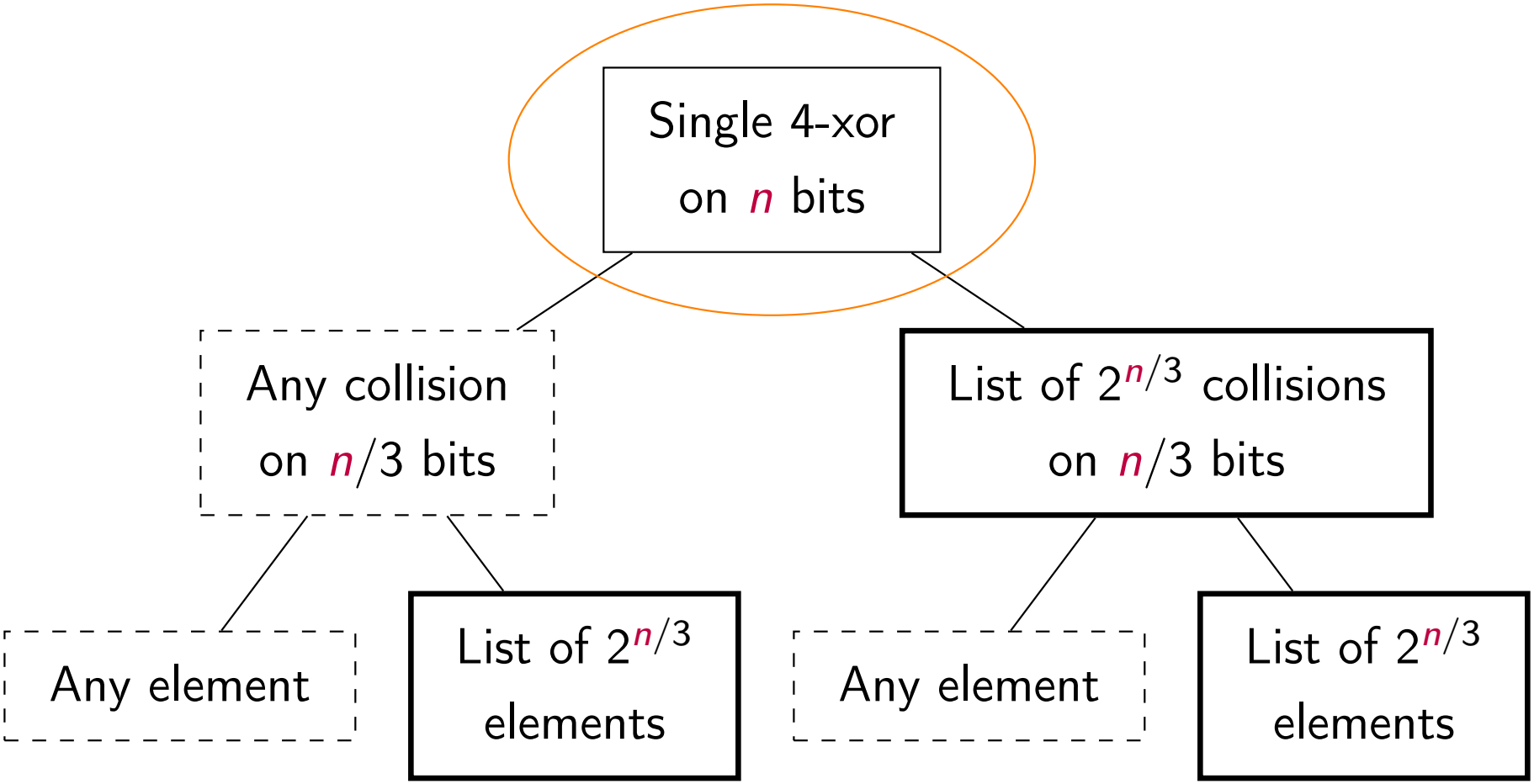
$\Downarrow$

An $n/3$-bit collision

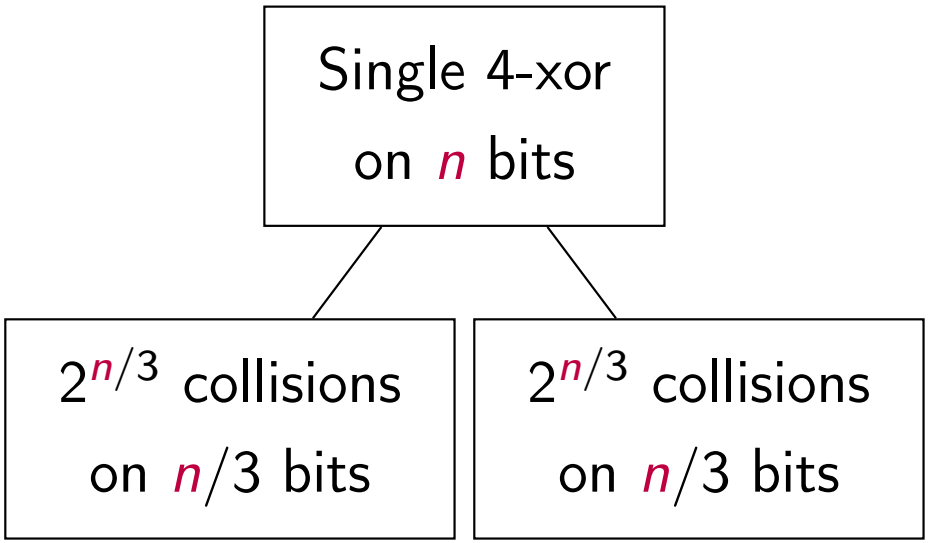
In time $2^{n/3}$ (sorted list).

From merging to matching (ctd.)

In this tree, each explicit list is built in time $2^{n/3}$.



Merging at the root



Before:

Two lists of $n/3$-bit collisions

$\Downarrow$

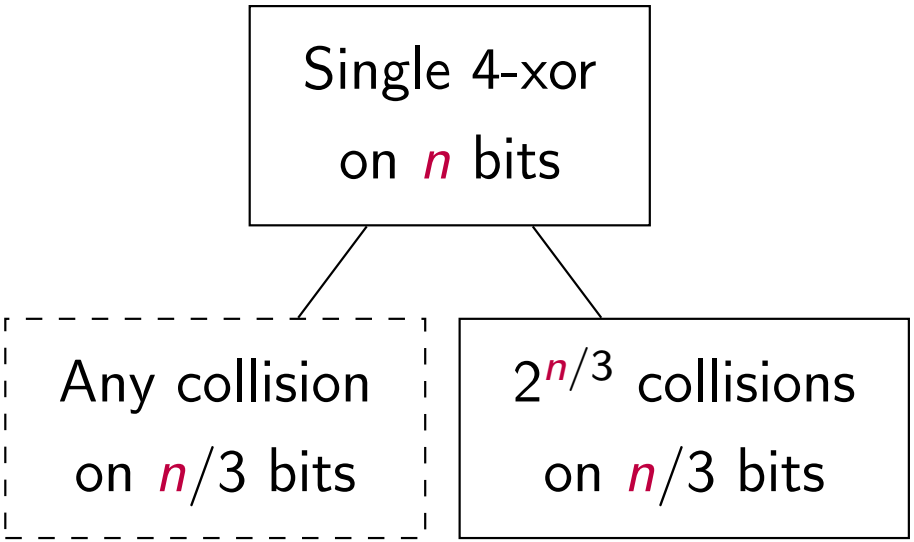
$2^{2n/3}$ $n/3$-bit 4-xors

$\Downarrow$

One n-bit 4-xor

In time $2^{n/3}$ (sorted lists).

Merging at the root



After:

A single list of $n/3$-bit collisions

$\Downarrow$

Produce $n/3$-bit collisions on the fly

$\Downarrow$

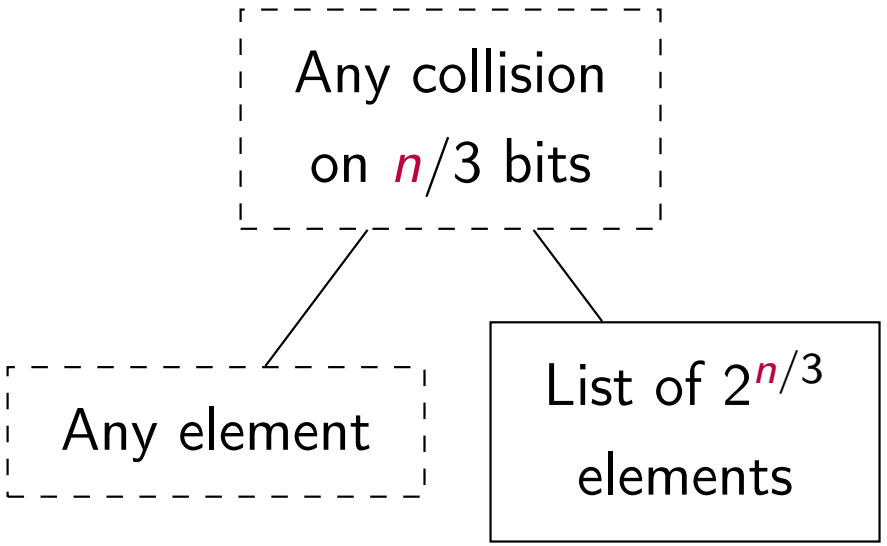
Each yields $2^{n/3}$ 4-tuples

$\Downarrow$

After $2^{n/3}$ trials, a n-bit 4-xor

In time $2^{n/3}$ (sorted list).

Partial collisions on the fly



A single list of $2^{n/3}$ elements

$\Downarrow$

Query H on the fly

$\Downarrow$

Each query yields $2^{n/3}$ pairs

$\Downarrow$

An $n/3$-bit collision

In time 1 (sorted list).

In this example

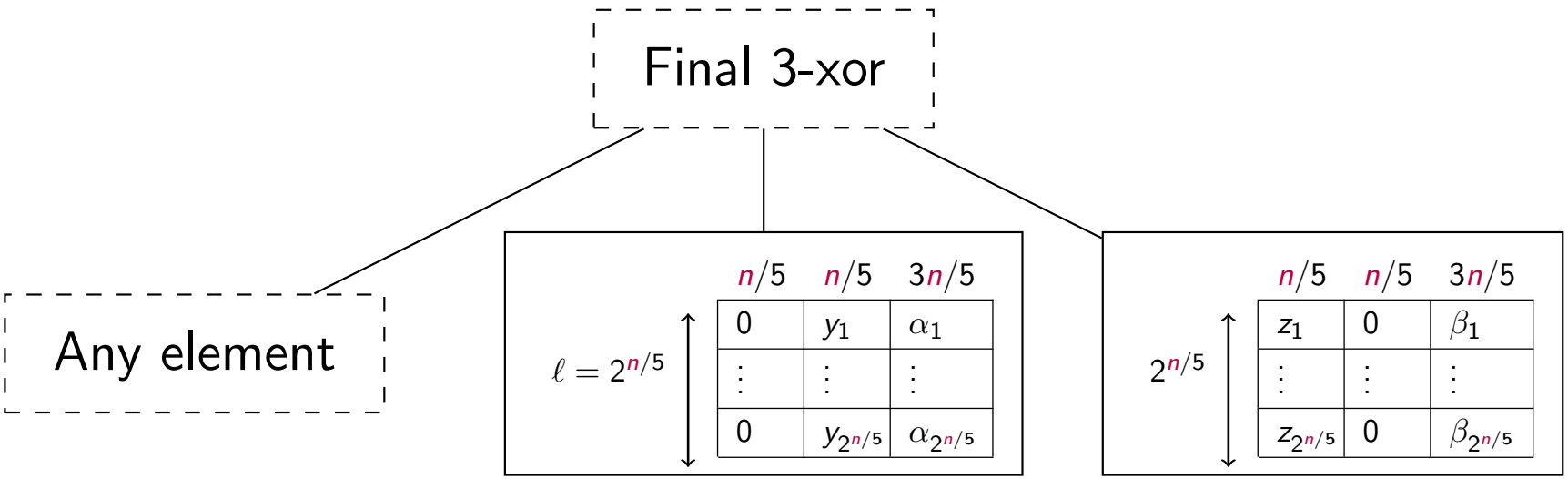
- Explicit (intermediate) lists are built in time $2^{n/3}$
- The last 4-xor is built by trying $2^{n/3}$ partial collisions
- . . . or trying $2^{n/3}$ elements

In this example

- Explicit (intermediate) lists are built in time $2^{n/3}$
- The last 4-xor is built by trying $2^{n/3}$ partial collisions
- . . . or trying $2^{n/3}$ elements
- We can use Grover search in the last step: time $2^{n/6}$
- . . . we should balance the tree: at total time $2^{n/4}$ in this example

Rephrasing previous algorithms

The 3-xor algorithms with two intermediate lists: trees of height 2.



- We found a better merging for 3-xor with qRAM: $\alpha_3 = \frac{2}{7} < \frac{3}{10}$
- (The low-qubits variant was optimal)

Finding the best trees: MILP

We fix the tree structure.

- Variables: sizes of the lists, their costs (in $\log_2$), prefixes
- Linear relations and constraints:
  - How we merge
  - How much this costs (classically or quantumly)
- An overall time complexity to minimize

Theorem – with qRAM

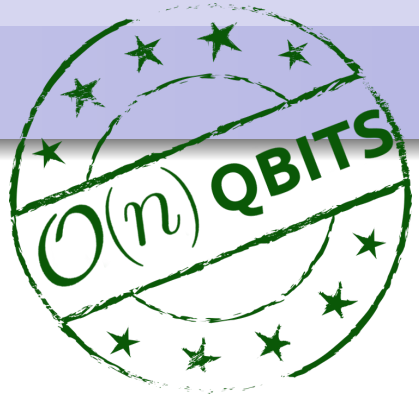
Theorem

If $\mathbf{k} \geq 2$ and $\kappa = \lfloor \log_2(\mathbf{k}) \rfloor$, the best merging-tree quantum time exponent is

$$\alpha_{\mathbf{k}} = \frac{2^{\kappa}}{(1+\kappa)2^{\kappa} + \mathbf{k}} \ .$$

Many trees give this time complexity, but one is obtained by using an "almost" binary tree.

Theorem – qRAM-free



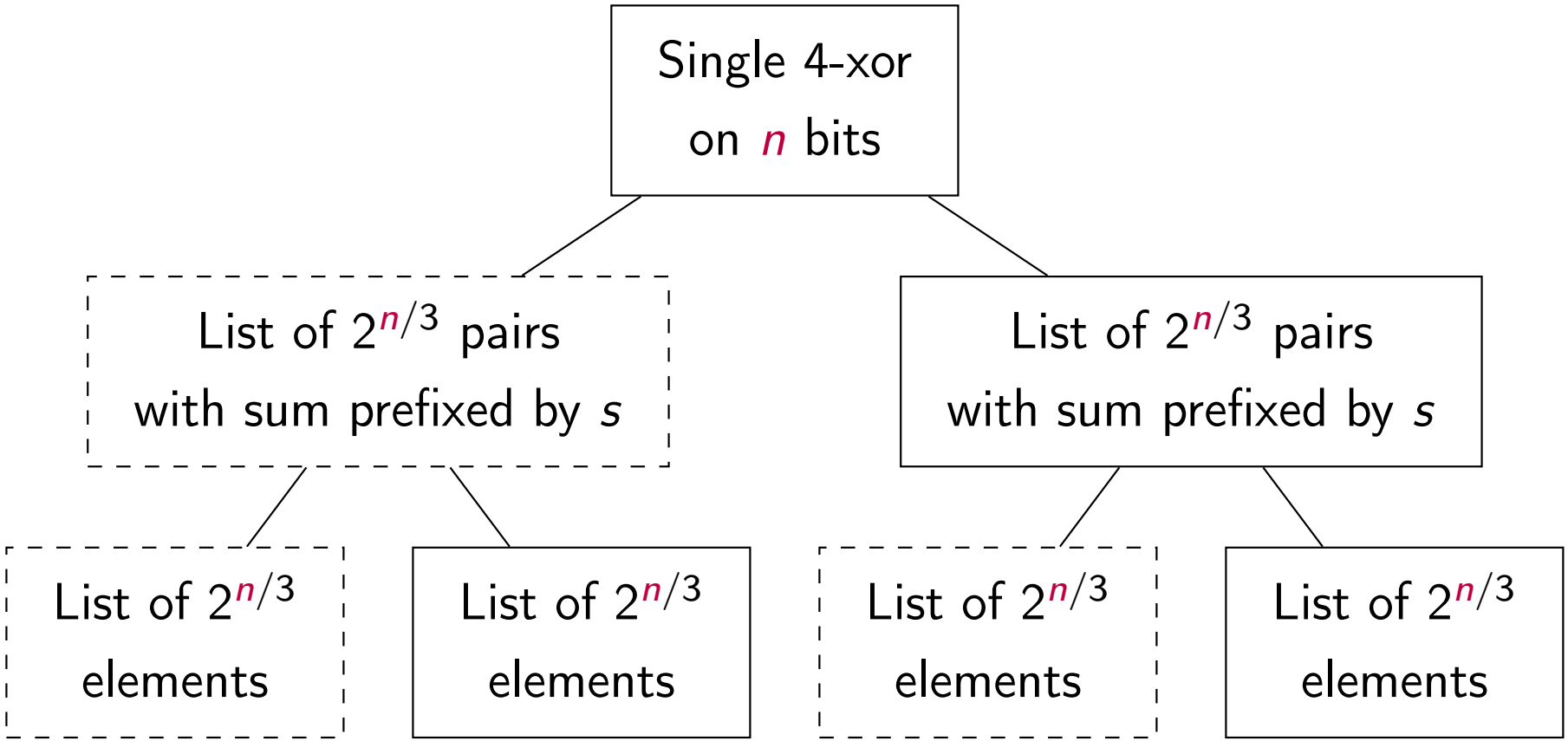
Theorem

If $\mathbf{k} > 2, \mathbf{k} \neq 3, 5$ and $\kappa = \lfloor \log_2(\mathbf{k}) \rfloor$, the best merging-tree quantum time exponent is:

$$\alpha_{\mathbf{k}} = \frac{1}{\kappa+1} \text{ if } \mathbf{k} < 2^{\kappa} + 2^{\kappa-1} \text{ or } \alpha_{\mathbf{k}} = \frac{2}{2\kappa+3} \text{ if } \mathbf{k} \geq 2^{\kappa} + 2^{\kappa-1}$$

Many trees give this time complexity, but one is obtained by using an "almost" binary tree.

Extending the merging framework



If the search space is too small, loop over the values of the prefix s.

Single-solution k-xor (Problem 3)

Given $\mathbf{k}$ lists of uniformly distributed n-bit strings, of size $2^{n/\mathbf{k}}$ each, find a k-xor on n bits.

- Previous work (Bernstein, Jeffery, Lange, Meurer, 2013): if $\mathbf{k}$ is a multiple of 4, time $\widetilde{\mathcal{O}}\left(2^{0.3n}\right)$ with a quantum walk.
- New: quantum time $\widetilde{\mathcal{O}}\left(2^{\beta_{\mathbf{k}} n}\right)$ with $\beta_{\mathbf{k}} = \frac{1}{\mathbf{k}} \frac{\mathbf{k}+\lceil \mathbf{k}/5 \rceil}{4}$, without a quantum walk.
  - Improves all $\mathbf{k}$ except multiples of 4
  - Meets 0.3 when $\mathbf{k}$ is a multiple of 5
  - Applies to $\mathbf{k}$-encryption

Conclusion

- We have found the optimal merging trees for quantum **k**-xor
- All of this works when replacing $\oplus$ by $+$
- We extended this to problems with less solutions and without quantum oracle access (Problem 1)

Future work / open questions

- Extend the framework (more techniques)
- Extend the cryptographic applications (approximate problems)

Open questions

- Quantum time complexity of collision search with $\mathcal{O}(n)$ qubits ("why 2/5?")
- Quantum time complexity of **k**-xor with a single solution ("why 0.3?")

Thank you.