

Low-gate Quantum Golden Collision Finding

Samuel Jaques¹, André Schrottenloher²

¹ Department of Materials, University of Oxford, United Kingdom

² CWI

Context

We study **generic quantum algorithms** for Golden Collision search.

- (Not the specifics of SIKE)

Problem: Golden Collision

Let $h : X \mapsto X$ be a random function. It has $\simeq |X|$ collisions pairs $(x \neq y, h(x) = h(y))$. Assume that there is a “golden” collision that we can recognize; find it.

Other variants:

- the function has a **single** collision pair and we’re looking for it (**element distinctness**)
- there are **two functions** f, g and we’re looking for a **claw** ($f(x) = g(y)$)
- If the functions are random, all of these problems are equivalent up to a constant factor.
- We may take into account the cost of h , but not in this presentation.

Classical methods

Problem

Given $h : X \mapsto X$ with a single golden collision ($|X| = N$), find it.

- **Lookup table:** query a lookup table of h , find all collisions ($T = N$ time, $M = N$ memory)
- **Limited lookup table:** query a table of M elements, examine all the collisions, repeat ($T = \mathcal{O}(N^2/M)$ time)
- **With limited memory:** use van Oorschot-Wiener distinguished point technique (M memory, $T = \mathcal{O}(N^{3/2}/M^{1/2})$ time)



van Oorschot and Wiener, "Parallel Collision Search with Cryptanalytic Applications", J. Cryptol. 1999



Adj, Cervantes-Vázquez, Chi-Domínguez, Menezes, Rodríguez-Henríquez, "On the Cost of Computing Isogenies Between Supersingular Elliptic Curves", SAC 2018

Outline

- 1 Cost Metrics
- 2 Quantum Walks
 - Prefix-based
 - Iteration-based
- 3 Parallelized Versions
- 4 Concluding Remarks

Cost Metrics

Cost metrics

An oversimplified view of the situation in [JS19] (Christian's talk last week).

The unit-cost “qRAM gate” does not exist

The unitary:

$$|x_1 \cdots x_m\rangle |i\rangle |y\rangle \mapsto |x_1 \cdots x_m\rangle |i\rangle |y \oplus x_i\rangle$$

costs $\tilde{O}(m)$ quantum gates (Clifford + T).

From now on we consider **baseline** quantum circuits with Clifford+T gates.

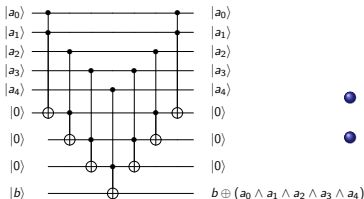


Jaques and Schanck, “Quantum Cryptanalysis in the RAM Model: Claw-Finding Attacks on SIKE”, CRYPTO 2019

Cost metrics (ctd.)

The total cost (“time complexity”) depends on the cost assigned to the **identity gate**: 0 or 1.

- ***G*-cost**: total number of gates
 - rationale: qubits can remain idle at no cost
- ***DW*-cost**: depth \times width product
 - rationale: qubits must be actively corrected at all times



- $G = 7$

- $D \times W = 7 \times 9$

Comparison with [JS19]

- Jaques and Schanck studied the known quantum algorithms for golden collision search (Grover, Ambainis) regarding G- and DW-cost.
- They concluded that it wasn't so good compared with classical vOW.
- In this paper we create new quantum algorithms tailored for the G- and DW-metrics (without unit-cost qRAM).
- Basically quantum walks / parallelized Grover where we re-optimize against the new cost of memory access.



Jaques and Schanck, "Quantum Cryptanalysis in the RAM Model: Claw-Finding Attacks on SIKE", CRYPTO 2019

Overview

G-cost

We can make a **quantum walk** (like Ambainis', but without qRAM gates), but it's memory-heavy.

- it follows the same time-memory trade-off, but stops earlier at gate cost $G = \tilde{O}(N^{6/7})$ instead of $\tilde{O}(N^{2/3})$.

DW-cost

We can make a parallelized Grover search with total depth-width $DW = N$.

Quantum Walks

Recap on quantum walks

The graph

A **Johnson graph** $J(X, R)$:

- Regular, undirected graph where vertices are subsets of size R of X .
- We go from a vertex to another by changing a single element

A vertex is “marked” if it contains the golden collision.



Ambainis, “Quantum walk algorithm for element distinctness”, SIAM J. Computing, 2007



Tani, “An improved claw finding algorithm using quantum walk”, MFCS 07

Cost of a classical random walk

We need procedures:

- To **setup** a starting arbitrary vertex (S)
- To **move** from one vertex to one of its neighbors (U)
- To **check** if a vertex is marked (trivial) (C)

We will find a marked vertex in time:

$$S + \underbrace{\frac{1}{\epsilon}}_{\epsilon \text{ proportion of marked vertices}} \left(\underbrace{\frac{1}{\delta}}_{\delta \text{ spectral gap of the graph}} U + C \right)$$

where $\frac{1}{\delta}$ is the number of updates before we reach a new uniformly random vertex. In a Johnson graph $J(X, R)$, $\frac{1}{\delta} \simeq R$.

Time of a quantum walk (Ambainis' or MNRS)

We need unitaries:

- To **setup** a uniform superposition over the graph
- To **move** from one vertex to the superposition of its neighbors
- To **check** if a vertex is marked (trivial)

$$S + \underbrace{\sqrt{\frac{1}{\epsilon}}}_{\text{Walk steps}} \left(\underbrace{\sqrt{\frac{1}{\delta}} U}_{\text{Mixing time}} + C \right)$$

In Ambainis' / Tani's algorithm:

$$R + \underbrace{\sqrt{\frac{N^2}{R^2}}}_{\text{Walk steps}} \left(\sqrt{R} \times 1 + 1 \right) = R + \frac{N}{\sqrt{R}} \geq N^{2/3}$$

With limited memory $R \leq N^{2/3}$: we have a trade-off $T^2 \cdot R = N^2$.



Magniez, Nayak, Roland, Santha, "Search via quantum walk", STOC 2007

The qRAM gate problem

A quantum walk like Ambainis' needs a cheap update operation U .

What does the classical U do?

- remove a random element from the vertex
- add a new random element to the vertex

What does the quantum U do? (basically)

- remove an element from the vertex **in superposition**
- add a new element to the vertex **in superposition**

This implies the ability to look at the whole memory R in time $\text{polylog}(R)$.

- It's OK if you have the unit-cost qRAM gate (see Stacey's thesis for the "quantum radix tree" data structure that supports these operations nicely).
- Otherwise, you need a circuit of $\tilde{O}(R)$ gates.

A walk without qRAM

Gate cost without the qRAM gate:

$$R + \underbrace{\sqrt{\frac{N^2}{R^2}}}_{\text{Walk steps}} \left(\sqrt{R} \times R + 1 \right)$$

- The update cost rises from

$$\underbrace{\mathcal{O}(1)}_{\text{Computing } h \text{ for the new element}} + \underbrace{\text{polylog}(R)}_{\text{Memory management}}$$

to

$$\underbrace{\mathcal{O}(1)}_{\text{Computing } h \text{ for the new element}} + \underbrace{R}_{\text{Memory management}}$$

- R is incompressible, so let's push more cost into the new element.

Idea 1: prefix-based walk

- We define subsets $X_D = \{x \in X, h(x) = D|\ast\}$ of **distinguished points** (of size θN).
 - Now we walk only on $J(X_D, R)$, a Johnson **subgraph** of $J(X, R)$.
- Upon update, we must find a new distinguished point: this new cost is going to balance with the memory cost R .
 - The proportion of marked vertices is increased: distinguished points have a higher probability to collide with each other.
 - Overall we win.

Complexity

Walking on the DPs:

$$\underbrace{R \times \frac{1}{\sqrt{\theta}} + R \log R}_S + \underbrace{\frac{N\theta}{R}}_{1/\sqrt{\epsilon}} \left(\underbrace{\sqrt{R}}_{1/\sqrt{\delta}} \underbrace{\left(\frac{1}{\sqrt{\theta}} + R \right)}_U + \underbrace{1}_C \right)$$

- Sorting the initial vertex: $R \log R$
- Updating the vertex: $R +$ finding a DP
- Finding a DP: $\frac{1}{\sqrt{\theta}}$ with Grover search
- Proportion of marked vertices: $\epsilon = \frac{(R^2)}{(N\theta)^2}$

We must find the good prefix: multiply this by $\frac{1}{\sqrt{\theta}}$

Complexity (ctd.)

For a given R :

- Optimal choice of θ is $\theta = \frac{1}{R^2}$
- Then the cost becomes roughly $R^3 + \frac{N}{\sqrt{R}}$
- When R is small, second term is dominant

When the memory is small, we run on the same trade-off curve $T^2 \cdot R = N^2$ as Ambainis' / Tani's.

- that's normal, because the dominant term is the same $\frac{N}{\sqrt{R}}$

However this stops at $G = \mathcal{O}(N^{6/7})$, when $R = N^{2/7}$. After that our DP set becomes too small and things go out of balance.

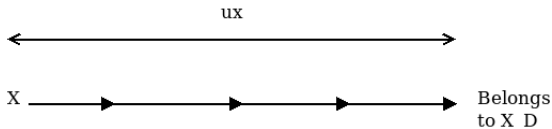
Idea 2: iteration-based walk

- We **change the definition** of distinguished points:

$$X_D = \{x, x = D|*\} \text{ for some prefix } D$$

- We **keep the same graph** $J(X, R)$
- We **change the function**:

$$h_D(x) = h^{u_x}(x) \text{ where } h^{u_x}(x) \text{ is the first iterate of } h \text{ belonging to } X_D$$

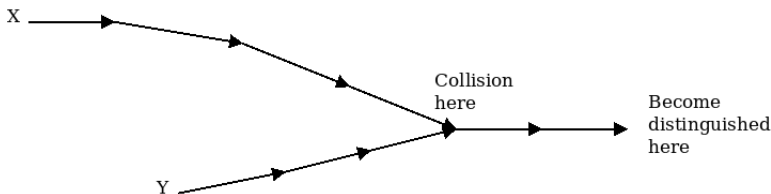


Now our vertex data structure contains R elements of the form $(x, h_D(x), u_x)$, which represent “trails” of length $\simeq \frac{1}{\theta}$.

This is actually a rewriting of the vOW DP technique as a random walk.

Idea 2: iteration-based walk (ctd.)

- A vertex is still marked if it **contains the golden collision**
- But now, collisions are **hidden in the trails**



If two $h_D(x) = h_D(y)$ collide in the vertex, we have: $h^{u_x}(x) = h^{u_y}(y)$. We recompute the trails to find the collision, and check if it is golden.

- Intuitively, the vertex is “compressed” (since we don’t store the whole trails),
- but we can still detect as many collisions as if it was “uncompressed”.

Updating a vertex

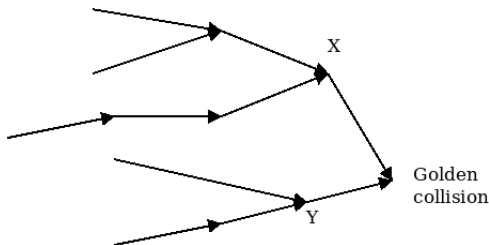
- ① we compute a new “trail” (in practice, we put a limit on the number of iterations, but it’s roughly $\frac{1}{\theta}$);
- ② we look at all the new collisions of h_D that occurred: on average $\frac{R}{N\theta^2}$ (**below 1 in practice**)
 - there are $\frac{1}{\theta}$ points on the new trail, and $\frac{R}{\theta}$ points on the current trails of the vertex, thus $\frac{R}{\theta^2}$ pairs which could collide
- ③ we recompute the trails ($\frac{1}{\theta}$ iterations) for each such collision, and we check if we have found the golden collision of h .

$$\text{Update cost: } U = \frac{1}{\theta} + R + \frac{R}{N\theta^3} .$$

Number of marked vertices

- A vertex contains R starting points.
- A vertex is marked if two of these starting points arrive at the golden collision.

So the proportion of marked vertices **depends on the number of predecessors** of both elements (x, y) of the golden collision $h(x) = h(y)$.



Theorem

- For a random function h
- For a pair (x, y) with $h(x) = h(y)$

The probability that both x and y have $\geq t$ predecessors is $\Theta(1/t)$.

Number of predecessors

- If we assume t predecessors, the probability of being marked is:
$$\simeq \frac{R^2 t^2}{N^2}$$
 - we must have one of the t predecessors of x ($\frac{Rt}{N}$) and one of the t predecessors of y ($\frac{Rt}{N}$)
- Actually we will find that $t \geq \frac{1}{\theta^2}$ is better, but then we bound the marked probability only by: $\geq \frac{R^2}{N^2 \theta^4}$
 - that's because we iterate h only $\frac{1}{\theta}$ times, so we can “see” only the predecessors at distance $\frac{1}{\theta}$ at most
 - among all the predecessors, $\Omega(\frac{1}{\theta^2})$ are in this case

(This bound comes from an analysis of random trees.)



Rényi, Szekeres, “On the height of trees”, J. of the Australian math. Soc. 1967

Complexity

The walk:

$$\underbrace{\frac{R}{\theta}}_S + \underbrace{\frac{N}{Rt}}_{1/\sqrt{\epsilon}} \left(\underbrace{R^{1/2}}_{1/\sqrt{\delta}} \left(\underbrace{\frac{1}{\theta} + R + \frac{R}{N\theta^3}}_U \right) + \underbrace{1}_C \right) .$$

- To ensure $t = \frac{1}{\theta^2}$ predecessors, we must re-randomize the function h .
- Thus we wrap the walk inside a Grover search with $\mathcal{O}(\sqrt{t})$ iterates.

Complexity (ctd.)

$$\sqrt{t} \left(\underbrace{\frac{R}{\theta}}_S + \underbrace{\frac{N}{Rt}}_{1/\sqrt{\epsilon}} \left(\underbrace{R^{1/2}}_{1/\sqrt{\delta}} \left(\underbrace{\frac{1}{\theta} + R + \frac{R}{N\theta^3}}_U \right) + \underbrace{1}_C \right) \right)$$

We choose $t = \frac{1}{\theta^2}$ and $R = \frac{1}{\theta}$:

$$R \left(R^2 + \frac{N}{R^3} R^{1/2} \times R \right) = R^3 + \frac{N}{\sqrt{R}} .$$

Same formula as before, so we recover the same time-space trade-off $T^2 \cdot R = N^2$ (also true if we take the cost of h into account).

Comparison

Prefix-based

- proportion $\frac{R^2}{N^2\theta^2}$ of marked vertices
- $\frac{1}{\theta}$ prefixes to search
- Cost $\frac{1}{\sqrt{\theta}}$ for an update

Iteration-based

- proportion $\frac{R^2}{N^2\theta^4}$ of marked vertices
- $\frac{1}{\theta^2}$ functions to search
- Cost $\frac{1}{\theta}$ for an update

- **Classically**, the advantage goes to iteration-based (vOW)
- **Quantumly**, we cannot use Grover search at the iteration step. The advantages cancel out because of that.

In general, iterations are not very useful in quantum collision-type algorithms.

Parallelized Versions

The depth-width cost

- We can parallelize the walks to meet some depth constraint, but it's painful and it does not give the best algorithms for the DW metric.
- So let's go back to Grover search instead, but a parallelized one.
- Here we will compute **with all the qubits, all the time**: $G = DW$ by design, up to poly factors.

On parallel memory accesses

Emulating a parallel RAM access

Consider P quantum processors with “free communication”.

- We can use them to store a memory of size P
- We can emulate P individual accesses to the memory in depth $\text{polylog}(P)$
- That is, if they **communicate freely**:

$$|y_0, i_0, 0\rangle \dots |y_{P-1}, i_{P-1}, 0\rangle \mapsto |y_0, i_0, y_{i_0}\rangle \dots |y_{P-1}, i_{P-1}, y_{i_{P-1}}\rangle$$

In a “baseline” quantum circuit, we can emulate P parallel qRAM gates spanning P registers in gate cost $\tilde{O}(P)$ and depth $\text{polylog}(P)$, because we use a **sorting network**.



Beals et al., “Efficient distributed quantum computing”, Proc. Royal Soc. London A: Mathematical, Physical and Engineering Sciences 469 (2013)

On parallel memory accesses (ctd.)

Let's assume first that every processor queries a different memory cell (thus $\{i_0, \dots, i_{P-1}\} = \{0, \dots, P-1\}$).

- ① For each processor we create two elements: “answer” and “query”:
 $(i, 1, y_i), (j_i, 0, 0)$
- ② We sort these $2P$ elements by lexicographic order
- ③ Each processor will keep its “answer” and have the corresponding “query”
- ④ We CNOT the answer on the query
- ⑤ We unsort

More generally

As many processors may query the same element, one must combine the **sorting step** with some **broadcast** where the answer of the query will be copied to all processors that queried it.



Beals et al., “Efficient distributed quantum computing”, Proc. Royal Soc. London A: Mathematical, Physical and Engineering Sciences 469 (2013)

Interlude: sorting networks

Beals *et al.*'s parallelization is powerful thanks to the existence of **data-oblivious sorting networks**.

- a sequence of comparators / swaps that are performed **independently of the values sorted**, and so, uses absolutely no RAM / qRAM access.

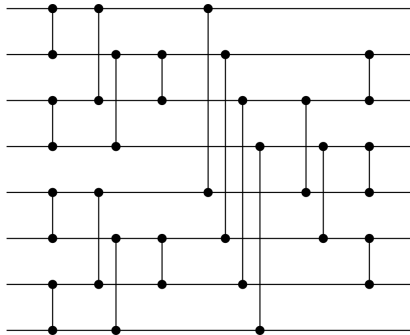


Illustration: the Batcher odd-even mergesort, by Octotron on Wikimedia Commons

Multi-Grover without locality constraints

Beals *et al.*'s algorithm with a circuit of width $\mathcal{O}(S)$:

- we take a set of S arbitrary elements $x_i \in X$ and store the values $(x_i, h(x_i))$
- we run S parallel Grover searches that look for y such that $\exists i, y \neq x_i, h(y) = h(x_i)$
- we wrap this under another Grover search, since the first half of the golden collision has to be in the initial set

Depth is: $\sqrt{\frac{N}{S}} \times \sqrt{\frac{N}{S}} = \frac{N}{S}$ and width is $\mathcal{O}(S)$, thus $DW = N$.



Beals *et al.*, “Efficient distributed quantum computing”, Proc. Royal Soc. London A: Mathematical, Physical and Engineering Sciences 469 (2013)

Multi-Grover without locality constraints (ctd.)

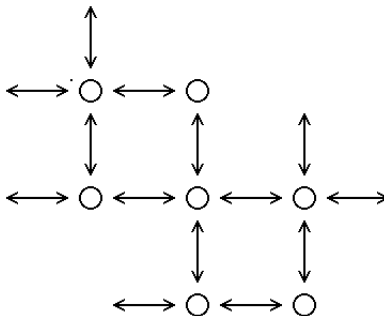
Alternative writing

We're searching for a set of S elements that contains the golden collision.

- $\sqrt{\frac{N^2}{S^2}} = \frac{N}{S}$ Grover iterates
- each iterate requires to create a new set (cost S , parallelized) and to check if it contains the golden collision (sorting network: $\tilde{O}(S)$, parallelized)

Locality constraints

- The “baseline” quantum circuit model assumes no cost for applying a two-qubit gate to any pairs of qubits.
- A popular “more realistic” assumption is to have the qubits “connected” in a **grid mesh**
- We can apply two-qubit gates only to adjacent qubits.



Local Multi-Grover

With a locality constraint, Beals *et al.*'s algorithm has an overhead corresponding to the cost of a sorting network on the circuit.

In the local case: sorting on the grid will cost $S^{3/2}$ gates and $S^{1/2}$ depth, so $DW = NS^{1/2}$.

$$DW = \underbrace{\frac{N}{S}}_{\text{Parallel Grover search}} \left(\underbrace{S}_{\text{Set}} + \underbrace{S^{3/2}}_{\text{Sorting}} \right).$$

We can improve on that.

Prefix-based multi-Grover

We reuse the **distinguished points** $X_D = \{x \in X, h(x) = D|*\}$. Now we can amortize the cost of sorting.

- we need $\frac{1}{\sqrt{\theta}}$ new external iterates
- the number of internal iterates is reduced from $\frac{N}{S}$ to $\frac{N\theta}{S}$
- computing the set costs $\frac{S}{\sqrt{\theta}}$

$$DW = \frac{1}{\sqrt{\theta}} \left(\frac{N\theta}{S} \left(\frac{S}{\sqrt{\theta}} + S^{3/2} \right) \right) .$$

- So we can choose $\theta = \frac{1}{S}$ and obtain $DW = N$ again.
- This works until $S = N^{1/2}$
 - at this point, the method consists in choosing a prefix ($N^{1/4}$ iterates), then computing the $N^{1/2}$ DPs in parallel ($N^{3/4}$ total work) and sorting.

Concluding Remarks

On the NIST security levels

Not in this presentation, unless you insist.

- **Levels 1, 3, 5:** Should be **as hard as** exhaustive key-recovery on AES-128, -192, -256
- **Levels 2, 4:** As hard as collision search on SHA3 (256 or 384 bits)
⇒ requires to study generic problems for these tasks (Grover & collisions)

Once we agree on corresponding G- and DW-costs (**not trivial**), we can try to compare our algorithms.


On the NIST security levels (ctd.)

Polynomial factors?

- Not from Grover search (just a constant)
- Not from Ambainis-style quantum walks on Johnson graphs (see the analysis by Childs & Eisenberg)
- Sorting networks on the **local grid mesh** induce some constants

The cost of isogeny computations?

- h computes a sequence of $\frac{1}{4} \log_2 p$ 2-isogenies.
- We can take the cost of h into account in the formulas, but it becomes heavier to write.

 Jaques, Naehrig, Roetteler, Virdia, "Implementing Grover Oracles for Quantum Key Search on AES and LowMC", EUROCRYPT 2020

 Childs, Eisenberg, "Quantum algorithms for subset finding", Quantum Inf. Comput. 2005

On the NIST security levels (ctd.)

The NIST introduced a parameter “Maxdepth” which is the maximum depth of quantum circuits (it goes from 2^{40} to 2^{90}).

- the “generic” G- and DW-costs now depend on Maxdepth
- the question becomes whether **at some point**, the attack is better than the intended level of security
- small values of “Maxdepth” incur insane amounts of parallelization in all cases

In general and without Maxdepth:

- The prefix-based walk wins on G-cost;
- Multi-Grover wins on DW-cost;
- None of this breaks SIKE parameters anyway, since we’re closer to the costs in [JS19] than “free qRAM”.

On the use of function iterates

- Classical algorithms take advantage of iterating the function h to reduce the memory used.
 - The best example is certainly classical collision search in $T = \mathcal{O}(N^{1/2})$ and $S = \text{poly}(\log N)$, using cycle-finding.
-
- Because function iterates are not quantumly accelerated, it might be impossible to go below the time-space tradeoff curve $T^2 \cdot M = N^2$ (and $T^2 \cdot M = N$ for the many-solutions case).
 - If so, then another interesting question is how much we can get rid of qRAM gates.

On the use of random-access gates

Let's picture a **classical algorithm** for golden collision search using N time and space.

- it makes N queries,
- then sorts,
- then finds the golden collision.

Classical collision algorithms or generalizations can be rewritten as **circuits without RAM access**, using sorting networks.

On the use of random-access gates (ctd.)

Classical algorithms

- They make many memory queries
- But these queries are independent, because exhaustive search is **stateless**
- Thus they can be “batched” and answered with a single sorting network

Quantum algorithms

- They make many queries
- But these queries are **not** independent, since quantum search (and derivatives) is stateful
- Thus they can’t be batched and we can’t use sorting networks

Conclusion

- Balancing with DPs was a classical algorithmic solution.
- If we want to move further down on the curve (or away from it), maybe we need a new quantum algorithmic idea.

Otherwise, we need more isogeny-specific thinking!

Thank you!