Chapter 3

Collision algorithms

3.1 Cryptographic hash functions

A cryptographic hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^n$ must satisfy the following properties.

- One-wayness: for a random $y \in \{0,1\}^n$, it should be hard to find x, such that $\mathcal{H}(x) = y$.
- Second-preimage resistance: for a fixed x, it should be hard to find $x' \neq x$ such that $\mathcal{H}(x) = \mathcal{H}(x')$.
- Collision resistance: it should be hard to find $x, x' \neq x$ such that $\mathcal{H}(x) = \mathcal{H}(x')$.

Brute force complexity of those algorithms classically: One-wayness: $O(2^n)$, second-preimage: $O(2^n)$, collision resistance: $O(2^{n/2})$.

Quantum brute force complexity: One-wayness: $O(2^{n/2})$, second-preimage: $O(2^{n/2})$, collision resistance: ??.

Collision problem. Here, we fix the input space to be of n bits as well, and we consider a random function f.

Collision problem
Input: a random function $f : \{0,1\}^n \to \{0,1\}^n$. Goal: find (x, y) such that $f(x) = f(y)$ and $x \neq y$. Because f is random, many such couples.

3.2 A classical algorithm

1. Pick a parameter r. Pick a random subset $I \subseteq \{0,1\}^n$ of size r. Construct the list $L = \{f(i)\}_{i \in I}$ and sort it. If we find 2 values $i, j \in I$ such that $f(i) = f(j)$ and $i \neq j$, output (i, j). Else:
2. Compute $f(x)$ for random values $x \notin I$ and for each such $f(x)$, we test if $f(x) \in L$. We continue until we find $f(x) \in L$. Since f is random, each $x \notin I$ will satisfy $f(x) \in L$ with probability $\frac{r}{2^n}$. We need to test on average $O(\frac{2^n}{r})$ to find one with high probability.

This algorithm uses the well known birthday paradox. The first step takes time $O(r \log(r))$ and the second one $O(\frac{2^n \log(r)}{r})$. By taking $r = 2^{n/2}$, we have an algorithm running in time $O(\mathsf{poly}(n)2^{n/2})$.

There are also algorithms that use a smaller amount of memory.

3.3 Quantum algorithms

3.3.1 A naive way

Consider the function $g(x,y) = 1$ if $(f(x) = f(y) \wedge x \neq y)$ and $g(x,y) = 0$ otherwise. We apply Grover on g. Because f is random, there are on average $O(2^n)$ couples (x,y) such that $g(x,y) = 1$. We perform Grover on g which takes time $O(\sqrt{\frac{2^{2n}}{2^n}}) = O(2^{n/2})$. This is not significantly better than the classical algorithm.

3.3.2 The quantum BHT (Brassard,Hoyer and Tapp) Algorithm

1. Pick a parameter r. Pick a random subset $I \subseteq \{0,1\}^n$ of size r. Construct the list $L = \{f(i)\}_{i \in I}$.
2. Let $g : \{0,1\}^n \to \{0,1\}$ satisfying $g(x) = 1 \Leftrightarrow [x \notin I \text{ and } \exists i \in I, f(x) = f(i)]$. $\approx r$ solutions when $r \ll 2^n$.
3. Apply Grover on g. Find y such that $g(y) = 1$.
4. Find $i \in I$ st. $f(i) = f(y)$. Output (i,y).

Time analysis of BHT Let $Time(f)$ the time required to compute f.

- List creation: time $Time(f) * r\log(r)$.
- Time to compute O_g : classically, g takes time $n\log(r)$. Can we use the standard reduction to construct O_g from g?

⚠We have to be careful here, because we don't actually have a classical circuit that runs in $n\log(r)$. We use a RAM that will perform the dichotomic search. We will assume for now that this is not a problem.

- Time to compute O_g : $n\log(r)$.
- Grover on g. g is a function from $\{0,1\}^n \backslash I \to \{0,1\}$. The input space is of size $2^n - r$ and there are on average r solutions. So we need to iterate Grover $\approx \sqrt{\frac{2^n - r}{r}}$ to find a solution to g. We define $|\psi_1\rangle = \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} |x\rangle|g(x)\rangle$

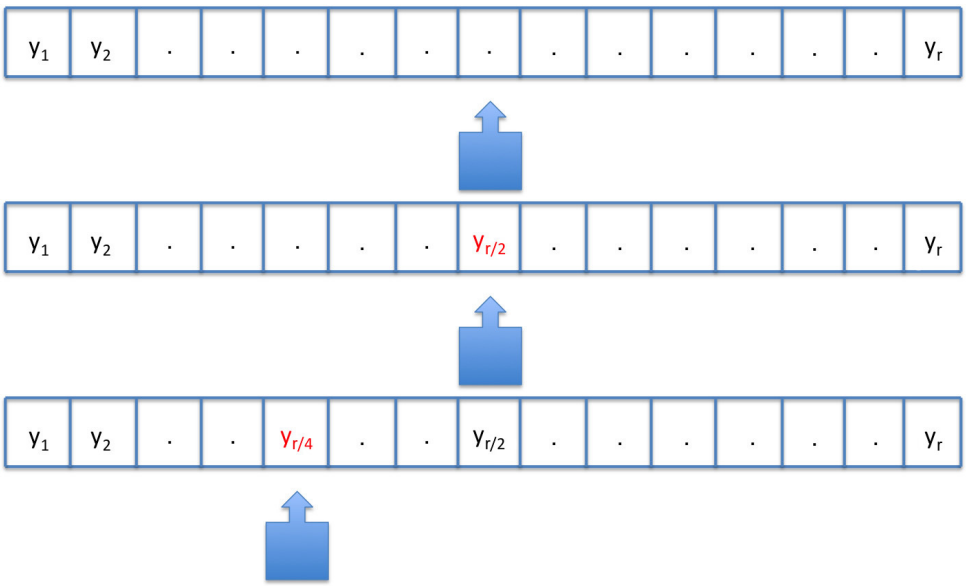
$$QTime(BHT) = \underbrace{Time(f) * r\log(r)}_{\text{List L creation}} + \underbrace{\sqrt{\frac{2^n - r}{r}}}_{\text{nb. of Grover iterations}} \left(\underbrace{n\log(r)}_{QTime(Ref_{|\psi_1\rangle})} + \underbrace{1}_{Ref_{|\psi_{\text{Bad}}\rangle}} \right).$$

If we take $r = \frac{2^{n/3}}{Time^{2/3}(f)}$, we obtain

$$QTime(BHT) = \widetilde{O}(Time^{1/3}(f) \cdot 2^{n/3}).$$

On quantum RAM, and on how to compute O_g.

The classical algorithm to compute g has a sorted list $L = (y_1, \ldots, y_r)$ that contains the values $f(x)$ for $x \in I$ and performs a dichotomic search on the memory.



We can perform this algorithm classically if we have access to a RAM operation $C(i) = y_i$. And this takes time $O(n \log(r))$.

In the quantum setting, we want to construct the quantum membership oracle

$$O_L(|x\rangle|0\rangle) = |x\rangle|\text{“}x \in L\text{”}\rangle.$$

where “$x \in L$” is the bit which is equal to 1 if $x \in L$ and 0 otherwise. In order to perform this, we need:

- r quantum bits of memory each storing $|y_i\rangle$.
- to perform the quantum operation U such that $U(|i\rangle|0\rangle) = |i\rangle|y_i\rangle$. This essentially requires having a reading head that can be in a superposition of states. This operation is called the QRAM operation.

There is still some debate in the community whether such an operation is as hard as building a quantum circuit or significantly harder. We'll consider here that a QRAM operation is theoretically feasible but very hard to do in practice. In order to compute O_g, we are given a sorted list $L = (y_1, \ldots, y_r)$.

If we have access to the elements in L in quantum registers $|y_1\rangle, \ldots, |y_r\rangle$ as well as access to quantum RAM, then we can perform the membership oracle for L in time $O(n \log(r))$ via dichotomic search. The proof will be omitted here but the idea is that with RAM access, classical dichotomic search can be expressed as a classical circuit of size $O(n \log(r))$. Similarly as in the transformation from classical circuits to quantum circuits, we can create the quantum membership oracle from dichotomic search.

If we don't have access to that QRAM, then the above can be done in time nr by checking each element separately. This is way worst for very large lists like in our example.

Quantum collision protocols without QRAM and with small quantum memory. Since 2016, we know that there exists a quantum algorithm that runs in time $O(2^{2n/5}poly(Time(f)))$ that solves the collision problem with n qubits and no QRAM.

Overview of known quantum algorithm for cryptography Here is an overview of quantum algorithms so far.

Algorithm	Qtime	Qspace	Applications
Shor (factoring)	poly(n)	n	RSA
Discrete log (elliptic curves)	poly(n)	n	Diffie Hellman, ECDH, ECDSA
Dihedral subgroup (solves SVP)	subexp(n)	n	NTRU, lattice based
Grover	$2^{n/2}$	n	generic (hash functions, AES, ...)
Collision	$2^{n/3}$ or $2^{2n/5}$	$2^{n/3}$ or n	generic (hash functions, symmetric)