Collision Finding Algorithms

Name: BOKKA UDAYA PEDDIRAJU
ID Number: 01-02-04-10-51-21-1-19424

1 Summary

Collision resistance is a major requirement in any cryptographic hash function, which is fulfilled by schemes like SHA-256. collision resistance property states that given a hash function H, it is hard to find two inputs such that H hash to the same output (basically it is infeasible to find any two distinct preimages for a given hash digest), it means that the security of the hash functions depends on the hardness of the collision finding problem. So collision finding is considered as an attack which is practically difficult to break, there are various algorithms proposed to find collisions in both classical and quantum, the main idea is to minimize the resources required especially the quantum algorithm for collision problem proposed by Gilles Bassard, Peter Hoyer and Alain Tapp was claimed to be the resource efficient taking time on the scale of $2^{n/3}$, let's call this algorithm as BHT algorithm. The hardware cost of this algorithm is also on the scale of $2^{n/3}$, but the claim is wrong because there are better non-quantum algorithms which are resource efficient than BHT, therefore the available quantum algorithms do not give any improvement than the available classical algorithms which was proved by Daniel J. Bernstein in the paper Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete?.

2 Collision Problem

Any function F is called as r to one if its outcome has exactly r distinct preimages, F is assumed to be a black box i.e., function description is unknown. If X is the domain of the function, then N=$|X|$ is the cardinality of the domain where N=2^n (n=number of bits in the input string). What is collision in F? a collision occurs when r distinct input values to F produce the same output value.

Guessing a collision in a hash function: a collision in H (hash function) which is constructed from the function F is just guessing a pair (x,y) such that $x \neq y$ and H(x) = H(y), the simplest way is to choose a pair from the domain of H and see whether it's a collision or not. If H maps (b+c) bit strings to b bit string where $c \geq 1$, if x and y are uniform random (b+c) bit strings then the chance that (x,y) is a collision in H is at least $1/2^b - 1/2^{b+c}$.

Proof: Let 2^b output values of H have $p_0, p_1, ..., p_{2^{(b-1)}}$ preimages (i.e., p_0 many preimages for 0th hash digest i.e, first output b-bit string of H) where $p_0 + p_1 + ... + p_{2^{(b-1)}} = 2^{(b+c)}$; note that each $p_i \geq 1$ then the number of collisions given by choosing any two preimage combinations from each preimage set under a hash digest.

$$
\begin{aligned}
&p_0(p_0 - 1) + p_1(p_1 - 1) + ... + p_{2^{(b-1)}}(p_{2^{(b-1)}}\text{-}1) \\
&= p_0^2 + p_1^2 + ... + p_{2^{(b-1)}}^2\text{-}(p_0 + p_1 + ... + p_{2^{(b-1)}}) \\
&\geq (p_0 + p_1 + ... + p_{2^{(b-1)}})^2/2^b \text{ - } (p_0 + p_1 + ... + p_{2^{(b-1)}}) \\
&= 2^{b+2c} \text{ - } 2^{b+c}
\end{aligned}
$$

by cauchy's inequality. A sequence of N independnt guesses succeeds with probability at least $1 - (1 - (1/2^b - 1/2^{b+c}))^N$ and involves at worst 2N computations of H. and in particular a sequence of $1/(1/2^b - 1/2^{b+c})$ independent guesses will succeed with probability more than 0.63 and involves 2^{b+1} computations of H at worst case.

According to the famous birthday paradox, if r=2 in an r to one function F then we can find a collision in F in approximately $O(\sqrt{N})$ evaluations of F which was discussed with a classical algorithm for collision problem as a part of BHT algorithm. Let's discuss the available quantum algorithms to find collisions and their resource complexity later we see the non-quantum algorithms that are resource efficient than the available quantum algorithms.

Simon's algorithm for collision finding problem: According to Simon's period finding algorithm, we can find a period 's' such that F is either one to one or two to one. Note that here finding 's' implies finding a collision in F and s can be obtained in a polynomial in n many evaluations of F (of course, it depends on the Gaussian elimination), unfortunately, Simon's algorithm doesn't have any practical applications.

2.1 Brassard–Høyer–Tapp (BHT) algorithm

2.1.1 A Classical Algorithm for Collision Problem

The three steps involved in this algorithm are,

1) select a random subset $K \subseteq X$ of cardinality k= $c\sqrt{(}N)$; c here is a constant and $c \geq 1$.

2)compute the pair $(x_i, F(x_i))$ for each $x \in K$ and store it in a table then sort to search for a collision.

3)Finally output a collision in K with the hope there's one.

This algorithm returns a collision with a probability of at least 1/2 in $O(\sqrt{N})$ many evaluations of F from birthday paradox and assuming $(x_i, F(x_i))$ as a unit of space this algorithm uses $O(\sqrt{N})$ space.

2.1.2 Simple Quantum Algorithm for Collision Problem

Grover discovered a quantum search algorithm in 1996 to find an item from an unstructured database with a probability of at least 1/2 in approximately $O(\sqrt{N})$ many evaluations of F. Let's generalize Grover's algorithm as Grover(F, y) where F is an unknown function that gives y such that $F(x) = y$ our task is to find x with Grover(F, y) i.e., Grover(F, y) is like a subroutine which returns x which is the preimage of y. Note that the database should be created before applying the Grover subroutine.

Simple quantum algorithm for collision problem based on Grover subroutine: the steps involved in this algorithm are,

1)pick an arbitrary element, $x_0 \in X$; where X is a set that contains all the preimages.

2)compute x_1= Grover(H,1), where H: X $\rightarrow$ {0, 1}; H(x)=1 if $x \neq x_0$ and $F(x) = F(x_0)$ (in a two to one function $x = x_1$). Grover ISR returns x_1 in $O(\sqrt{(N/t)})$ evaluations of F and 't' here indicates the number of solutions, notice that Grover's method searches O(N/t) possible preimages in $O(\sqrt{(N/t)})$ steps. For a two to one function t=1. i.e, there's only one value for x which satisfies $H(x_1) = 1$.

3)Finally output the collision (x_0, x_1). This simple quantum algorithm takes approximately $O(\sqrt{N})$ many evaluations of F assuming the cardinality of F as N, and this algorithm reduces the space requirement to a constant value (just the value of x_0 and $F(x_0)$ need to be stored).

2.1.3 BHT Algorithm

By combining the classical algorithm discussed above and the simple quantum algorithm, the BHT algorithm was proposed and this was claimed to provide an improvement in efficiency by Bassard-Hoyer-Tapp, though there are better non-quantum algorithms that are discussed below in later sections. The main idea is to select a subset K of X and then use Grover ISR to find a collision (x_0, y) with $x_0 \in K$ and y $\in$ X excluding the elements in K. The number of evaluations of F and the space required depends on the cardinality of the subset K.

The steps involved in BHT algorithm are,

1) Pick an arbitrary subset K $\subseteq$ X of cardinality k. Construct a table L of size k with an assumption of each item in L holds a distinct pair $(x_i, F(x_i))$ with x $\in$ K i.e., the initial assumption of no collision in subset K.

2) Sort L according to the second entry in each item of L i.e, F(x_i) for searching collision. Check if L contains a collision, i.e., check if there exist any two distinct elements ($x_i, F(x_i)$), (x_j, F(x_j)) $\in$ L for which F(x_i) = F(x_j) where i $\neq$j. If there's a collision then our initial assumption is wrong so return collision and stop.

3)Use the definition of the subroutine Grover and compute y = Grover (H, 1). In simple words, define H(y) as 1 if F(y) is in the set {F(x_1), F(x_2),...,F(x_k)} while y is not in the set {x_1,x_2,...,x_k}. Define H(y) as 0 otherwise, here this subroutine involves one evaluation of F since we already know the set {F(x_1), F(x_2),...,F(x_k)}. (Note that x_0 is unique in K if it exists since we already checked that there are no collisions in L).

4)Now, using the obtained y find (x_0 , F(y)) $\in$ L. and Output the collision {x_0, y}. also notice that a preimage of 1 under H reveals the collision in F.

This algorithm returns a collision with O(k+$\sqrt{(N/k)}$) many evaluations of F since it evaluates F function k times in the first step and $\sqrt{(N/k)}$ is the time taken by the Grover subroutine. Also, it requires a space of O(k), if we choose the initial subset with k= $\sqrt[3]{N}$ then the algorithm requires approximately O($\sqrt[3]{N}$) many evaluations of F and uses O($\sqrt[3]{N}$) space. i.e., If k=$2^{n/3}$ then BHT takes O($2^{n/3}$) time. Imagine that there is some magical way to quickly carry out comparisons to a giant pre-computed list of targets F(x_1), F(x_2), ..., F(x_k). The series of $2^{n/3}$ quantum evaluations of H still includes a series of $2^{n/3}$ quantum evaluations of F, taking time on the scale of $2^{n/3}$. The hardware cost of this algorithm is also on the scale of $2^{n/3}$.

Now let's generalize the BHT algorithm for an r to one function. Collision finding in an r to one function follows the same procedure as two to one, but it takes O(k+$\sqrt{(N/rk)}$) evaluations of F and uses the same space i.e., O(k). if we choose k= $\sqrt[3]{N/r}$ then the algorithm takes approximately O($\sqrt[3]{N/r}$) evaluations of F and O($\sqrt[3]{N/r}$) space.

Note that if H is a (b+c) bit to b bit string hash function then by using the preimage search (Grover ISR) discussed before we can find a preimage by quantum search with $2^{b/2}$h quantum operations on O(h) qubits where h is the cost of evaluating the H. From the previous discussion, the complexity of the algorithm seems to be an improvement and the same was claimed by Bassard-Hoyer-Tapp but it's way back than the performance of the classical algorithms (parallel collision search and the rho method) introduced by Van Oorschot and Michael J.Wiener. The below figure between the number of steps vs machine size shows different algorithms and their costs. Considering the time as a resource constraint parallel rho algorithm is performing well compared to other well-known traditional and quantum algorithms.

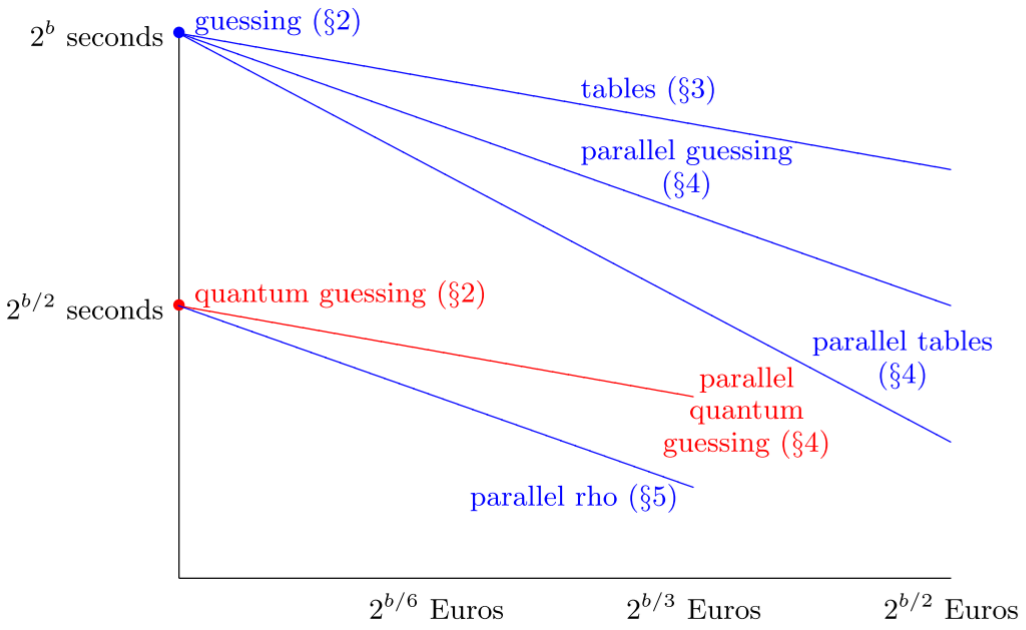

Figure 1: time vs cost of the machine

.

2.2 Non-Quantum Algorithms for Collision Finding and their Quantum Versions

2.2.1 Table lookups

The steps involved in this algorithm are as follows,

1) Choose M many inputs x_1, x_2, . . . , x_M from a subset of X.

2) Compute H(x_1), H(x_2), . . . , H(x_M), store the pair (H(x_1), x_1) in a table and lexicographically sort the M pairs (H(x_1), x_1), (H(x_2), x_2), . . . , (H(x_M),x_M).

3) Now choose the remaining N many inputs y_1, y_2, . . . , y_N from the set X excluding the elements that are chosen initially, Now compute H(y_i) and search for it in the table hoping to find a collision.

This attack has same effect as that of searching all MN pairs (x_i,y_j) for collisions H(x_i) =H(y_j), and it finds collision with a high probability of success in $2^{b/2+1}$(i.e., M+N) steps if M=N=$2^{b/2}$ and it requires M(2b+c) bits of memory.

Now let's look into the quantum advantage, a basic preimage searching algorithm by considering the cost of evaluating H and the access time of an element in the table, this algorithm states that first, we have to fix M inputs x_1, x_2, . . . , x_M from a subset of space X and define a function F such that F(y)=1 if there is a collision among (x_i,y); otherwise F(y)=0. This attack is all about finding a preimage of 1 under F. on the basis of this idea BHT algorithm was proposed and claimed to be the improved one with O(k+$\sqrt{(N/rk)}$)(T + log(k)) time complexity where N (i.e., 2^{b+c}) is the number of hash function inputs and N/r is the number of hash digests (i.e., 2^b), k is nothing but M and T is the cost of evaluating the hash function H, there fore time complexity can be further written as O(M + ($2^{b/2}$)/($M^{1/2}$))(h + log M)).

The problem with the BHT algorithm claim is it requires $2^{b/2}/M^{1/2}$ evaluations of F not merely H and also BHT requires comparing H(y) to the stored hash values in the table but if M is large then there are some issues with this algorithm during comparison.

1) Realistic two-dimensional models of quantum computation require time $M^{1/2}$ for random access to the table of size M which increases with M.

2) A traditional circuit to compare H(y) to all the hash digests in the table requires O(Mb) bit operations so a quantum circuit for the same operation requires O(Mb) qubit operations, sorting the table doesn't help the quantum circuit to reduce the comparison time as quantum search (i.e, Grover ISR) takes a superposition of b bit strings (i.e., hash digests) so it's output depends on all the Mb bits in the table.

Next, let's see the other available simple algorithms that are operating with the speed of BHT and the quantum advantage of those.

2.2.2 Parallelization

The idea is to build a machine of size M that finds collisions, here it consists of M small independent units for collision finding and all those machines run in parallel (i.e., H(x_1), H(x_2), . . ., H(x_M) and H(y_1), H(y_2), . . . , H(y_M) etc..). This machine produces a collision after O(2^b/M) many steps with high probability. In a realistic model of communication this algorithm has probability approximately $M^2/2^b$ of finding a collision, assuming $M \leq 2^{b/2}$. Repeating the same procedure $2^b/M^2$ times takes $2^b/M^{3/2}$ steps and has high probability of finding a collision; if M=$2^{b/3}$ then the time required is $2^{b/2}$.

Now, Let's consider a size M quantum computer that consists of M small independent collision searching units running in parallel, it finds a collision in $(2^{b/2}$ h$)/M^{1/2}$ operations. If M=$2^{b/3}$ then it can find collisions in $2^{b/3}$ steps and so on. Finally, the point here is, parallelization could also achieve the same performance as the BHT algorithm. But if we try to build machine M quantumly, consider a function F that takes 2Mb bits (x_1, x_2, . . . , x_M, y_1, y_2, . . . , y_M) and outputs 1 if and only if there's a collision in H. with the help of quantum search it could find the collision in O($(2^{b/2}$h)/M) i.e., a quantum computer(M) of size $2^{b/3}$ would be able to find collisions in time approximately $2^{b/6}$ in a naive model with cost-free communication, also by considering M pair, this algorithm has a chance of producing M^2 collisions in M operations (imagine what happens if all the chosen pairs collide) unlike M collision chance in the mindless parallelism discussed above. But the problem with this is that it has to evaluate h many times also it requires qubits instead of bits, but replacing bits with qubits is an inefficient way as it doesn't provide time-saving, so practically quantum parallelisation cannot be realizable.

2.2.3 The rho method

The rho method is used to find hash collisions without requiring a lot of memory, it uses a small hashing core that starts from some random input x and computes a series of hashes like H(x), H^2(x), H^3(x), etc. Here H^2(x) means H(H(x)); H^3(x) means H(H(H(x))); etc., (note that here I assumed that, hash digest and its preimage length are same which is different for below case i.e., (b+c) bit to b bit hash function was considered) the idea is to keep hashing the result of the previous hash operation. The important thing to note here is that there's no requirement of storing all the hash digests, the hash sequence will halt whenever it finds a distinguished point.

1)first choose a (b+c) bit string x_0 then compute the b bit hash digest H(x_0).

2)now apply an injective padding function pi to produce a b+c bit string x_1=π(H(x_0)), then compute H(x_1). Note: injective padding function is one that appends c zero bits to the b bit hash digest generated with the previous input string.

3)now compute x_2= pi(H(x_1)) then compute H(x_2) and so on. Repeat the process until you reach a distinguished point (the distinguished point is one which has b/2 bits as 0's i.e., the starting bits of the string).

4)Now consider another sequence y_0, y_1, . . . , y_j like x_0, x_1, . . . , x_i then follow the previous steps until you reach a distinguished point. (We can call each of such x_0, y_0 initial strings as cores and run the hash sequence (machine) in parallel by putting distinguishing points to each machine). There are 2^b inputs (x_i,y_j) before the distinguished points, so there's a possibility of a collision if so then their distinguished points will be identical; therefore the difference between sequence lengths i and j will reveal the collision (here the sequence i length is the number of steps taken before you reach the point of collision in x_i core machine and similarly j is for y_j core machine).

Now, let's take a few possible cases that may occur in this process, what if the hash sequence enters a cycle before reaching a distinguished point? For example, what happens if the sequence goes with four different non-distinguished points for input x i.e., H(x), H^2(x), H^3(x) and H^4(x) and if the output of H^4(x) is the same as H^1(x) then it forms a loop. To avoid this put a time limit on the computation, so that core won't cycle forever.

Here is the example of finding a collision: say $H^{77584}(x_7)$, is different from another, say $H^{92797}(y_{20})$, but the next hash value $H^{77585}(x_7)$ collides with $H^{92798}(y_{20})$ therefore the hash value after that i.e., $H^{77586}(x_7)$ equals $H^{92799}(y_{20})$, and so on. So x_7 and y_{20} will end up at the same distinguished point $H^{77584+s}(x_7) = H^{92797+s}(y_{20})$ as these are preimages for collided digest, for some positive integer s, assuming that both sequences run long enough to reach the distinguished point. We can find the collision by computing the difference 15213 of the counters and then checking $H^0(x_7)$ against $H^{15213}(y_{20})$, checking $H^1(x_7)$ against $H^{15214}(y_{20})$, etc. In other words, to find collisions of hash values, we simply need to find the collisions of distinguished points. This is useful because there are far fewer distinguished points than hash values.

3 Conclusion

Finally, we can conclude that the rho method takes much less hardware and also less time. Specifically, it reduces the time complexity approximately to $2^{(b/2)}$/M, where M is the number of sequence machines; if M=$2^{(b/3)}$ then the rho method takes $2^{(b/6)}$ steps to find a collision. Therefore, the BHT algorithm which has O($2^{(b/3)}$) time complexity is way back than the non-quantum algorithm the rho method and Bassard-Hoyer-Tapp's claim is proven to be wrong. since achieving $2^{(b/3)}$ improvement is easy with a machine size M=$2^{(b/6)}$. Even if one thinks about the rho method in the quantum version replacing qubits in place of bits doesn't give any time efficiency.