QUANTUM COLLISION FINDING FOR HOMOMORPHIC HASH FUNCTIONS

JUAN CARLOS GARCIA-ESCARTIN [1], VICENT GIMENO [2],
AND JULIO JOSÉ MOYANO-FERNÁNDEZ [2]

ABSTRACT. Hash functions are a basic cryptographic primitive. Certain hash functions try to prove security against collision and preimage attacks by reductions to known hard problems. These hash functions usually have some additional properties that allow for that reduction. Hash functions which are additive or multiplicative are vulnerable to a quantum attack using the hidden subgroup problem algorithm for quantum computers. Using a quantum oracle to the hash, we can reconstruct the kernel of the hash function, which is enough to find collisions and second preimages. When the hash functions are additive with respect to the group operation in an Abelian group, there is always an efficient implementation of this attack. We present concrete attack examples to provable hash functions, including a preimage attack to SWIFFT and collision finding for certain multiplicative homomorphic hash schemes.

1. QUANTUM ALGORITHMS IN CRYPTOGRAPHY

Quantum computing offers efficient algorithms that solve problems for which known classical methods are impractical. A prime example is Shor's algorithm for factoring and the discrete logarithm which runs in polynomial time [Sho97]. Many public key cryptographic protocols are based on these two, or closely related, problems. In order to prepare for future quantum computers, there is an active search of quantum resistant cryptographic systems, which are collectively known as post-quantum cryptography [BL17].

For many other classical cryptographic protocols, known quantum algorithms are of little or no consequence. For brute force search of the key space in symmetric cryptography, Grover's algorithm [Gro97] can only offer a quadratic speedup, which can quickly be solved by doubling the key length. Similarly, for ideal hash functions, quantum computer can only offer modest speedups [CNPS17].

While these general attacks are limited, there are still quantum attacks that are efficient against particular families of symmetric cryptosystems. For instance, symmetric ciphers based on the Even-Mansour construction become insecure in a quantum setting [KM12]

(1) DEPARTAMENTO DE TEORÍA DE LA SEÑAL Y COMUNICACIONES E INGENIERÍA TELEMÁTICA. ETSI DE TELECOMUNICACIÓN. UNIVERSIDAD DE VALLADOLID. CAMPUS MIGUEL DELIBES. PASEO BELÉN 15. 47011 VALLADOLID. SPAIN.

(2) UNIVERSITAT JAUME I, CAMPUS DE RIU SEC, DEPARTAMENT DE MATEMÀTIQUES & INSTITUT UNIVERSITARI DE MATEMÀTIQUES I APLICACIONS DE CASTELLÓ–IMAC, 12071, CASTELLÓN DE LA PLANA, SPAIN.

E-mail addresses: juagar@tel.uva.es, gimenov@uji.es, moyano@uji.es.

Date: July 30, 2021.

and certain common modes of operation in authentication and authenticated encryption can be attacked with quantum period finding [KLLNP16].

In this paper, we show that certain families of cryptographic hash functions that are additive or multiplicative are vulnerable to quantum attacks. These functions are sometimes the basic element in homomorphic hash applications [KFM04].

2. Cryptographic hash functions

An ideal hash function is a function $H(x) = y$ which takes an input binary string of an arbitrary length x into an output $y \in \{0,1\}^n$ with a fixed number of bits n. Depending on the intended use, there are many definitions of what constitutes a proper cryptographic hash function. Some common requirements, in a broad formulation, are [MVO96]:

- *Collision resistance*: It should be infeasible to find two values x, x' with $x \neq x'$ such that $H(x) = H(x')$.
- *Preimage resistance*: For a fixed hash value y it should be infeasible to find a string x such that $H(x) = y$.
- *Second preimage resistance*: For a fixed input x it should be infeasible to find a second string x' such that $H(x) = H(x')$.

In practice, we can consider ideal hash functions as random transformations that take any input x into a random string of n bits and for which even the smallest change in x (1 flipped bit) results in completely new output (which has, on average, only half bits in common with the first hash).

We present a second preimage attack for hash functions that are additive or multiplicative (see Section 2.1). This automatically gives a family of collisions. With a number of operations polynomial in the number of input bits we can find an exhaustive list of collisions for any desired input.

2.1. **Homomorphic hash functions.**

A general hash function works on lengths of an arbitrary ouput. In the following we are adopting a definition with a fixed input size:

Definition 2.1. An m-to-n hash function $H(x) : \{0,1\}^m \to \{0,1\}^n$ is a function that takes an m-bit string x into an n-bit string y with $m > n$.

In the following, we will use the term hash function to talk about m-to-n hash functions. This covers some existing fixed-size hash functions and the general case, where we have to restrict to inputs of the same size as the string for which we want a collision. In both cases we can obtain a valid collision (or a second preimage).

Hash functions in cryptography should be inversion, collision and preimage resistant. In many cases, this resistance is assumed from the statistical mixing inside the function. However, in the functions generally known as *provably secure* hash functions, resistance to attacks is founded on reductions to assumed hard problems (like factoring or the discrete logarithm problem). Proofs are possible because of an additional imposed structure on the functions. Similarly, for some applications like homomorphic encryption, there are additional properties which prevent the concerned hash functions to behave as fully random transformations. This is usually not a problem for many applications as long as we can keep collision resistance or similar properties.

Many provable hash functions have an additive or multiplicative property, depending on the group operation. These functions are defined by a homomorphism in that group.

Definition 2.2. A hash function $H(x)$ is **homomorphic** if, for any input pair x and y, $H(x+y) = H(x) + H(y)$ for the group operation $+$ in the input and output groups.

For instance, l-bit strings, together with the XOR operation, form an Abelian group so that a hash function $H(x \oplus y) = H(x) \oplus H(y)$ for the bitwise XOR for m (input) and n bits (output) is an additive hash function.

In the paper we will speak of additive functions and work with groups $(G,+)$. In some contexts, the most natural way to think of the group operation is as a product $\times$ (and to replace the null element by a unit element). Apart from this unimportant nomenclature issue, additive (or multiplicative) hash functions have the same behaviour and are subject to quantum attacks that can help to find collisions.

3. The hidden subgroup problem and hash collisions

The most notable quantum algorithms which offer superpolynomial speedups over classical known methods, like Shor's algorithm, solve instances of the hidden subgroup problem [BL95, Lom04, CvD10].

Definition 3.1. Let G be a finite group with a group operation $+$ which can be computed efficiently for any pair $x,y \in G$. Let $f : G \to S$ be a function on the group for some set of values S that defines a subgroup $H = \{k \in G : f(k+g) = f(g) \text{ for all } g \in G\}$. **The hidden subgroup problem** consists in finding a set of generators of this $H < G$ given f and G.

For Abelian groups, quantum computers can solve the hidden subgroup problem efficiently [Lom04].

The additive hash functions of Definition 2.2 take the elements of a Boolean group $G = (\{0,1\}^m,+)$ to the set $S = \{0,1\}^n$ and play the role of the hidding function f. For any $x,y \in \{0,1\}^m$, $H(x+y) = H(x)+H(y)$.

For the collision attack, we consider the hidden subgroup K defined by elements y for which $H(y)$ is the identity element with respect to $+$ in $\{0,1\}^n$. We call this identity element e the null element of the sum and denote it by $\mathbf{0}$. The subgroup K is the kernel of the hash function H.

The additive hash functions we consider are group homomorphisms and their kernel is a normal subgroup of G [Lau03].

If we can find an element in the subgroup $K = \{y_i \mid H(y_i) = \mathbf{0}\}$ we have a preimage attack. $H(x+y_i) = H(x)+H(y_i) = H(x)+\mathbf{0} = H(x)$ and we have two inputs $x+y_i$ and x mapping to the same output. The only exception is the y_0 equal to the identity element in the origin group $\mathbf{0}$, which is always an element of K. For hash functions, $m > n$ and the order (number of elements) of the input group is always greater that the order of the output set ($|G| > |S|$). The hash function can never be injective and the kernel has at least one element apart from the identity of G.

3.1. **Quantum computers.** In a quantum computer, we will represent each element in a group G with $|G| = M$ by states $|k\rangle$ with a label k for each integer $0 < k < M$. When $M = 2^m$ we can alternatively write the integer as the corresponding binary string. These states will form a basis for all the possible states $|\psi\rangle = \sum_{k=0}^{M} \alpha_k |k\rangle$ with complex α_k so that $\sum_{k=0}^{M} |\alpha_k|^2 = 1$. For binary strings, we can also write state $|k\rangle$ in terms of m individual qubits $|k\rangle = |k_{m-1}\rangle \cdots |k_1\rangle |k_0\rangle$.

All the operations on the state, except for measurement, are reversible and can be written as a unitary $M \times M$ matrix U. We use the usual notation $U_1 \otimes U_2$ and $U^{\otimes n}$ to denote the tensor product of the operations U_1 and U_2 and the U operation applied to n different inputs (of the corresponding dimension) respectively.

One particularly useful evolution on a single qubit is given by the Hadamard gate $H|x\rangle = \frac{|0\rangle + (-1)^x |1\rangle}{\sqrt{2}}$. Among other uses, it can be used to prepare uniform superpositions starting from an initial $|0\rangle \cdots |0\rangle |0\rangle$ state.

While quantum states can be in superpositions of multiple values, in order to retrieve information from the system we need to perform a measurement and we can only recover a single value. Thus the advantage of quantum computing lies not in superpositions alone but in being able to choose a quantum evolution U which results in a destructive interference for the states we are not interested in and a constructive interference between the states we want. For that reason, where quantum computers really shine is in problems with a strong hidden structure where we can extract global properties which are usually inaccessible to classical computers without heavy sampling (checking most of the possible values)

A more detailed description of quantum computing can be found in standard textbooks [NC00, Mer07].

3.1.1. *Quantum Fourier Transform in finite Abelian groups.* A key operation in quantum computers is the Quantum Fourier Transform, which helps us to produce the necessary constructive and destructive interference which reveals the solution we search for. In this Section, we describe its implementation for Abelian groups. We first need a few definitions.

Definition 3.2. A finite Abelian group $(G,+)$ has $|G|$ distinct one-dimensional irreducible representations called **characters**. A character is a multiplicative function $\chi : G \to \mathbb{C} \setminus \{0\}$ so that, for the $+$ operation in G, $\chi(x+y) = \chi(x)\chi(y)$ for any pair $x, y \in G$.

From the structure theorem for finite abelian groups, G can be written as a direct sum $G = \mathbb{Z}_{N_1} \oplus \cdots \oplus \mathbb{Z}_{N_k}$ of k cyclic groups $\mathbb{Z}_i$ of orders N_i. The elements $g \in G$ can be described as k-tuples $g = (g_1, \ldots, g_k)$ taking each g_j as an integer modulo N_j. The identity of G becomes $e = \mathbf{0} = (0,0,\ldots,0)$.

We can now define a decomposition in terms of each of the cyclic groups from the tuples $\beta_1 = (1,0,0,\ldots,0), \beta_2 = (0,1,\ldots,0), \ldots, \beta_k = (0,0,0,\ldots,1)$, with all these $\beta_j \in G$ [Lom04]. For any $g = (g_1, g_2, \ldots, g_k) \in G$

$$\chi(g) = \chi\left(\sum_{j=1}^{k} g_j \beta_j\right) = \prod_{j=1}^{k} \chi\left(\beta_j\right)^{g_j} \tag{3.1}$$

where the effect of χ on any g is completely determined from the values it takes on the β_j.

For each $g \in G$, we can define a character $\chi_g(h) = \prod_{j=1}^{k} \omega_{N_j}^{g_j h_j}$ for $h \in G$ and the roots of unity $\omega_{N_j} = e^{i\frac{2\pi}{N_j}}$.

For any fixed character of a finite Abelian group χ:

$$\sum_{g\in G} \chi(g) = \begin{cases} |G| \text{ if } \chi = \chi_e \\ 0 \text{ if } \chi \neq \chi_e \end{cases} \tag{3.2}$$

where χ_e is the identitity character which sends any $g \in G$ to 1.

We define a quantum Fourier transform over G, QFT_G from a character as the operator:

$$QFT_G \left|g\right\rangle = \frac{1}{\sqrt{|G|}} \sum_{h\in G} \chi_h(g) \left|h\right\rangle . \tag{3.3}$$

For a cyclic group $G = \mathbb{Z}_N$, the characters $\chi_h(g) = e^{i\frac{2\pi hg}{N}}$ are defined from the roots of unity. We can similarly compute the characters for any group that is a known direct product of cyclic groups.

We can build efficient quantum circuits giving the QFT_G operation for finite Abelian groups. For $G = \mathbb{Z}_2^n$, the most common group when working with binary strings, the operation $H^{\otimes n}$ (a Hadamard gate on each qubit) gives an efficient implementation. Similarly, for any cyclic group $G = \mathbb{Z}_N$, even for an unknown N, the Quantum Fourier Transform

$$QFT_G \left|x\right\rangle = \sum_{y\in G} \omega_N^{xy} \left|y\right\rangle \tag{3.4}$$

with $\omega_N = e^{i\frac{2\pi}{N}}$ can be computed efficiently (and is indeed the QFT used in Shor's algorithm) [HH00, Lom04].

For a group with a know factorization $G = \mathbb{Z}_{U_1} \times \cdots \times \mathbb{Z}_{U_{k-1}} \times \mathbb{Z}_{U_k}$ (using a direct product notation), there are also efficient constructions using the unitary evolution $QFT_G = QFT_{U_1} \otimes \cdots \otimes QFT_{U_{k-1}} \otimes QFT_{U_k}$ resulting from the tensor product of the QFT in each known cyclic group.

In fact, for any Abelian group, we can approximate the corresponding Quantum Fourier Transform and even use a simpler version that still works as expected for the Hidden Subgroup Problem using Fourier Sampling [HH00, CvD10].

The hash functions we review are all defined for finite Abelian groups, but there exist QFT generalizations which could help in additional problems [MRR06, GSVV01].

3.1.2. *Orthogonal subgroups and cosets.* We also used two important results related to any subgroup $H < G$.

Definition 3.3. For a subset $X \subseteq G$, we say $h \in G$ is **orthogonal** to X if $\chi_h(x) = 1$ for all $x \in X$.

Definition 3.4. For any subgroup $X < G$, the **orthogonal subgroup** $H^{\perp} = \{g \in G \mid \chi_g(h) = 1 \text{ for all } h \in H\}$ is the set of all the elements in G orthogonal to H. This $H^{\perp}$ is a subgroup of G and determines H uniquely.

Definition 3.5. Let H be a subgroup of $(G, +)$. For a fixed element $g_i \in G$, the **left coset** is the set $g_iH = \{g_i + h \text{ for all } h \in H\}$ and the **right coset** is $Hg_i = \{h + g_i \text{ for all } h \in H\}$. For an Abelian group both cosets are the same.

A key result for the Fourier Transform over Abelian groups is that it takes uniform superpositions from a subgroup H into a uniform superposition in the orthogonal subgroup $H^{\perp}$ [Lom04]

$$QFT_G\left(\frac{1}{\sqrt{|H|}}\sum_{h\in H}|h\rangle\right)=\frac{1}{\sqrt{|H^{\perp}|}}\sum_{h'\in H^{\perp}}|h'\rangle. \tag{3.5}$$

Proof. We have

$$\begin{aligned} QFT_G\left(\frac{1}{\sqrt{|H|}}\sum_{h\in H}|h\rangle\right) &= \frac{1}{\sqrt{|H|}}\sum_{h\in H}QFT_G|h\rangle=\frac{1}{\sqrt{|G||H|}}\sum_{h\in H}\sum_{g\in G}\chi_g(h)|g\rangle \quad (3.6)\\ &= \frac{1}{\sqrt{|G||H|}}\sum_{g\in G}\left(\sum_{h\in H}\chi_g(h)\right)|g\rangle. \quad (3.7)\end{aligned}$$

The character χ_g of G is also a character of H and the sum is 0 unless it is the identity on H, when it becomes $|H|$ (see Eq. (3.2)). That $\chi_g(h)=1$ for all the elements $h\in H$ is precisely the definition of the elements of the orthogonal subgroup $H^{\perp}$ (see Definition 3.4). So

$$\frac{1}{\sqrt{|G||H|}}\sum_{g\in G}\left(\sum_{h\in H}\chi_g(h)\right)|g\rangle = \frac{1}{\sqrt{|G||H|}}\sum_{g\in H^{\perp}}|H||g\rangle=\sqrt{\frac{|H|}{|G|}}\sum_{g\in H^{\perp}}|g\rangle, \tag{3.8}$$

which is a uniform superposition over $H^{\perp}$ which is G/H and has $\frac{|G|}{|H|}$ elements.

□

Assuming an Abelian group, which is the case for the additive hash functions under study, we call H_i to the coset $g_iH=Hg_i$. We are concerned with the Fourier Transform

$$QFT_G\left(\frac{1}{\sqrt{|K|}}\sum_{g\in H_i}|g\rangle\right)=\frac{1}{\sqrt{|K^{\perp}|}}\sum_{h\in K^{\perp}}\chi_h(g_i)|h\rangle \tag{3.9}$$

for any fixed g_i (representative) giving the coset H_i.

A quantum collision algorithm will sample random elements from H_i until it can deduce a generating set for K. Each element of $H^{\perp}$ gives one condition in a system of equations which completely describes H after sampling a number of orthogonal elements logarithmic with the size of G.

4. General collision algorithm

The tools from the previous sections allow us to define a general collision finding algorithm with the following steps:

- Prepare an initial state $|0\rangle|0\rangle$ with two registers, the first with m qubits, the second with n.
- Create a uniform superposition

$$\frac{1}{\sqrt{M}}\sum_{x=0}^{M-1}|x\rangle|0\rangle. \tag{4.1}$$

This can be done with a $H^{\otimes m} \otimes I^{\otimes n}$ or, depending on our group, $QFT_G \otimes I^{\otimes n}$.

- Call the hash oracle to transform the uniform superposition into
$$\frac{1}{\sqrt{M}} \sum_{x=0}^{M-1} |x\rangle |H(x)\rangle . \tag{4.2}$$
For binary strings, we use the usual unitary $U_f |x\rangle |0\rangle = |x\rangle |y \oplus f(x)\rangle$, which can be always implemented for functions f with an efficient classical implementation (as hash functions should). In other groups, such as the multiplicative group of integers modulo N, we can use modular addition. In general, for a $+$ operation in the image group of H, we have an efficient method to map the null element into $H(x)$.
- Measure the second register. The new quantum state is
$$\frac{1}{\sqrt{|K|}} \sum_{y_i \in K} |x_0 + y_i\rangle |H(x_0)\rangle . \tag{4.3}$$
We use that $H(x+y) = H(x) + H(y)$ for the $y_i \in K$. For $m > n$ (any useful hash function), there will be more than one value mapping to the same h. We call x_0 to the smallest such value.

  The result is a uniform superposition over the values $x_0 + y_i$ for all the y_i in the desired subgroup (the kernel of the hash function H). The second register can be ignored from this point.
- Compute the QFT_G of the first register in the corresponding Abelian group. The first register has a uniform superposition of the elements in the x_0K coset and the result will be a uniform superposition of the elements of the orthogonal subgroup $K^\perp$ with
$$QFT_G \left(\frac{1}{\sqrt{|K|}} \sum_{y_i \in K} |x_0 + y_i\rangle |H(x_0)\rangle \right) = \frac{1}{\sqrt{|K^\perp|}} \sum_{z \in K^\perp} \chi_z(x_0) |z\rangle |H(x_0)\rangle . \tag{4.4}$$
Before QFT_G, measuring the first register would only give an input/output pair. We exploit the hidden structure to force a destructive intereference for all the elements outside the orthogonal group.
- Measure the first register to find a random element of $K^\perp$ with equal probability.

This finishes the quantum part. Once we have a random sample of the orthogonal subgroup, we obtain a new restriction to the possible elements in the generating set of K. We repeat the process until we have enough information to find the whole generator. Strictly speaking, for a collision or preimage attack, it suffices to find one element $y_k \neq \mathbf{0} \in K$. We can stop as soon as we get the first random element of K which is not the identity. Then, for any input string x, we can generate a message $x \oplus y_k$ so that $H(x \oplus y_k) = H(x) + H(y_k) = H(x)$.

The method is efficient as long as:

- We can efficiently generate a uniform superposition over the group G.

  Typically, we need access to inputs which are arbitary binary strings (we can restrict to m bits with each attack) or integers in a range from 0 to N (usually converted from a binary string). In both cases it is easy to create the superposition

either from the $|0\rangle$ string and a Hadamard gate for each bit (input bits) or from the $|0\rangle$ state and the QFT as used in Shor's algorithm (integers).
- We have an efficient quantum function computing $H(x)$ for $x \in G$. The classical hash function must have a reasonable computation time in order to be useful. Any classical binary function can be converted into a reversible function if we keep the input and compute $|x\rangle|y\rangle \rightarrow |x\rangle|y \oplus H(x)\rangle$ for a bitwise XOR operation $\oplus$, which is enough to go from $|x\rangle|0\rangle$ to $|x\rangle|H(x)\rangle$.
- There is an efficient Quantum Fourier Transform. For Abelian groups, we have seen in Section 3.1.1 there are either efficient quantum circuits or good approximations which can still be used to find elements in the orthogonal subgroup.

In particular, for binary strings and the XOR operation, we have simple quantum circuits. The set of binary strings with m bits, together with the bitwise XOR operation, forms an Abelian group which can be written as $\mathbb{Z}_2 \times \ldots \times \mathbb{Z}_2$ with m factors. For this decomposition, in each of the cyclic groups associated to each bit, $\omega_2 = e^{i\frac{2\pi}{2}} = -1$ is a root of unity and the character $\chi_{g_i}(h_i) = (-1)^{g_i h_i}$ is a valid character for the possible values $g_i, h_i \in \mathbb{Z}_2$ that correspond to ith bits of g and h. Then, we have a valid character for m-bit strings and the XOR operation:

$$\chi_g(h) = \prod_{i=1}^{m}(-1)^{g_i h_i} = (-1)^{g\cdot h} \tag{4.5}$$

where $g \cdot h$ is the inner product on the bit strings representing g and h (the parity of the bitwise AND of the strings).

For this character, the quantum Fourier transform in the group can be written as

$$QFT_G|g\rangle = \frac{1}{\sqrt{M}}\sum_{h\in G}\chi_g(h)|h\rangle = \frac{1}{\sqrt{M}}\sum_{h\in G}(-1)^{g\cdot h}|h\rangle, \tag{4.6}$$

which corresponds to the quantum operation $QFT_G = H^{\otimes m}$ (applying a Hadamard gate to each qubit).

With this Fourier transform we get a random z in the subgroup orthogonal to the kernel, $z \in K^{\perp} \iff (-1)^{x\cdot z} = 1$ for all $x \in K$. Any two elements $z \in K^{\perp}$ and $x \in K$ obey $z \cdot x = 0$.

Each measurement gives a restriction to the possible values of the elements in $K^{\perp}$, which allows us to discover a generating set of $K^{\perp}$ after a number of measurements polynomial in the number of bits m.

Furthermore, once we have a generating set of $K^{\perp}$, we can compute a random element in K efficiently (polynomial time in m) and, from that, a generating set of K in expected polynomial time. The classical method is described in [Lom04, Dam04]. Basically, each measurement gives, with high probability, a new equation from a linear system which can be solved to obtain a generating set for K. For additive hash functions the system will always have a solution and the result can be used to find collisions or a second preimage to any input x. This completes the attack.

5. Examples

In this Section, we examine some hash proposals which would be insecure under our quantum attack. Somewhat ironically, these functions try to guarantee security against collisions by reduction to a hard problem, but the additional structure imposed on the functions allow for a quantum attack.

5.1. **$\oplus$-linear hash functions.** In [Kra94] Krawczyk presented two families of $\oplus$-linear hash functions $H(x) : \{0,1\}^m \to \{0,1\}^n$ which are additive with respect to the XOR operation. For any two inputs $x_1, x_2 \in \{0,1\}^m$, $H(x_1 \oplus x_2) = H(x_1) \oplus H(x_2)$.

The designs are based on Cyclic Redundancy Codes and Linear Feedback Shift Registers and have some some desirable properties. For instance, uniformity can be proved instead of assumed like in most hash functions. Unfortunately, the $\oplus$-linearity also opens the door for a quantum attack.

The attack is, basically, a quantum algorithm for a generalized Simon's problem. In the original Simon's algorithm we have a promise that a function $f(x) : \{0,1\}^n \to \{0,1\}^n$ such that $f(x \oplus s) = f(x)$ only for two values (with a secret string s). Here we have a slightly different problem. For a balanced function there will be 2^{m-n} strings with the same output value. The group is $(\{0,1\}^m, \oplus)$ and the hidden subgroup is the kernel of $H(x)$. After the quantum algorithm we get elements y_i with $H(y_i) = \mathbf{0}$ so that $H(x \oplus y_i) = H(x) \oplus H(y_i) = H(x)$.

5.2. **Pseudocollisions in SWIFFT.** Another proposal for provably secure hashes is based on the Fast Fourier Transform [LMPR08]. The proposal was a finalist for the NIST SHA-3 competion and has sometimes be presented as a good candidate for a quantum resistant hash [MR09].

SWIFFT functions are defined in families with three parameters: a power of 2, n, a small integer $m > 0$ and an integer $p > 0$ that will be used as a modulus (usually chosen to be prime for convenience). The functions act on the ring $R = \mathbb{Z}_p[\alpha]/(\alpha^n + 1)$ (the ring of polynomials in α with integer coefficients, modulo p and $\alpha^n + 1$). Any element of R can be written as a polynomial of a degree smaller than n with coefficients in $\mathbb{Z}_p = \{0, \dots, p-1\}$, giving a way to go back and forth between binary strings and the elements of R.

For the parameters n, m and p, a particular hash function from the family is specified by m fixed elements $a_1, \dots, a_m \in R$ (the multipliers) so that the function corresponds to the operation

$$\sum_{i=1}^{m} a_i \cdot x_i \in R \tag{5.1}$$

for the polynomials $x_1, \dots, x_m \in R$ derived from the polynomials with binary coefficients which correspond to the binary input string of length mn after multiplying them by fixed constants and taking the Fourier transform of the result.

Each polynomial multiplication $a_i \cdot x_i$ can be implemented efficiently with the Fast Fourier Transform (FFT). These multiplications are the most computation intensive part of the method.

For the ring R, any chosen hash function is linear, as $H(x+y) = H(x) + H(y)$ with respect to addition in the ring. Like all the hash functions discussed in this paper, this makes SWIFFT unsuitable for certain applications like working as a pseudorandom oracle, but this known limitation is not considered as an obstacle for its use when collision resistance is needed. The claim of collision resistance for SWIFFT is based on the assumed difficulty of the Shortest Vector Problem (SVP) in lattices [MG02, LMPR08].

The first step, which only accepts polynomials with binary coefficients, introduces additional protection and prevents a direct application of our quantum collision finding algorithm.

However, the ring $\mathbb{Z}_p[\alpha]/(\alpha^n+1)$ is, in particular, an Abelian group under addition and the hidden subgroup attack we have proposed will reveal a kernel with the elements y_i such that $H(y_i)=0$, which is enough to find *pseudocollisions*: for any input polynomial $x\in R$ with a hash $H(x)$, we can produce other polynomials with the same output string. We only need to choose polynomials that come from adding x to an element from the kernel, which result from any linear combination of the y_i found in the collision finding algorithm.

This falls short of a full collision. In order to find a complete collision we need to find two from these polynomials which correspond to valid inputs. The initial transformation is easy to invert, but not all the polynomials in R map to a valid binary input.

It is not clear if pseudocollisions can help to find true collisions, but other slightly different pseudocollisions have been studied before [BL09, Lin11]. Usually, they are considered an intermediate stage for a full attack and a proof of security against pseudocollisions gives an argument for the strength of the complete SWIFFT function.

5.3. **Homomorphic hash function with multiplication.** The attack can be translated to multiplicative hashes in groups where the group operation is more naturally cast as a multiplication and the null element as the unit.

We are going to see two examples with hashes in the multiplicative group of integers modulo N (the group of units in $\mathbb{Z}_N$). The group operation is multiplication modulo N and the identity element is the integer 1.

Our first example is the RSA hash $E(x)=x^e \mod N$ for an $N=pq$ with unknown factorization, which has a multiplicative property: $E(xy)=E(x)E(y)$. The proposed attack finds the kernel consisting in all the messages x_i for which $E(x_i)=1$. This particular example is not useful as a hash function. It depends on trusting no one knows the factorization of N.

However, multiplicative and additive properties appear in many proposals for homomorphic encryption and any hash function derived from them should be checked against quantum attacks.

For instance, the collision resistant hash function used in the homomorphic hash scheme proposed by Krohn, Freedman and Mazières [KFM04] is vulnerable to a quantum attack. The basic transformation is defined as

$$h_G(\mathbf{b}_j)=\prod_{i=1}^{m} g_i^{b_{i,j}} \mod p \tag{5.2}$$

for a message block b_j composed of m integers $b_{i,j}$ from 0 to a prime q dividing $p-1$. The integer p is a random prime and g_i are randomly chosen integers of order q modulo p. For any two blocks $\mathbf{b}_i$ and $\mathbf{b}_j$,

$$h_G(\mathbf{b}_i+\mathbf{b}_j)=h_G(\mathbf{b}_i)h_G(\mathbf{b}_j), \tag{5.3}$$

where $\mathbf{b}_i + \mathbf{b}_j$ is a vector with elements $b_{1,i} + b_{1,j} \mod q$ to $b_{m,i} + b_{m,j} \mod q$.

The inputs are in the additive group of integers modulo q and the hash takes them into the group of units modulo $\mathbb{Z}_p$. Finding a kernel for h_G gives blocks $\mathbf{b}_e$ with $h_G(\mathbf{b}_e) = 1$, which yield collisions for any desired input block $\mathbf{b}_i$.

The hash function is a compression function and the kernel will contain multiple elements. Most of them will be useful for collisions with two exceptions. First, the kernel will always contain a trivial zero block which maps each block to itself and for which all the $b_{i,j}$ are 0. Second, some of the blocks $\mathbf{b}_e$ might not correspond to valid binary sequences. The number of binary digits n for each block is chosen so that $2^n < q$. Unlike what happened in SWIFFT, the set of the integers mapping to a valid binary input is not vanishingly small.

6. DISCUSSION

We have shown quantum computers can find collisions for additive hash functions by finding its kernel subgroup. The attack is valid for hash functions with a strong structure, such as those usually proposed for provably secure hashing.

We have given examples of the attack working on the $\oplus$-linear hash functions of Krawczyk [Kra94] and its application to find pseudocollisions in the SWIFFT hash family [MG02] and in certain homomorphic hashing schemes [KFM04].

As opposed to some previous quantum decryption algorithms, which should have access to a quantum oracle encrypting with an unknown key, the attacker can always find a quantum version of the function and produce the required superpositions.

Like all collision attacks, quantum collision finding can be performed offline using any fixed input x_0. Once the kernel of $H(x)$ has been found, it can be directly used for second preimage attacks in real time to find $H(x') = H(x)$ by adding to the known input x any linear combination of the y_i in the kernel.

The kernel can also help to craft fake messages that replace a signed string. For instance, for the group of binary strings of n bits under the XOR operation, the attacker can try to alter specific bits from a message by XORing the input string with elements from the kernel that change only the target part of the message and, maybe, also unimportant bits which will not be noticed (like color or gray level bits in a picture). It is not obvious how to perform this kind of attack and it would be highly dependent on the particular structure of the kernel and the concrete addition operation of the relevant group, but it could reduce the complexity of a forgery, at least for some specific scenarios.

The attack exposes a general problem of hash functions: either there is a formal proof of security at the cost of imposing a structure or we are limited to transformations which appear to be random but are difficult to analyze.

In that respect, many provable hash functions use reductions to problems which can be solved efficiently on a quantum computer, such as factoring, and could be vulnerable to quantum attacks. The quantum security of these hash functions should be studied further. The attacks might not be straightforward. In many cases the reduction is not shown in both directions: finding a collision might solve factoring but it is not known whether factoring provides a collision or not.

It is also open whether the collision finding method of this paper can be extended or not to other provable hash functions with more complex additive or multiplicative properties. Some possible candidates are VHS [CLS06], where $H(\mathbf{0})H(x\vee y)\equiv H(x)H(y) \mod N$ for $x\wedge y=\mathbf{0}$, or the muHASH, adHASH and LtHASH families [BM97].

Acknowledgements

The first author has been funded by the Spanish Government and FEDER grant PID2020-119418GB-I00 (MICINN) and by Junta de Castilla y León (project VA296P18). The second author has been partially supported by the Research Program of the University Jaume I–Project UJI-B2018-35, as well as by the Spanish Government and FEDER grant PID2020-115930GA-I00 (MICINN). The third author was partially supported by the Spanish Government, Ministerios de Ciencia e Innovación y de Universidades, grant PGC2018-096446-B-C22, as well as by Universitat Jaume I, grant UJI-B2018-10.

References

[BL95] D. Boneh and R. J. Lipton, *Quantum cryptanalysis of hidden linear functions*, Advances in Cryptology — CRYPT0’ 95 (Berlin, Heidelberg) (Don Coppersmith, ed.), Springer Berlin Heidelberg, 1995, pp. 424–437.

[BL09] J. Buchmann and R. Lindner, *Secure parameters for SWIFFT*, Progress in Cryptology - INDOCRYPT 2009 (Berlin, Heidelberg) (B. Roy and N. Sendrier, eds.), Springer Berlin Heidelberg, 2009, pp. 1–17.

[BL17] D. J. Bernstein and T. Lange, *Post-quantum cryptography*, Nature **549** (2017), 188–194.

[BM97] M. Bellare and D. Micciancio, *A new paradigm for collision-free hashing: Incrementality at reduced cost*, Advances in Cryptology — EUROCRYPT ’97 (Berlin, Heidelberg) (Walter Fumy, ed.), Springer Berlin Heidelberg, 1997, pp. 163–192.

[CLS06] S. Contini, A. K. Lenstra, and R. Steinfeld, *VSH, an efficient and provable collision-resistant hash function*, Advances in Cryptology - EUROCRYPT 2006 (Berlin, Heidelberg) (Serge Vaudenay, ed.), Springer Berlin Heidelberg, 2006, pp. 165–182.

[CNPS17] A. Chailloux, M. Naya-Plasencia, and A. Schrottenloher, *An efficient quantum collision search algorithm and implications on symmetric cryptography*, Advances in Cryptology – ASIACRYPT 2017 (Cham) (T. Takagi and T. Peyrin, eds.), Springer International Publishing, 2017, pp. 211–240.

[CvD10] A. M. Childs and W. van Dam, *Quantum algorithms for algebraic problems*, Reviews Modern Physics **82** (2010), 1–52.

[Dam04] I. Damgård, *QIP note: on the quantum Fourier transform and applications*, Published on http://www. brics. dk/˜ ivan/Fourier.pdf (2004).

[Gro97] L. K. Grover, *Quantum mechanics helps in searching for a needle in a haystack*, Physical Review Letters **79** (1997), no. 2, 325.

[GSVV01] M. Grigni, L. Schulman, M. Vazirani, and U. Vazirani, *Quantum mechanical algorithms for the nonabelian hidden subgroup problem*, Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing (New York, NY, USA), STOC ’01, Association for Computing Machinery, 2001, pp. 68–74.

[HH00] L. Hales and S. Hallgren, *An improved quantum Fourier transform algorithm and applications*, Proceedings 41st Annual Symposium on Foundations of Computer Science, 2000, pp. 515–525.

[KFM04] M.N. Krohn, M.J. Freedman, and D. Mazieres, *On-the-fly verification of rateless erasure codes for efficient content distribution*, IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004, 2004, pp. 226–240.

[KLLNP16] M. Kaplan, G. Leurent, A. Leverrier, and M. Naya-Plasencia, *Breaking symmetric cryptosystems using quantum period finding*, Advances in Cryptology – CRYPTO 2016 (Berlin, Heidelberg) (M. Robshaw and J. Katz, eds.), Springer Berlin Heidelberg, 2016, pp. 207–237.

[KM12] H. Kuwakado and M. Morii, *Security on the quantum-type Even-Mansour cipher*, 2012 International Symposium on Information Theory and its Applications, 2012, pp. 312–316.

[Kra94] H. Krawczyk, *LFSR-based hashing and authentication*, Advances in Cryptology — CRYPTO '94 (Berlin, Heidelberg) (Yvo G. Desmedt, ed.), Springer Berlin Heidelberg, 1994, pp. 129–139.

[Lau03] N. Lauritzen, *Concrete abstract algebra: From numbers to Gröbner bases*, Cambridge University Press, 2003.

[Lin11] R. Lindner, *Towards efficient lattice-based cryptography*, Ph.D. thesis, Technische Universität, Darmstadt, January 2011.

[LMPR08] V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen, *SWIFFT: A modest proposal for FFT hashing*, Fast Software Encryption (Berlin, Heidelberg) (Kaisa Nyberg, ed.), Springer Berlin Heidelberg, 2008, pp. 54–72.

[Lom04] C. Lomont, *The hidden subgroup problem - review and open problems*, quant-ph/0411037 (2004).

[Mer07] N. D. Mermin, Quantum Computer Science, first ed., Cambridge, UK, 2007.

[MG02] D. Micciancio and S. Goldwasser, *Complexity of lattice problems - a cryptograhic perspective*, The Kluwer international series in engineering and computer science, 2002.

[MR09] D. Micciancio and O. Regev, *Lattice-based cryptography*, Post-Quantum Cryptography (Berlin, Heidelberg) (D. J. Bernstein, J. Buchmann, and E. Dahmen, eds.), Springer Berlin Heidelberg, 2009, pp. 147–191.

[MRR06] C. Moore, D. Rockmore, and A. Russell, *Generic quantum Fourier transforms*, ACM Transactions Algorithms **2** (2006), no. 4, 707–723.

[MVO96] A- J. Menezes, S. A. Vanstone, and P.C. Van Oorschot, *Handbook of applied cryptography*, 1st ed., CRC Press, Inc., USA, 1996.

[NC00] M. A. Nielsen and I. L. Chuang, Quantum Computation and Quantum Information, first ed., Cambridge, UK, 2000.

[Sho97] P.W. Shor, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM Journal on Computing **26** (1997), no. 5, 1484.