On the (In)Security of Cryptography in the Quantum Era

Della Giustina Lorenzo
dellagiustina.lorenzo@spes.uniud.it
158854

October 30, 2025

Abstract

The advent of quantum computing poses significant challenges to modern cryptographic systems. This thesis examines two fundamental quantum algorithms - Grover's unstructured search algorithm and Shor's factorization algorithm - and their implications for cryptographic security. Grover's algorithm provides a quadratic speedup over classical brute-force search, reducing the effective security of symmetric key ciphers and hash functions by approximately half. We analyze its application to AES-128 encryption and hash function collision attacks, demonstrating how quantum circuit implementations of cryptographic primitives enable practical attacks. Shor's algorithm offers an exponential speedup for *integer factorization* and *discrete logarithm* problems, posing a threat to widely-used public-key cryptosystems such as RSA.

Keywords Quantum Computing · Cryptography · Grover · Shor · AES-128 · Collision Attacks · RSA

Contents

1 Introduction to Quantum Computing

Before delving into the specifics of Grover's and Shor's algorithms, it is essential to establish a foundational understanding of quantum computing concepts that underpin these algorithms.

This section provides a concise introduction to the fundamental principles of quantum computing necessary for comprehending the subsequent discussions.

1.1 Qubits

A *qubit* (quantum bit) is the basic unit of quantum information, analogous to a classical bit. Unlike a classical bit that can be in one of two states (0 or 1), a qubit can exist in a *superposition* of both states simultaneously. Mathematically, the state of a qubit can be represented as a linear combination of the basis states $|0\rangle$ and $|1\rangle$:

$$|\psi\rangle = \alpha|0\rangle + \beta\ |1\rangle, \tag{1}$$

where $|\psi\rangle$ is the state vector of the qubit, and α and β are complex numbers (called *amplitudes*) satisfying the normalization condition:

$$|\alpha|^2 + |\beta|^2 = 1. \tag{2}$$

The notation $|\psi\rangle$ (ket notation) is part of the Dirac notation commonly used in quantum computing:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{3}$$

It is important to note that the state of a qubit cannot be directly observed without collapsing it to one of the basis states upon measurement. In particular, measuring a qubit in the state $|\psi\rangle$ yields the outcome 0 with probability $|\alpha|^2$ and 1 with probability $|\beta|^2$. However, qubits can be manipulated using quantum gates whose outcomes depend on the amplitudes α and β.

Systems of multiple qubits are represented using the tensor product of individual qubit states. For example, a two-qubit system can be represented as:

$$|\psi\rangle = \alpha_{00}\ |00\rangle + \alpha_{01}\ |01\rangle + \alpha_{10}\ |10\rangle + \alpha_{11}\ |11\rangle. \tag{4}$$

1.2 Quantum Gates and Circuits

Quantum gates are the building blocks of quantum circuits, analogous to classical logic gates. They are represented by *unitary* matrices[1] (in order to preserve the normalization condition Equation 2) that operate on the state vectors of qubits.

[1] In linear algebra a matrix U is unitary if its inverse U^{-1} is equal to its conjugate transpose $U^\dagger$: $U^\dagger U = UU^\dagger = I$.

For example, a single-qubit gate is represented by a 2x2 unitary matrix that transforms the coefficients of the qubit's state vector:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} a * \alpha + b * \beta \\ c * \alpha + d * \beta \end{pmatrix}. \tag{5}$$

It is important to note that quantum gates are reversible.

Some common qubit gates used in the algorithms discussed in this thesis include:

- *Hadamard Gate (H)*: acts on a single qubit and has the following matrix representation:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{6}$$

  The Hadamard gate on the basis states produces superpositions: $H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. This means that applying the Hadamard gate to a qubit in a basis state results in a state that has equal probability to collapse to either $|0\rangle$ or $|1\rangle$. In general this gate is used to create superpositions from basis states and then perform operations on all the states simultaneously.
- *Pauli-X Gate (X)*: also known as the quantum NOT gate, it flips the state of a single qubit. Its matrix representation is:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}. \tag{7}$$

  The Pauli-X gate acts on the basis states as follows: $X|0\rangle = |1\rangle, X|1\rangle = |0\rangle$.
- *Controlled NOT Gate (CNOT)*: a two-qubit gate that flips the state of the second qubit (target) if the first qubit (control) is in the state $|1\rangle$. Its matrix representation is:

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{8}$$

2 Grover's Unstructured Search Algorithm

Lov K. Grover proposed in 1996 a quantum algorithm that provides a quadratic speedup for unstructured search problems (Grover 1996; Wikipedia contributors 2025b; IBM, n.d.).

With a classical algorithm, searching an unsorted database of N items requires on average $\frac{N}{2}$ queries ($O(N)$ time complexity). Grover's algorithm, however, can find the desired item in $O\left(\sqrt{N}\right)$ time.

2.1 Problem statement

Let us define the problem formally. Given:

- a set of $N = 2^n$ possible states labeled $S_1, S_2, ..., S_N$;
- a condition function $C(S_i)$ evaluable in $O(1)$ time with domain $\{S_1, ..., S_N\}$ and codomain $\{0, 1\}$;
- a state S_ω such that $C(S_\omega) = 1$ and $\forall i \neq \omega(C(S_i) = 0)$;

the goal is to find S_ω.

2.2 Algorithm

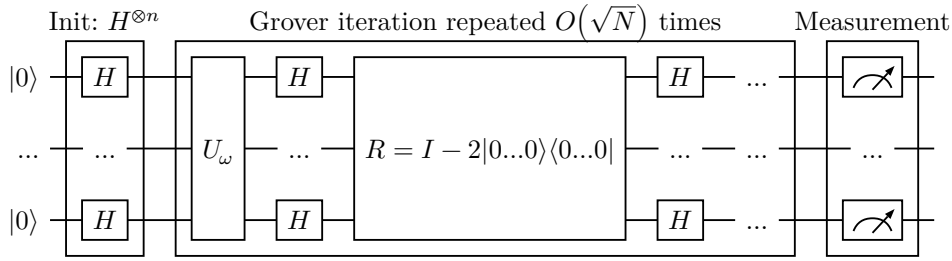

Figure 1: Overview of the Grover's algorithm circuit.

Initialization The algorithm starts with an n-qubit register initialized to the state $|0...0\rangle$. The Hadamard gate is applied to each qubit to create the uniform superposition.

$$H^{\otimes n} \ |0...0\rangle = \frac{1}{\sqrt{2^n}} * \sum_{y \in \{0,1\}^n} |y\rangle \tag{9}$$

This step requires $O(n) = O(\log(N))$ operations.

At this point if we measure $|x\rangle$, its superposition would collapse to any one of the basis states with the same probability $\frac{1}{N} = \frac{1}{2^n}$. The chances of guessing the right value is $\frac{1}{2^n}$: on average the number of tries needed to guess the correct item is the same as classical search, i.e. $\frac{N}{2} = 2^{n-1}$.

Grover iteration The following step is repeated $O\left(\sqrt{N}\right)$ times:

1. **oracle**: *phase inversion.* Rotate the amplitude of the searched state $|S_\omega\rangle$ by π radians (i.e. invert its sign). This is done using an **oracle** U_ω that implements the condition function C. The oracle acts on the orthonormal basis as:
$$U_\omega \ |S_i\rangle = (-1)^{C(S_i)} \ |S_i\rangle \tag{10}$$
This means:
   - if $C(S_i) = 1$ (i.e. S_i is the searched ω state), the amplitude is multiplied by -1;
   - if $C(S_i) = 0$ (non-ω state), the amplitude is unchanged;

2. **diffusion operator**: *inversion about the average.* After the oracle marks the searched state S_ω by flipping its sign, a second unitary operation D is applied to "amplify" its amplitude. D is defined as:
$$D = H^{\otimes n} R H^{\otimes n} \tag{11}$$
where $R = I - 2|0...0\rangle\langle 0...0|$ (ket-bra notation for a matrix with 1 in the top right element, -1 in the remaining elements of the top-right to bottom-left diagonal, and 0 everywhere else) is a simple reflection.

   The effect of D is to reflect all amplitudes about their average value: each amplitude α_i is mapped to $2 \cdot \bar{\alpha} - \alpha_i$, where $\bar{\alpha}$ is the average amplitude. This increases the amplitude of the searched state by $O\left(\frac{1}{\sqrt{N}}\right)$ while slightly decreasing the others.

Geometrically, each Grover iteration (oracle + diffusion) corresponds to a rotation of the state vector towards the searched state. This means that after $k \approx \pi \frac{\sqrt{N}}{4}$ iterations (which is the optimal number of iterations which yields ≈ 1 probability of success for big enough N), the probability of measuring the searched state worsens.

Measurement After $O\left(\sqrt{N}\right)$ Grover iterations, a measurement is performed on the register. The result is the searched state S_ω with high probability. It can be checked classically if the result is correct by evaluating $C(S_\omega)$.

2.3 Generalization to multiple solutions

Grover's algorithm can be naturally extended to the case where there are M solutions (instead of just one) among the N total states. When $M > 1$, the algorithm converges faster, requiring approximately $\frac{\pi}{4}\sqrt{\frac{N}{M}}$ iterations to find one of the solutions with high probability. This is a significant speedup compared to the single-solution case when M is not too small.

The oracle remains the same: it marks all M solutions by flipping their phases. The key difference is that after each oracle application, the average amplitude is computed over all N states, but now M of them have negative phase. This

results in a larger rotation angle per iteration which explains the reduced number of iterations needed.

Note that since the probability of measuring any one of the M solutions worsens after $O\left(\sqrt{\frac{N}{M}}\right)$ iterations, for applying the standard Grover's algorithm we need to know the exact number of solutions M. However several ways to circumvent this limitation have been developed. A simple approach is just to run Grover's algorithm multiple times with increasing estimates of M until a solution is found.

This generalization is particularly relevant in cryptographic applications where the goal is often to find *any* valid solution (e.g., any hash collision or any valid key) rather than a specific one.

2.4 Brute force (Talman 2025)

Grover's algorithm has the potential of being used to speed up any search problem. In particular can solve the worst case of NP-complete problems faster than any known classical algorithm (by providing a quadratic speedup over brute-force search).

In the cryptography setting, many algorithms rely on the hardness of certain problems to guarantee security. In general those problems are functions that are believed to be hard to invert. Grover's algorithm provides a quadratic speedup for the naive brute-force search for pre-images of these functions. While asymmetric cryptography is mostly threatened by Shor's algorithm (see Section 3), Grover's algorithm poses mainly a threat to symmetric key ciphers and hash functions.

In general, if a symmetric key cipher or hash function has a classical security level of n bits (i.e. it requires 2^n operations to break it) in the classical setting, Grover's algorithm can reduce the security level to about $\frac{n}{2}$ bits.

2.4.1 AES-128 (Almazrooie et al. 2018)

The difficulty of implementing Grover's algorithm in practice lies mainly in the construction of efficient quantum oracles for the cryptographic primitives being attacked.

As a case study, let's consider the Advanced Encryption Standard (AES) with a 128-bit key.

AES-128 is a symmetric block cipher that operates on 128-bit blocks (called states and represented as 4×4 matrices) using a 128-bit key. Encryption starts with the `AddRoundKey` step, where the plaintext is XORed with the key to initialize the state. Then the algorithm proceeds through 10 rounds. Each round applies four steps to the 16-byte state:

- `SubBytes`: a byte-wise non-linear substitution (basically bytes in the state are replaced with other values according to a transformation called S-box);

- `ShiftRows`: a fixed row-wise byte permutation (the i-th row is shifted left by i bytes);
- `MixColumns`: a linear mixing of each 4-byte column;
- `AddRoundKey`: a bitwise XOR with a round-dependent key.

The final round omits `MixColumns`.

In order to exploit Grover's algorithm to perform a brute-force key search against AES-128, we need to construct a quantum circuit that can efficiently evaluate the AES encryption function, i.e. implement an oracle that marks the correct key given a known plaintext-ciphertext pair. The goal is to minimize the number of qubits and gates used in the circuit, as these are the main constraints in current quantum hardware.

`AddRoundKey` The naive way to implement the `AddRoundKey` step is to store the plaintext and the key as qubits (a total of $2 * 128$ qubits) and use CNOT gates to XOR them. This approach would let us define a plaintext-agnostic circuit that works for any given plaintext. But in practice it would be very expensive in terms of qubits (which is the main bottleneck in current quantum hardware).

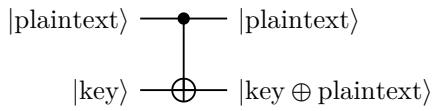

Figure 2: Bitwise XOR of plaintext and key using CNOT gates.

A better solution (in terms of number of qubits) is to hardcode the plaintext in the circuit using Pauli-X gates to flip the bits of the key where the plaintext has a 1. In this way we can achieve the same result by using only 128 qubits.

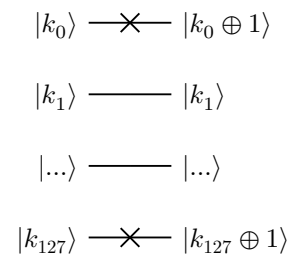

Figure 3: Example of hardcoding the plaintext 10...1 using Pauli-X gates.

`SubBytes` The SubBytes step replaces each byte with its AES S-box value. This substitution is non-linear and it is what gives AES its security. The AES S-box is constructed by a two-step process:

1. finding the multiplicative inverse in the finite field GF(2^8), i.e. finding b such that $a \times b = 1$ modulo the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$;
2. apply an affine linear transformation over GF(2).

Overall, the S-box transformation can be expressed as:

$$b = M \cdot a^{-1} + c \tag{12}$$

where:

- a is the input byte;
- b is the output byte;
- a^{-1} is the multiplicative inverse of a in GF(2^8), with the convention that $0^{-1} = 0$;
- M is a fixed 8x8 binary matrix with 1s and 0s;
- c is a fixed 8-bit constant vector.

The goal is to implement a quantum circuit that performs the `SubBytes` transformation using the least number of qubits and gates:

$$|\text{input byte}\rangle\ |0...0\rangle \rightarrow |\text{input byte}\rangle\ |\text{S-box(input byte)}\rangle \tag{13}$$

Classically this is done with a small lookup table to enable faster computation. But since the main goal is to minimize the number of qubits used, it is preferable to implement the S-box dynamically. The approach used in (Almazrooie et al. 2018) is:

- compute the multiplicative inverse in GF(2^8) using a reversible arithmetic network (linear XOR layers plus Toffoli-based multiplications);
- apply the final affine linear transformation with only CNOT and X gates.

First of all, regarding the inverse computation in GF(2^8), in (Almazrooie et al. 2018) it is found that *Itoh-Tsujii algorithm* uses the least number of ancilla qubits for the computation of the multiplicative inverse in GF(2^8).

In particular Itoh-Tsujii algorithm (Wikipedia contributors 2025c) states that in GF(2^8):

$$a^{-1} = 2^{2^8-2}a^{254} = t_2 * t_4 * \left(\left(\left((t_5 * t_3)^2\right)^2\right)^2\right)^2, \tag{14}$$

where:

- $t_1 = a^2$ (one squaring);
- $t_2 = t_1 * a (= a^3)$ (one multiplication);
- $t_3 = t_2^2 (= a^6)$ (one squaring);
- $t_4 = t_3^2 (= a^{12})$ (one squaring);

Note that in total the formula requires only 4 multiplications and 7 squarings.

(Almazrooie et al. 2018) provides an implementation of multiplication and squaring in GF(2^8) of quantum bytes (8 qubits) using Toffoli and CNOT gates. Squaring is a linear operation in GF(2^8) and can be implemented with only CNOT gates and without ancilla qubits; while multiplication is implemented using a sequence of Toffoli and CNOT gates and requires an ancilla quantum byte to store

the result. In total the inverse computation requires 48 ancilla qubits (that are later uncomputed to be reused).

Lastly, the affine linear transformation is applied to the result of the inverse computation. It is trivially implemented with CNOT and X gates as in (Almazrooie et al. 2018).

Overall the `SubBytes` implementation (including the affine transformation) in (Almazrooie et al. 2018) has depth 676.

`ShiftRows` The `ShiftRows` step is a fixed row-wise permutation of the bytes in the state. As in the classical case, it can be implemented "for free" in the quantum circuit by just wiring the qubits accordingly, without any gates or ancilla qubits.

`MixColumns` The `MixColumns` step is a linear transformation that operates on each of the 4 columns of the state independently.

Each column a is treated as a four-term polynomial $a(x)$ over $\mathrm{GF}(2^8)$:

$$a(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3, \tag{15}$$

where a_i are the bytes in the column. This polynomial is then multiplied modulo $x^4 + 1$ with the fixed polynomial $b(x) = 03_{16}x^3 + 01_{16}x^2 + 01_{16}x^2 + 02_{16}$.

The result is:

$$c(x) = b(x) * a(x) \bmod (x^4 + 1) \tag{16}$$

This is equivalent to the matrix multiplication:

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 02_{16} & 03_{16} & 01_{16} & 01_{16} \\ 01_{16} & 02_{16} & 03_{16} & 01_{16} \\ 01_{16} & 01_{16} & 02_{16} & 03_{16} \\ 03_{16} & 01_{16} & 01_{16} & 02_{16} \end{pmatrix} * \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} \tag{17}$$

The multiplication by 01_{16} is just the identity, while the multiplications by 02_{16} and 03_{16} are implemented as the multiplication in the inverse computation step performed in `SubBytes`. The additions are simply XORs implemented with CNOT gates.

Overall the implementation of `MixColumns` requires no ancilla qubits and has a circuit depth of 200.

Key schedule `AddRoundKey` requires round keys derived from the original key using the AES key schedule. In total 11 round keys are needed for AES-128 (one for the initial `AddRoundKey` and one for each of the 10 rounds).

For each round i the key is represented as 4 words $K_{i,0}, K_{i,1}, K_{i,2}, K_{i,3}$. The first 4 words are directly taken from the original key. The words for round $i \neq 0$ are generated with the following rule:

$$K_{i,0} = \text{RotWords}\big(\text{SubBytes}\big(K_{i-1,3}\big)\big) \oplus \text{Rcon}_i \oplus K_{i-1,0},$$
$$K_{i,j} = K_{i,j-1} \oplus K_{i-1,j} \forall j \in \{1,2,3\}, \tag{18}$$

where `RotWords` is a rotation implemented similarly to `ShiftRows` (Section 2.4.1.3), `SubBytes` is the S-box transformation, and `Rcon_i` is a fixed round constant implemented with Pauli-X gates similarly to the plaintext hardcoding in `AddRoundKey` (Section 2.4.1.1).

The key schedule cost is round-dependent. In (Almazrooie et al. 2018) an in-depth analysis of the key schedule implementation is provided.

Full AES-128 circuit Combining all the steps described above, the full 10 rounds AES-128 encryption circuit can be constructed. In (Almazrooie et al. 2018) it is reported that the overall AES-128 circuit requires 928 qubits and 48 ancilla qubits.

Oracle implementation and key uniqueness The circuit described above implements the AES encryption function as a quantum circuit. It can be used to build the oracle needed for Grover's algorithm such that:

$$U_\omega \, |\text{key}\rangle = (-1)^{C(\text{key})} \, |\text{key}\rangle, \tag{19}$$

where $C(\text{key}) = 1$ if and only if AES-128$(\text{plaintext}, \text{key}) = \text{ciphertext}$ for the known plaintext-ciphertext pair.

In order to do so, after computing the AES encryption of the known plaintext with the given key, the result is compared with the known ciphertext. If they match, a phase flip is applied to the state.

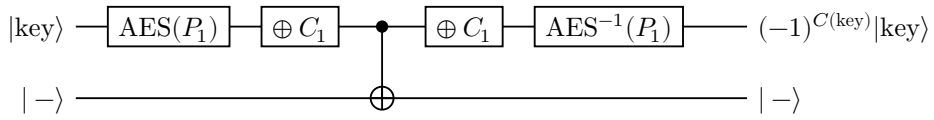

Figure 4: Exploiting the phase kickback effect to implement the AES oracle for Grover's algorithm. The candidate key is tested against a single plaintext-ciphertext pair. If the encryption matches the ciphertext, the oracle qubit $|-\rangle$ is flipped, inducing a phase flip on the key register. AES(P_1) represents the AES encryption circuit described above, AES$^{-1}(P_1)$ its inverse, and $\oplus C_1$ the comparison with the known ciphertext. The circuit shown uses an extra qubit to implement the phase kickback effect.

However, a single plaintext-ciphertext pair may not guarantee key uniqueness: multiple keys might (by chance) produce the same ciphertext for a given plaintext. To ensure that Grover's algorithm finds exactly one solution (M=1), the oracle should check multiple plaintext-ciphertext pairs. The paper (Almazrooie et al. 2018) proposes using three pairs to guarantee with higher probability that only the correct key is marked by the oracle. For each additional pair, the AES encryption circuit is repeated with the same key register, and the result is compared with the

corresponding ciphertext. The phase if flipped only for keys that match all pairs. In (Almazrooie et al. 2018) it is proposed a circuit that requires an extra ancilla qubit for each additional pair.

Overall AES oracle cost Overall, the AES-128 oracle for Grover's algorithm constructed as described above requires $928 + 48 + 1 + 2 = 979$ qubits.

It is important to note that as of today, quantum computers with this many **logical** qubits[2] do not exist. Therefore, while Grover's algorithm poses a theoretical threat to AES-128, practical attacks remain infeasible with current technology. However, as quantum hardware continues to improve, this situation may change in the future, making the transition to larger key sizes (e.g., AES-256) or post-quantum cryptographic schemes increasingly important.

2.4.2 Hash functions (Wang et al. 2020; Ramos-Calderer et al. 2020)

Another important application of Grover's Brute Force search is in the context of hash functions.

Hashes are functions that take an input and return a fixed-size string of bytes. Hash functions are widely used in cryptography for a variety of purposes, including data integrity verification and digital signatures. In general it is desirable that a hash function has the following properties:

- **pre-image resistance**: given a hash value h, it should be computationally infeasible to find any input x such that $\text{hash}(x) = h$;
- **second pre-image resistance**: given an input x_1, it should be computationally infeasible to find another input x_2 such that $\text{hash}(x_1) = \text{hash}(x_2)$;
- **collision resistance**: it should be computationally infeasible to find any two different inputs x_1 and x_2 such that $\text{hash}(x_1) = \text{hash}(x_2)$.

However, even for well-designed hash functions, pre-images, second pre-images, and collisions can be found using a brute-force search of the input space.

Collision attacks (Bellare and Rogaway 2005; Oorschot and Wiener 1994; Wikipedia contributors 2025d) Collision attacks try to find two different inputs that produce the same hash output. If the hash behaves like a random function with an n-bit output, there are $N = 2^n$ possible outputs.

Classical approach The standard collision search is based on the **birthday paradox**: when sampling inputs at random and looking at their hashes, collisions appear much sooner than one might naively expect (Bellare and Rogaway 2005).

[2] A logical qubit is an abstraction over physical qubit implementation that behaves as expected. Due to the issues faced by the current hardware implementations of qubits, many physical qubits are needed to implement a single "entity" that behaves as a single qubit.

Modeling the hash outputs as independent uniform draws from a set of size N, the probability that k independent samples have distinct outputs is:

$$P(\text{all } k \text{ distinct}) = \prod_{i=0}^{k-1}\left(1 - \frac{i}{N}\right) \approx e^{-\frac{k(k-1)}{2N}} \approx e^{-\frac{k^2}{2N}}, \tag{20}$$

for $k < \sqrt{2N}$. Therefore the probability of seeing at least one collision among k samples is:

$$P(\text{collision in } k \text{ samples}) = 1 - e^{-\frac{k^2}{2N}}. \tag{21}$$

Solving for k gives the number of samples required to achieve collision with probability p: $k \approx \sqrt{2N \ln\left(\frac{1}{1-p}\right)}$. In particular, for $p = \frac{1}{2}$ we get $k \approx \sqrt{2\ln(2)}\sqrt{N} \approx 1.177\sqrt{N}$.

It can be calculated that the expected number of samples until the first collision is approximately $\sqrt{\pi\frac{N}{2}} \approx 1.253\sqrt{N}$.

Translating back to an n-bit hash ($N = 2^n$) yields the well-known birthday attack time complexity for collision finding: $O\left(2^{\frac{n}{2}}\right) = O\left(\sqrt{N}\right)$.

A straightforward implementation of the birthday attack stores the seen hash values in a table and checks for duplicates, so it also requires about $O\left(2^{\frac{n}{2}}\right)$ memory.

Pollard's rho (and other cycle-finding methods) achieve the same asymptotic time complexity $O\left(2^{\frac{n}{2}}\right)$ for collision search but with dramatically lower memory. Pollard's rho iterates through a pseudorandom walk from a sequence $x_{i+1} = h(x_i)$[3] with starting value x_0. Because the codomain is finite, the sequence must eventually cycle. The repetition is detected using cycle-detection techniques (Floyd's "tortoise and hare" or Brent's variant). Because it never needs to store all previous values, the memory requirement is $O(1)$.

Practically, Pollard's rho provides two important advantages over the naive birthday table approach:

- much lower memory usage: constant instead of $O\left(2^{\frac{n}{2}}\right)$;
- efficiently parallelizable: Pollard's rho can be efficiently scaled across many workers.

The parallelization can be achieved initializing each worker to perform independent pseudorandom walks $x_{i+1} = f(x_i)$ starting from a random seed. As soon as two walks touch each other, the two walks will merge from that point on. Thus a collision is found. The expected number of steps taken by each processor is $\frac{\sqrt{\pi\frac{N}{2}}}{m}$ (where m is the number of processors).

[3]This is possible only when the domain and codomain of the hash function are the same. If not, the codomain can be mapped back to the domain with a suitable function.

Note that Pollard's rho does not change the asymptotic time complexity: both methods require about $O\big(2^{\frac{n}{2}}\big)$ work, but it is usually the preferred practical technique for collision search when one wants to minimize memory usage or when one wants an efficient parallel implementation.

Quantum approach If the attacker has access to a quantum computer with n qubits, the classical Grover search algorithm can be used to find a collision with a chosen element $x_0 \in X$ (with X being the domain of the hash function) in $O\big(2^{\frac{n}{2}}\big)$ function evaluations by defining an oracle that marks inputs x_1 such that $\text{hash}(x_1) = \text{hash}(x_0)$. However, this approach does not improve over the classical birthday attack. In fact we are solving a more specific (and difficult) problem than the general collision finding problem: we are trying to find a collision with a specific element, which is harder than finding any collision.

Brassard, Høyer and Tapp showed a better quantum/classical hybrid strategy that reduces collision finding to $O\Big(\sqrt[3]{\frac{N}{r}}\Big)$ time and memory (Brassard, HØyer, and Tapp 1998) when the hash function is *r-to-one*[4]. The idea is to preprocess a subset of possible hashes classically, then use Grover to search for collisions against that subset (instead of against a specific element as in the naive Grover approach):

- Let $N = 2^n$ be the number of possible hash outputs Y ($|Y| = N$). Choose a subset $K \subseteq Y$ of size $k = |K|$. Compute the subset's hashes and store the $(\text{input}, \text{hash})$ pairs in a table (sorted or hashed). If a collision is found during this step, the algorithm can stop early. Otherwise, we have a table of k distinct hashes. Note that if the algorithm does not stop early, and the hash function is r-to-one, then it is guaranteed that there will be collisions between the set K and the rest of the domain. This preprocessing costs $O(k)$ time and $O(k)$ memory.
- Now use Grover's algorithm to search over the remaining inputs for one whose hash matches an entry in the table. Each Grover oracle evaluation computes the hash of a candidate input and checks membership in the precomputed table (a single lookup).

The probability that a random candidate collides with one of the k stored hashes is about $\frac{k}{N}$. Grover thus needs $O\Big(\sqrt{\frac{N}{k}}\Big)$ oracle calls to find such a candidate. Substituting $k = N^{\frac{1}{3}}$ gives:

$$\text{Number of oracle calls} : \sqrt{\frac{N}{k}} = \sqrt{\frac{N}{N^{\frac{1}{3}}}} = N^{\frac{1}{3}} = 2^{\frac{n}{3}}. \tag{22}$$

Adding the preprocessing cost $k = 2^{\frac{n}{3}}$ yields a total time (and memory) complexity of $O\big(2^{\frac{n}{3}}\big)$.

[4]A function is said to be *r-to-one* if every element in its image has exactly r distinct preimages.

In summary, the approach above combines classical preprocessing with Grover search to achieve a collision-finding algorithm with time and memory complexity $O\left(2^{\frac{n}{3}}\right)$, which is significantly better than the classical birthday attack's $O\left(2^{\frac{n}{2}}\right)$ complexity.

Pre-image attacks Pre-image attacks try to break the unidirectionality of hash functions by finding an input that hashes to a specific output. Grover's algorithm can be applied to this problem, providing a quadratic speedup over classical brute-force search. The attack is conceptually similar to the key search against symmetric ciphers: given a target hash value h, the oracle marks any input x such that $\text{hash}(x) = h$. For an n-bit hash function, the classical brute-force search requires $O(2^n)$ hash evaluations, while Grover's algorithm reduces this to $O\left(2^{\frac{n}{2}}\right)$ evaluations. Thus, the security level against pre-image attacks is also halved in the quantum setting.

2.5 Impact on Cryptographic Security

Grover's algorithm provides a quadratic speedup for unstructured search problems, which has significant implications for symmetric cryptography. As demonstrated in this chapter, the algorithm can reduce the effective security level of symmetric key ciphers and hash functions by approximately half: a cipher with n-bit keys requires $O\left(2^{\frac{n}{2}}\right)$ quantum operations to break, compared to $O(2^n)$ classical operations.

2.5.1 Practical Limitations

However, several practical challenges limit the immediate threat posed by Grover's algorithm:

Oracle implementation complexity. The construction of efficient quantum oracles for cryptographic primitives is non-trivial. As shown in the AES-128 case study, implementing the encryption function as a quantum circuit requires more logical qubits than are available on current quantum hardware. Moreover each cryptographic primitive requires a custom oracle design.

Current hardware constraints. As of 2025, quantum computers with hundreds of logical qubits do not exist. The gap between theoretical algorithms and practical implementations remains large, and it may take decades before quantum computers capable of breaking real-world cryptographic systems become available.

Limited applicability. Grover's algorithm only provides a speedup for *unstructured* search problems. When additional structure is known about the search space, or when the problem can be solved more efficiently through other means, Grover's algorithm offers no advantage. For example, searching in sorted data structures or problems with known patterns do not benefit from Grover's quadratic speedup.

Parallelization challenges. Unlike classical brute-force search, which scales linearly with the number of processors, Grover's algorithm does not parallelize efficiently. The quadratic speedup is inherent to the algorithm's structure, and running multiple quantum computers in parallel does not provide the same benefits as classical parallel search.

2.5.2 Defensive Measures

The potential threat from Grover's algorithm can be mitigated through straightforward countermeasures:

Key size increase. Doubling the key size restores the original security level against quantum attacks. For instance, transitioning from AES-128 to AES-256 ensures 128-bit security even against quantum adversaries, as Grover's algorithm would require $O(2^{128})$ operations.

Hash output length increase. Similarly, using hash functions with larger output sizes (e.g., SHA-512 instead of SHA-256) maintains collision resistance and pre-image resistance against quantum attacks.

2.5.3 Conclusion

While Grover's algorithm represents a theoretical threat to symmetric cryptography, its practical impact is limited by current technological constraints and the availability of simple countermeasures. The algorithm's quadratic speedup is significant from a complexity theory perspective, but doubling key sizes provides an effective defense. Nevertheless, the existence of Grover's algorithm underscores the need for ongoing research in post-quantum cryptography and careful consideration of security margins in cryptographic system design.

3 Shor algorithm (Nielsen and Chuang 2010; Wikipedia contributors 2025a)

In 1994, the American mathematician Peter Shor developed a quantum algorithm for integer factorization that runs in polynomial time with respect to the number of bits of the input integer, specifically in time $O((\log N)^3)$, where N is the integer to be factored. This was a groundbreaking discovery because it provides an exponential speedup over the best-known classical factoring algorithms. This advancement could potentially compromise the security of widely used public-key cryptographic systems, such as RSA, which rely on the difficulty of factoring large integers.

The **factoring problem** is the task of decomposing a composite integer N into its prime factors. For example, the number 15 can be factored into 3 and 5.

Shor's algorithm consists of two main parts:
1. the **reduction** of the **factoring problem to** the **order-finding problem**;
2. a **quantum algorithm to solve the order-finding problem**.

3.1 Classical reduction

The factoring problem turns out to be reducible to the order-finding problem, which can be solved efficiently using quantum algorithms.

In particular, below it is shown an algorithm to decompose a composite integer N in two non-trivial integers that may or may not be prime. This algorithm can be used to factor N by recursively applying it until all prime factors are found.

The key idea is the following:
- we pick a random integer a such that $1 < a < N$. If $\gcd(a, N) \neq 1$, then we have found a non-trivial factor of N and we are done;
- otherwise we use a quantum algorithm to find the **order** r of a modulo N, which is the smallest positive integer such that $a^r \equiv 1 \bmod N$. This means that for some integer k we have that $kN = a^r - 1 = \big(a^{\frac{r}{2}} - 1\big)\big(a^{\frac{r}{2}} + 1\big)$. $a^{\frac{r}{2}} - 1$ does not divide N (otherwise $\frac{r}{2}$ would be the order of a). So at least one of the factors of N divides $a^{\frac{r}{2}} + 1$. To find it, we compute $\gcd\big(N, a^{\frac{r}{2}} + 1\big)$. If this is a non-trivial factor of N, we are done; otherwise we repeat the process from the beginning.

Algorithm 1: Factoring algorithm

```
procedure FACTORING ALGORITHM(N)
    while true do
        ▷ Pick a random integer 1 < a < N (in O(log N) time).
        a ← random(1, N)
```

```
5:
6:      ▷ Compute the greatest common divisor of a and N (in O((log N)^2)
        time with the Euclidean algorithm).
7:      K ← gcd(a, N)
8:
9:      if K ≠ 1 then
10:         ▷ K is a non-trivial factor of N (the other factor is N/K).
11:         return K, N/K
12:     else
13:         ▷ Otherwise proceed to the quantum order-finding step.
14:         r ← quantum_order(a, N)
15:
16:         if is_odd(r) then
17:             ▷ If r is odd, then retry from the beginning.
18:             CONTINUE
19:         else
20:             ▷ Otherwise compute the greatest common divisor of N and a^(r/2) +
                1 (in O((log N)^3) time using the fast exponentiation algorithm
                modulo N and the Euclidean algorithm).
21:             g ← gcd(N, a^(r/2) + 1)
22:             if g ≠ N then
23:                 ▷ g is non-trivial: the other factor is N/g.
24:                 return g, N/g
25:             end
26:             ▷ Otherwise retry from the beginning.
27:         end
28:     end
29: end
30: end
```

Each iteration of the `while` loop takes $O((\log N)^3)$ operations in the classical steps, plus one call to the quantum order-finding algorithm.

It has been shown that this will be likely to succeed after a few runs, so overall the time complexity of the classical reduction is $O((\log N)^3)$.

3.2 Order-finding quantum algorithm

The classical algorithm shown above relies on the ability to solve the **order-finding problem** efficiently. The order-finding problem is defined as follows: given two coprime integers N and a, such that $1 < a < N$, find the smallest r such that

$$a^r \equiv 1 \bmod N. \tag{23}$$

This algorithm relies heavily on the *phase estimation* algorithm, which in turn relies on the *Quantum Fourier Transform* (*QFT*). Below these two building blocks are explained.

3.2.1 Quantum Fourier Transform

The **quantum Fourier transform** (QFT) is the quantum analogue of the **discrete Fourier transform** (DFT) and plays a central role in many quantum algorithms, including Shor's algorithm.

The DFT on a complex vector $(x_0, x_1, ..., x_{N-1})$ outputs a new complex vector $(y_0, y_1, ..., y_{N-1})$ defined by:

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{2\pi i jk/N}. \tag{24}$$

The QFT is the same transformation, but applied to the amplitudes of a quantum state in a superposition of basis states. Its action on a quantum state is:

$$\sum_{j=0}^{N-1} x_j \ |j\rangle \rightarrow \sum_{k=0}^{N-1} y_k \ |k\rangle, \tag{25}$$

where y_k is defined as in the DFT above.

Let $n = \log_2(N)$ and $j = j_1 * 2^{n-1} + j_2 * 2^{n-2} + ... + j_n * 2^0$ be the binary representation of j. With this notation, and by some algebraic steps, the QFT on orthonormal basis states $|0\rangle ... \ |N-1\rangle$ can be expressed as:

$$|j_1 ... j_n\rangle \rightarrow$$
$$\frac{(|0\rangle + e^{2\pi i 0.j_n} \ |1\rangle)(|0\rangle + e^{2\pi i 0.j_{n-1} j_n} \ |1\rangle) ... (|0\rangle + e^{2\pi i 0.j_1 j_2 ... j_n} \ |1\rangle)}{2^{\frac{n}{2}}}, \tag{26}$$

where $0.j_l j_{l+1} ... j_m = \frac{j_l}{2} + \frac{j_{l+1}}{2^2} + ... + \frac{j_m}{2^{m-l+1}}$.

This product representation of the QFT suggests a circuit implementation as in (Nielsen and Chuang 2010).

The number of gates required by the construction in (Nielsen and Chuang 2010) is $\frac{n(n-1)}{2}$. Thus, the time complexity of the QFT is $O(n^2) = O((\log N)^2)$.

3.2.2 Phase estimation

Let's consider an unitary operator U and one of its eigenvectors $|u\rangle$ such that its eigenvalue is $e^{2\pi i \varphi}$:

$$U \ |u\rangle = e^{2\pi i \varphi} \ |u\rangle. \tag{27}$$

The **phase estimation** algorithm aims to estimate the value of φ given access to controlled applications of U and the eigenvector $|u\rangle$.

The quantum phase estimation algorithm uses two quantum registers:
1. the first one with t qubits is initialized to the state $|0\rangle$;
2. the second one is initialized to the eigenvector $|u\rangle$ of the unitary operator U whose eigenvalue phase we want to estimate.

The algorithm proceeds with three main steps:
1. the set up of the first register;

2. the application of the *inverse QFT* on the first register;
3. the measurement of the first register to obtain an estimate of the phase φ.

Registers setup The first step consists of:

1. applying a Hadamard gate to each qubit of the first register, putting it in an equal superposition of all basis states;
2. applying controlled-U^{2^j} operations, where the j-th qubit of the first register controls the application of U^{2^j} on the second register. This step entangles the two registers, encoding the phase information into the amplitudes of the first register.

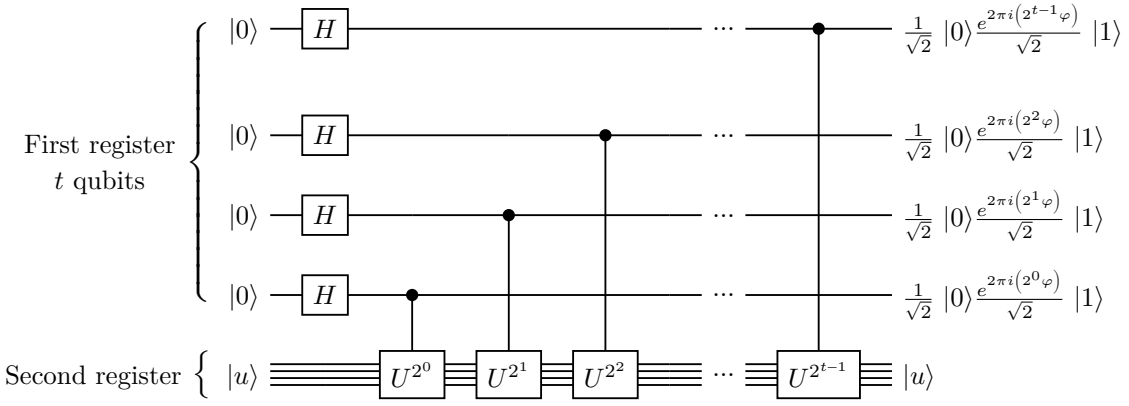

Figure 5: Phase estimation register setup with t control qubits and eigenvector $|u\rangle$.

After this step, the state of the first register is:

$$\frac{1}{2^{\frac{t}{2}}}\Big(|0\rangle + e^{2\pi i 2^{t-1}}\ |1\rangle\Big)\Big(|0\rangle + e^{2\pi i 2^{t-2}}\ |1\rangle\Big)...\Big(|0\rangle + e^{2\pi i 2^{0}}\ |1\rangle\Big)$$
$$= \tag{28}$$
$$\frac{1}{2^{\frac{t}{2}}}(|0\rangle + e^{2\pi i 0.\varphi_t}\ |1\rangle)(|0\rangle + e^{2\pi i 0.\varphi_{t-1}\varphi_t}\ |1\rangle)...(|0\rangle + e^{2\pi i 0.\varphi_1\varphi_2...\varphi_t}\ |1\rangle).$$

***Inverse* QFT** The second step is the application of the *inverse QFT* on the first register. In fact we note that the state of the first register after the setup step in Equation 28 can be seen as the result of applying the QFT Equation 26 to the state

$$|\varphi_1\varphi_2...\varphi_t\rangle. \tag{29}$$

Thus, applying the *inverse QFT* will give a t qubit approximation of the phase φ.

3.2.3 Order finding via phase estimation

Finally the phase estimation algorithm can be used to create the quantum order-finding algorithm used in the classical reduction in Section 3.1. Order finding is the problem of determining the order r of an integer x modulo N, which is the smallest positive integer such that:

$$x^r \equiv 1 (\mathrm{mod}\, N). \tag{30}$$

This means that, starting from $x^0 \equiv 1(\mathrm{mod}\, N)$, repeated multiplication by x generates a sequence of residues modulo N that eventually cycles back to 1 after r multiplications:

$$x^0, x, x^2, x^3, ..., x^{r-1}, x^r \equiv 1, x^{r+1} \equiv x, ... \tag{31}$$

In order to find r let's define the unitary operator U that acts on $|y\rangle$:

$$U\ |y\rangle = |(xy \bmod N)\rangle. \tag{32}$$

Note that applying U r times will cycle through the states:

$$U^r\ |y\rangle = |x^r y \bmod N\rangle = |y\rangle, \tag{33}$$

because $x^r \equiv 1 \bmod N$. So we have that $U^r = I$ (the identity operator).

So, let's take a eigenvector $|u\rangle$ of U:

$$U\ |u\rangle = \lambda\ |u\rangle, \tag{34}$$

for some eigenvalue λ. Applying U r times gives:

$$U^r\ |u\rangle = \lambda^r\ |u\rangle = I\ |u\rangle = |u\rangle. \tag{35}$$

This implies that $\lambda^r = 1$, so the eigenvalues of U are the r-th roots of unity (Wikipedia contributors 2025e):

$$\lambda_k = e^{2\pi i k/r}, k = 0, 1, ..., r-1. \tag{36}$$

The corresponding eigenvectors can be constructed as:

$$|u_k\rangle = \frac{1}{\sqrt{r}} \sum_{j=0}^{r-1} e^{-2\pi i k j/r}\ |x^j \bmod N\rangle. \tag{37}$$

Using the **quantum phase estimation** algorithm, we can estimate the phase $\varphi_k = \frac{k}{r}$ associated with the eigenvalue λ_k of U.

3.3 RSA and quantum threat

The RSA cryptosystem, named after its developers Rivest, Shamir, and Adleman, is one of the most widely used public-key cryptographic systems in the world: it is used in digital signatures, asymmetric encryption, and key exchange protocols.

In RSA, a pair of keys is generated: a public key, which is used for encryption and signing, and a private key, which is used for decryption and signature verification.

Its security relies on the difficulty of factoring large composite integers.

3.3.1 Key generation

The objective of the key generation is to produce a pair (n, e) (public key) and d (private key) such that:

$$(x^e)^d \equiv x(\mathrm{mod}\, n), \tag{38}$$

and finding x given a y such that:

$$x^e \equiv y(\mathrm{mod}\, n) \tag{39}$$

is computationally unfeasible.

More specifically the key generation process involves the following steps:

1. select two large prime numbers p and q (usually 2048 bits long to make factoring unfeasible);
2. compute the modulus $n = p * q$;
3. compute the totient $\lambda = \text{lcm}(p-1)(q-1)$;
4. choose an integer e such that $1 < e < \lambda$ and $\gcd(e, \lambda) = 1$;
5. compute the private exponent d such that $d * e \equiv 1 (\text{mod}\, \lambda)$.

The public key is the pair (n, e), and the private key is d.

3.3.2 Encryption and decryption

To encrypt a message m using the public key (n, e), the sender computes the ciphertext c as:

$$c \equiv m^e (\text{mod}\, n). \tag{40}$$

To decrypt the ciphertext c using the private key d, the receiver computes the plaintext message m as:

$$m \equiv c^d (\text{mod}\, n). \tag{41}$$

3.3.3 Quantum threat to RSA

The security of RSA relies on the practical difficulty of factoring the large composite integer n into its prime factors p and q. Classical algorithms for integer factorization, such as the general number field sieve, have sub-exponential time complexity[5], making it computationally infeasible to factor large integers (e.g., 2048 bits) within a reasonable timeframe. However, Shor's algorithm provides a polynomial-time quantum algorithm for integer factorization. Specifically, it can factor large integers in $O((\log N)^3)$ time, where N is the integer to be factored. This represents an exponential speedup over the best-known classical algorithms, posing a significant threat to the security of RSA and other cryptographic systems based on the difficulty of factoring large integers.

3.4 Period finding

The algorithm proposed for order finding is in fact just an instance of the more general **period finding** problem which the Shor's algorithm is able to solve efficiently.

3.4.1 Problem statement

Given a periodic function $f(x)$, the period finding problem consists of finding the smallest positive integer $0 < r < 2^L$ (with L number of bits to represent the input domain) such that:

[5]Sub-exponential time complexity is a classification of algorithms that are more efficient than exponential time but not as efficient as polynomial time.

$$\forall x(f(x+r) = f(x)). \tag{42}$$

3.4.2 Quantum period finding algorithm

The quantum period finding algorithm relies on the availability of a quantum oracle U that computes the function $f(x)$:

$$U\ |x\rangle\ |y\rangle = |x\rangle\ |y \oplus f(x)\rangle. \tag{43}$$

The algorithm starts with two registers $|0\rangle\ |0\rangle$. The first register has $t = O\big(L + \log\big(\frac{1}{\varepsilon}\big)\big)$ qubits, while the second register has enough qubits to represent the outputs of $f(x)$.

The algorithm then proceeds as follows:

1. apply a Hadamard gate to the first register to create an equal superposition of all possible inputs:

$$|0\rangle\ |0\rangle \rightarrow \frac{1}{\sqrt{2^t}} \sum_{x=0}^{2^t-1} |x\rangle\ |0\rangle; \tag{44}$$

2. apply the oracle U to entangle the two registers:

$$\rightarrow \frac{1}{\sqrt{2^t}} \sum_{x=0}^{2^t-1} |x\rangle\ |f(x)\rangle. \tag{45}$$

3. applying the *inverse QFT* to the first register transforms it to:

$$\mathrm{QFT}^{-1}\ |x\rangle = \frac{1}{\sqrt{2^t}} \sum_{y=0}^{2^t-1} e^{-2\pi i x y/2^t}\ |y\rangle. \tag{46}$$

Thus, the state of the two registers after the operation becomes:

$$\begin{aligned} &\rightarrow \mathrm{QFT}^{-1} \left(\frac{1}{\sqrt{2^t}} \sum_{x=0}^{2^t-1} |x\rangle\ |f(x)\rangle \right) = \\ &= \frac{1}{2^t} \sum_{x=0}^{2^t-1} \sum_{y=0}^{2^t-1} e^{-2\pi i x y/2^t}\ |y\rangle\ |f(x)\rangle = \\ &= \frac{1}{2^t} \sum_{y=0}^{2^t-1} |y\rangle \sum_{x=0}^{2^t-1} e^{-2\pi i x y/2^t}\ |f(x)\rangle. \end{aligned} \tag{47}$$

The periodicity of $f(x)$ implies that $f(x) = f(a + k * r) = f(a)$ with $0 \leq a < r$ and $0 \leq k < \frac{N}{r}$:

$$\begin{aligned} &= \frac{1}{2^t} \sum_{y=0}^{2^t-1} |y\rangle \sum_{a=0}^{r-1} |f(a)\rangle \sum_{k=0}^{\frac{N}{r}-1} e^{-2\pi i (a+kr) y/2^t} = \\ &= \frac{1}{2^t} \sum_{y=0}^{2^t-1} |y\rangle \sum_{a=0}^{r-1} |f(a)\rangle e^{-2\pi i a y/2^t} \sum_{k=0}^{\frac{N}{r}-1} e^{-2\pi i k r y/2^t}. \end{aligned} \tag{48}$$

The geometric series in the last term is:

$$\sum_{k=0}^{\frac{N}{r}-1} e^{-2\pi i k r y/2^t} = \begin{cases} \frac{2^t}{r} \text{ if } y \equiv 0 (\mathrm{mod}\, 2^t/r) \\ 0 \text{ otherwise.} \end{cases} \tag{49}$$

Thus the amplitude of states where first register is $|y\rangle$, with $y \not\equiv 0 (\mathrm{mod}\, 2^t/r)$, is zero;

4. measuring the first register yields a value $y = \widetilde{l/r}$, for some $l \equiv 0 (\mathrm{mod}\, 2^t/r)$ (it is an approximation because 2^t is not necessarily a multiple of r). This value can be used to estimate the period r using classical post-processing techniques, such as the continued fraction expansion.

3.4.3 Discrete logarithm

Let's consider two integers a and b such that:

$$a^s \equiv b (\mathrm{mod}\, N). \tag{50}$$

We define the following periodic function:

$$f(x_1, x_2) = b^{x_1} a^{x_2} \bmod N = a^{s x_1 + x_2} \bmod N. \tag{51}$$

This function has period $(l, -ls)$:

$$\begin{aligned} \forall (x_1, x_2) \big(f(x_1 + l, x_2 - ls) = a^{s(x_1+l)+x_2-ls} \bmod N = \\ = a^{s x_1 + x_2} \bmod N = f(x_1, x_2) \big). \end{aligned} \tag{52}$$

Determining the period of this function by exploiting the quantum period finding algorithm allows to find an s such that $a^s \equiv b (\mathrm{mod}\, N)$, thus solving the discrete logarithm problem.

3.4.4 Diffie-Hellman

The Diffie-Hellman key exchange protocol, developed by Whitfield Diffie and Martin Hellman in 1976, is a method for two parties to securely share a secret key over an insecure communication channel.

Protocol steps The Diffie-Hellman key exchange works as follows:

1. Alice and Bob publicly agree on a large prime number p and a base g (a primitive root modulo p[6]);
2. Alice chooses a secret integer a and computes $A = g^a \bmod p$, which she sends to Bob;
3. Bob chooses a secret integer b and computes $B = g^b \bmod p$, which he sends to Alice;
4. Alice computes the shared secret as $s = B^a \bmod p$;
5. Bob computes the shared secret as $s = A^b \bmod p$.

Both parties arrive at the same shared secret key. To see why, note that $B = g^b \bmod p$ means there exists an integer k such that $B = g^b - kp$. Then:

[6] A primitive root modulo p is an integer g such that any integer a coprime to p is congruent to a power of g modulo p, i.e. for some k, $g^k \equiv a (\mathrm{mod}\, p)$. g needs to be a primitive root modulo p to ensure that the generated keys cover the entire range of possible values modulo p.

$$B^a = \left(g^b - kp\right)^a \equiv g^{ba} (\mathrm{mod}\, p), \tag{53}$$

where the congruence follows from the fact that all terms in the binomial expansion of $\left(g^b - kp\right)^a$ containing p vanish modulo p. Similarly, $A^b \equiv g^{ab} (\mathrm{mod}\, p)$. Since $g^{ab} = g^{ba}$, we have:

$$B^a \equiv g^{ab} \equiv A^b (\mathrm{mod}\, p). \tag{54}$$

This shared secret $s = g^{ab} \bmod p$ can then be used for symmetric encryption.

Quantum threat The Diffie-Hellman protocol shares publicly the values p, g, $A = g^a \bmod p$, and $B = g^b \bmod p$. The security of the protocol relies on the difficulty of computing the shared secret $s = g^{ab} \bmod p$ given only these public values.

An eavesdropper with a quantum computer can intercept the public values $A = g^a \bmod p$ and $B = g^b \bmod p$, and use the quantum discrete logarithm algorithm described above to efficiently solve for a (from $A = g^a \bmod p$) or b (from $B = g^b \bmod p$). Once the attacker knows either a or b, they can compute the shared secret $s = g^{ab} \bmod p$ and decrypt all communications between Alice and Bob.

This vulnerability extends to all cryptographic systems based on the discrete logarithm problem.

Therefore, like RSA, the Diffie-Hellman protocol and its variants require replacement with quantum-resistant alternatives in a post-quantum cryptography era.

Bibliography

[1] A. Talman, "Exploring Grover's Algorithm in Brute Force Attacks," 2025. [Online]. Available: https://oulurepo.oulu.fi/bitstream/handle/10024/58304/nbnfioulu-202509125825.pdf?sequence=1

[2] M. Almazrooie, A. Samsudin, R. Abdullah, and K. N. Mutter, "Quantum reversible circuit of AES-128," *Quantum Information Processing*, vol. 17, no. 5, p. 112, Mar. 2018, doi: 10.1007/s11128-018-1864-3.

[3] P. Wang, S. Tian, Z. Sun, and N. Xie, "Quantum algorithms for hash preimage attacks," *Quantum Engineering*, vol. 2, no. 2, p. e36, 2020, doi: https://doi.org/10.1002/que2.36.

[4] S. Ramos-Calderer, E. Bellini, J. I. Latorre, M. Manzano, and V. Mateu, "Quantum Search for Scaled Hash Function Preimages." [Online]. Available: https://arxiv.org/abs/2009.00621

[5] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[6] Wikipedia contributors, "Shor's algorithm — Wikipedia, The Free Encyclopedia." [Online]. Available: https://en.wikipedia.org/w/index.php?title=Shor%27s_algorithm&oldid=1316087334

[7] L. K. Grover, "A fast quantum mechanical algorithm for database search." [Online]. Available: https://arxiv.org/abs/quant-ph/9605043

[8] Wikipedia contributors, "Grover's algorithm — Wikipedia, The Free Encyclopedia." [Online]. Available: https://en.wikipedia.org/w/index.php?title=Grover%27s_algorithm&oldid=1314138628

[9] IBM, "Qiskit Textbook." [Online]. Available: https://github.com/Qiskit/textbook/tree/main/notebooks/ch-demos

[10] Wikipedia contributors, "Itoh-Tsujii inversion algorithm — Wikipedia, The Free Encyclopedia." [Online]. Available: https://en.wikipedia.org/w/index.php?title=Itoh%E2%80%93Tsujii_inversion_algorithm&oldid=1270446894

[11] M. Bellare and P. Rogaway, "The Birthday Problem," *Introduction to Modern Cryptography*. pp. 273–274, 2005. [Online]. Available: https://web.cs.ucdavis.edu/~rogaway/classes/227/spring05/book/main.pdf

[12] P. C. van Oorschot and M. J. Wiener, "Parallel collision search with application to hash functions and discrete logarithms," in *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, in CCS '94. Fairfax, Virginia, USA: Association for Computing Machinery, 1994, pp. 210–218. doi: 10.1145/191177.191231.

[13] Wikipedia contributors, "Birthday attack — Wikipedia, The Free Encyclopedia." [Online]. Available: https://en.wikipedia.org/w/index.php?title=Birthday_attack&oldid=1309344704

[14] G. Brassard, P. HØyer, and A. Tapp, "Quantum cryptanalysis of hash and claw-free functions: Invited paper," in *LATIN'98: Theoretical Informatics*, Springer Berlin Heidelberg, 1998, pp. 163–169. doi: 10.1007/bfb0054319.

[15] Wikipedia contributors, "Root of unity — Wikipedia, The Free Encyclopedia." [Online]. Available: https://en.wikipedia.org/w/index.php?title=Root_of_unity&oldid=1313928663