Post-quantum Cryptography

Lecture notes for the winter term 2024/25

Jan Luca de Riese Dominique Unruh

2025-11-25

Table of contents

Welcome

These are the lecture notes for the "Introduction to Quantum Computing" lecture held by Dominique Unruh at RWTH Aachen in the summer term 2025. They should be viewed as an addition to the handwritten notes and the lecture recordings.

If you spot an error, please report them via Gitlab. If you have a question of understanding, please ask it in the Moodle forum.

The Jupyter notebooks created during the lectures can be found in the JupyterHub in the course "[IQC] Introduction to Quantum Computing" or in Moodle. If changes are made to the files, they can be easily reset with the usual git commands using "Git" and then "Open Git Repository in Terminal".

1 Opening Remarks

1.1 What is post-quantum crypto?

Quantum computers are not like classical computers, in the sense that while classical computers work on bits, 0s and 1s, quantum computers work on qubits. Instead of being either in the state 0 or 1, these qubits can be in a superposition of the two states. Using this difference in structure, quantum computers can perform *some* calculations very quickly.

Designing clever algorithms that take advantage of the way qubits interact, has already delivered attacks existing and ubiquitous crypto systems. For example, Shor's algorithm [9] factors large integers very quickly, breaking RSA. Shor's algorithm also computes discrete logarithms, thereby breaking Elliptic curve cryptography. Another example of a widely known quantum algorithm is Grover's algorithm [1]. This algorithm accelerates search, reducing a brute force attack on a symmetric cipher (with a 256-bit key) from 2^{256} steps classically, to 2^{128} steps quantumly.

Because of these attacks that are made possible by quantum computers, we need to secure our crypto systems against quantum attacks. Not only developing but also deploying new standards in a thorough and secure way takes years, waiting for a quantum computer to become available is not feasible. Additionally, harvest-now-and-decrypt-later attacks mean that super sensitive data already needs to be protected against such attacks today.

Post-quantum crypto is about cryptographic systems that are classical (e.g. they run on classical computers) but withstand attacks executed using Quantum computers.

To fulfill these expectations, NIST has run a PQC standardization process. In this, researchers worldwide enter a competition where they suggest and analyze signature as well as encryption schemes. NIST picks the winners after multiple rounds of competition (e.g. Kyber, Dilithium, Falcon, Sphincs).

This lecture is about methods in post-quantum crypto. We will look at some candidates for algorithms, as well as some unique challenges that the quantum setting brings with it. To reiterate: the goal of post quantum cryptography is to deliver cryptographic protocols that run on classical computers, but withstand quantum attacks. When doing this, two branches of PQC exist in practice.

The first branch concerns itself with finding and understanding quantum attacks. Figuring out which schemes are not secure against quantum attacks, and especially why they are not secure is important. This helps peer review the existing attacks, preventing weak algorithms from being used in the real world. And, even better, the learnings from this help us understand how to better secure our own schemes from exactly these attacks. (Or maybe how secure is secure enough)

Additionally, we want to find secure schemes, and then prove that they are actually secure. There are two approaches to constructing schemes. Ideally, you would want provable security, where you can mathematically guarantee certain properties of our system. Sometimes, however, that is not possible. (This of course does not mean that the maths say its insecure, just that the maths are maybe not applicable.) Whenever mathematically proving security is not feasible, we follow best practices and make use of the peer-review mechanism to strengthen trust in our scheme.

However, even the provably secure schemes are often based on a computational hardness assumptions. These assumptions are then treated similarly to the best-effort schemes were in the first place.

In this lecture we will see schemes out of both of these branches.

If you want to read more in-depth about these topics, take a look at [6] for the quantum background, as well as [3] and [8] for classical crypto notions. The concepts from classical crypto are still very useful, since many of the concepts in this lecture are analogous concepts.

2 Recap of Quantum mechanics

This is a very brief recap of the most important things you need to know about quantum mechanics for this lecture. You can find more in the script from intro to quantum computing.

Definition 2.1 (Quantum Systems)**.** Quantum systems are described by a set $\{1, \dots, N\}$ of labels, called classical possibilities or classical states.

A quantum system is in a certain state. We call this state a quantum state.

Definition 2.2 (Quantum State)**.** A quantum state is a vector $\psi \in \mathbb{C}^N$, $\|\psi\| = 1$ ($\sqrt{\Sigma|\psi_i|^2} = 1$). ψ_i is the "amplitude" of classical possibility i. $|\psi_i|^2$ is the probability of classical possibility i.

Examples

In Ket-notation $|n\rangle$ denotes the n-th entry being 1. E.g.

$$|n\rangle = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \leftarrow 1 \text{ at the } i\text{-th position}$$

We also write $|\psi\rangle$, to emphasize that ψ is a quantum state. Some often used states include:

$$|0\rangle := \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad |1\rangle := \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad |+\rangle := \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} \quad |-\rangle := \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix}$$

These quantum states can have operators applied to them. These operators are matrices, more specifically unitaries.

Definition 2.3 (Unitary matrices)**.** Unitary matrices describe the Evolution of a quantum system: $|\psi\rangle \to \mathrm{U}\,|\psi\rangle$. Here U must be a unitary matrix. This means U is $\in \mathbb{C}^{N\times N}$ and $U^\dagger U = UU^\dagger = I$. $U^\dagger$ is the conjugate transpose of U and I is the identity matrix. Unitary operations are length preserving, e.g. $\|U\,|\psi\rangle\| = \||\psi\rangle\|$.

After doing computations, you would naturally want to look at the results. To find out what state our system is in, we need to measure it.

Definition 2.4 (Complete Measurement)**.** When we measure a state ψ in its entirety, we get outcome i with probability $|\psi_i|^2$. The post-measurement-state after measuring outcome i is $|i\rangle$.

Instead of measuring the entire state, we might only want information about some of its components.

Measuring partially causes only the measured part of the system to collapse. This means the qubit(s) we measure decide what the partitions are.

Definition 2.5 (Partial Measurement)**.** When we partially measure a state ψ, we get one alternative $A_j \subseteq \{1, \dots, N\}$, where each A_j forms a partition.
The state is then in partition A_j with probability $\sum_{x \in A_j} |\psi_x|^2$. This would result in the non-normalized post-measurement-state $\tilde{\psi} = \begin{cases} \psi_i \text{ if i} \in A_j \\ 0 \text{ if i} \notin A_j \end{cases}$. We can then get the normalized post-measurement state like this: $\psi = \frac{\tilde{\psi}}{\|\tilde{\psi}\|}$.

Example

What is the first bit of a two bit state?
$A_0 = \{00, 01\}, A_1 = \{10, 11\}$
The probability of outcome j would then be $\sum_{x \in A_j} |\psi_x|^2 = \begin{cases} |\psi_{00}|^2 + |\psi_{01}|^2 (j = 0) \\ |\psi_{10}|^2 + |\psi_{11}|^2 (j = 1) \end{cases}$
If we measured outcome 1, our state would be: $\begin{pmatrix} 0 \\ 0 \\ \gamma \\ \delta \end{pmatrix}$, and after normalizing $\begin{pmatrix} 0 \\ 0 \\ \frac{\gamma}{\sqrt{|\gamma^2|+|\delta^2|}} \\ \frac{\delta}{\sqrt{|\gamma^2|+|\delta^2|}} \end{pmatrix}$

You can compose two quantum systems together. If we have quantum system A with classical possibilities $\{1, \dots, N\}$ and quantum system B with possibilities $\{1, \dots, M\}$, the composite system AB has the following classical possibilities $\{(1,1), \dots, (1,M), (2,1), \dots, (N,M)\}$

Similarly, we can compose quantum states. The states $\phi \in \mathbb{C}^N$ and $\psi \in \mathbb{C}^M$ of our systems A and B respectively can be composed as $\phi \otimes \psi$.

Example

$$\phi \otimes \psi = \begin{pmatrix} \phi_1\psi \\ \vdots \\ \phi_N\psi \end{pmatrix} = \begin{pmatrix} \phi_1\psi_1 \\ \vdots \\ \phi_1\psi_M \\ \phi_2\psi_1 \\ \phi_N\psi_M \end{pmatrix}$$

The result vector then has $N \times M$ entries.

Warning

While you can always tensor two states together, there are states of AB that *cannot* be written separately as $\phi \otimes \psi$. These entangled states are the result of ϕ and ψ having already interacted.

When writing down quantum operations, we do so using so-called quantum circuits.

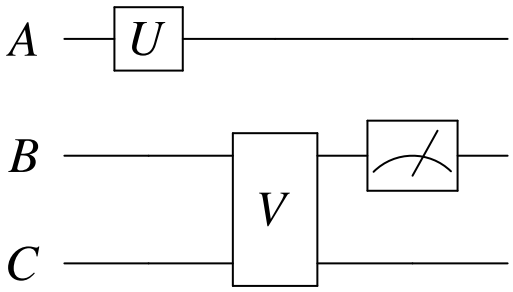


Figure 2.1: A basic quantum circuit

In this example circuit, we have three qubits, A, B and C. We first apply the unitary U to the first qubit, but not to the second and third (this is written $\mathrm{U} \otimes \mathrm{I} \otimes \mathrm{I}$). Then, we apply the unitary V on qubit B and C, and finally we measure qubit B. Mathematically, the second operation can be written as $\mathrm{I} \otimes \mathrm{V}$.

Example

Measuring just the second wire would be the operation $I \otimes M \otimes I$. Here, M is measuring a single qubit and thus the alternatives for that wire are $M = \{\{0\}, \{1\}\}$. The system then

has either a 0 on wire B ($A_0 = \{000, 001, 100, 101\}$) or a 1 ($A_1 = \{010, 011, 110, 111\}$).

Common quantum transformations that are used often have names associated with them.

Example: Some Unitary transformations

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \quad H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

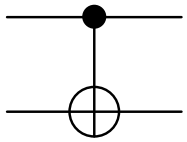
The X gate can be called a "bit flip", because it flips the basis states:

$$X|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle$$

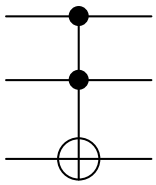
Hadamard is useful because it switches the basis from computational to diagonal and back:

$$\mathrm{H}|0\rangle = |+\rangle, \mathrm{H}|1\rangle = |-\rangle, \mathrm{H}|+\rangle = |0\rangle, \mathrm{H}|-\rangle = |1\rangle$$

Sometimes, large matrices are harder to visualize. A matrix such as CNOT can more easily be described by expressing what it does to an input. Like so: $\mathrm{CNOT}|x, y\rangle \to |x, y \oplus x\rangle$. (E.g. $\mathrm{CNOT}|10\rangle = |1, 0 \oplus 1\rangle = |11\rangle$). This describes the behavior of an operator in full, since the computational basis states that you plug in to the equation form a basis of the vector space.



(a) CNOT



(a) Toffoli

Visually, you would express CNOT as the circuit displayed above. Here, the dot always represents the controlling wire, which flips the other connected wire depending on its state. Using more than one controlling wire makes a gate called a Toffoli.

With Toffoli, CNOT and X we can implement any classical function $f : \{0,1\}^n \to \{0,1\}^m$

quantumly. Since functions work differently than the matrices we normally apply as operators when working in a quantum system, we need to represent this in a meaningful way. We use $U_f|x,y\rangle \to |x, y \oplus f(x)\rangle$, with $x \in \{0,1\}^n$ and $y \in \{0,1\}^m$. Since quantum operations are always reversible, the output has to contain both the input and the output of f. Using Toffoli, CNOT and X, we can implement U_f with approximately twice the number of gates that a classical implementation of f would use.

We can also replace Toffoli, CNOT and X with other gates. Any set of gates that, together, can replace any gate at all, is called a universal set of gates. There is a constant factor in runtime, but otherwise it does not matter what universal set you use.

3 Recap: Grover and Shor

We looked at two algorithms: Grover and Shor.

3.1 Shor's Algorithm

In order to look at Shor, we need to understand the discrete Fourier transform, or DFT. The matrix for DFT is defined as follows.

Definition 3.1 (Discrete Fourier Transformation (DFT))**.** The discrete Fourier transform (DFT) is a linear transformation on $\mathbb{C}^M$ represented by the matrix

$$\mathrm{DFT}_M = \frac{1}{\sqrt{M}}(\omega^{kl})_{kl} \in \mathbb{C}^{M\times M}$$

with $\omega = e^{2i\pi/M}$, which is the M-th root of unity. This means that ω taken to the power of 2^n equals 1.

We use the DFT to map a t-periodic vector to a vector with non-zero entries at multiples of $\frac{2^n}{t}$. The DFT can be implemented as a unitary operation on n qubits using $O(n^2)$ gates.

Shor's algorithm is used for period finding. Given a function $f(x) = f(x+t)$, we are looking for t.

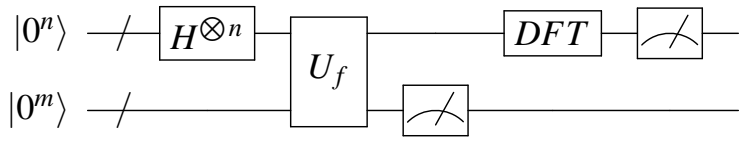


Figure 3.1: The circuit for Shor's algorithm

The algorithm works as follows:

1. We start with a $|0\rangle$ entry on every wire.
2. We bring the top wire into the superposition over all entries. The quantum state is then $2^{\frac{-n}{2}} \sum_x |x\rangle \otimes |0^m\rangle$.
3. We apply U_f, which is the unitary corresponding to $f : \{0,1\}^n \to \{0,1\}^m$. This calculates the superposition over all possible values $f(x)$ on the bottom wire. The resulting quantum state is $2^{\frac{-n}{2}} \sum_x |x, f(x)\rangle$.
4. To understand the algorithm better, we measure the bottom wire at this point. This will give us one random value $f(x_0)$ for some x_0. The top wire will then contain a superposition over all values x where $f(x) = f(x_0)$. Since f is known to be t-periodic, we know, that $f(x) = f(x_0)$ iff $x \equiv x_0 \bmod t$. This means that the resulting quantum state on the top wire is periodic and can be written as $\frac{\sqrt{t}}{\sqrt{2^n}} \sum_{x \equiv x_0 \bmod t} |x\rangle \otimes |f(x_0)\rangle$. For simplicity we assume, that $t \mid 2^n$ holds.
5. We apply the Discrete Fourier Transform on the top wire. This will "analyze" the top wire for the period and output a vector with entries at multiples of $\frac{2^n}{t}$. For simplicity we still assume, that $t \mid 2^n$ holds.
6. We measure the top wire and get one random multiple of $\frac{2^n}{t}$, which we can denote as $a \cdot \frac{2^n}{t}$

After some post-processing, this gives us t. (Note that if t is not divisible by 2^n, things get more complicated in the post-processing, but it still works)

3.1.1 Factoring Integers

If we want to factor M, we would pick a random a relatively prime to M. We then use Shor to compute $r := \operatorname{ord} a \pmod M$. This means it is the smallest $r > 0$ such that $a^r \equiv 1 \pmod M$. We can do this by computing the period of the function that maps x to $a^x \bmod M$. For an even r, we can now say that $(a^{\frac{r}{2}+1})(a^{\frac{r}{2}-1}) \equiv a^r - 1 \equiv 0 \pmod M$. It follows, that this product is a multiple of M. Every prime factor of M is in one of the terms. If we are lucky, and not

all factors are in the same term, $gcd(a^{\frac{r}{2}+1}, M)$ is a nontrivial factor of M. In a typical crypto case, where M has only two factors, we are done.

Shor thus factors integers in $O(|M|^3)$. This is fast enough to practically break RSA. (Assuming you have quantum computers that can handle such an algorithm)

3.2 Grover's algorithm

Assume we are given a function $f : \{0,1\}^n \to \{0,1\}$, which maps $f(x_0)$ to 1 and everything else to 0. Our goal here is to find x_0. For this, we need two unitaries, V_f and FLIP_*. V_f, which we can implement using U_f, is defined as follows:

$$V_f |x\rangle = \begin{cases} -|x\rangle & \text{if } f(x) = 1 \\ |x\rangle & \text{else} \end{cases}$$

FLIP_* leaves the uniform superposition untouched, and flips the sign of everything that is orthogonal to it.

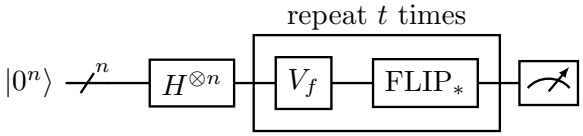


Figure 3.2: The circuit for Grover's algorithm

The algorithm works as follows:

1. We start with a $|0\rangle$ entry on every qubit.
2. We bring the system into the superposition over all entries by applying $H^{\otimes n}$. The quantum state is then $2^{\frac{-n}{2}} \sum_x |x\rangle$ which we also call $|*\rangle$.
3. We apply the unitary V_f.
4. We apply the unitary FLIP_*.
5. We repeat steps 3 and 4 t-times, and then do a measurement.

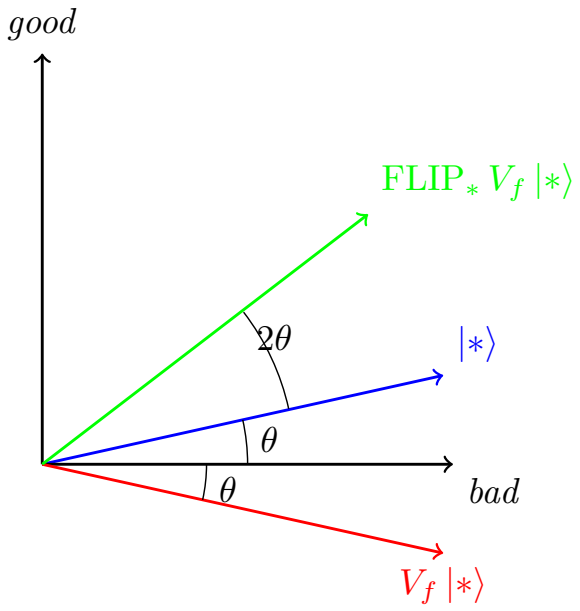


Figure 3.3: Visualization of Grover

We are searching for $|x_0\rangle$, here called the good state. The state orthogonal to it we call the bad state space.Our original state when starting the computation is close to the bad state. Every application of Grover rotates our state 2θ towards the goal state. Theta is the angle between $|x\rangle$ and $|bad\rangle$. It can be calculated as $\cos^{-1}\sqrt{\frac{2^n-1}{2^n}}$. After $t \approx \frac{\pi}{4}\sqrt{2^n}$ iterations we have rotated by $t \cdot 2\theta \approx \frac{\pi}{2} \approx |x_0\rangle$. Therefore the final measurement gives us x_0.

Grover runs in $O(\sqrt{2^n})$ evaluations of f, while the classically best algorithm takes $O(2^n)$ time.

With some tweaks, we can make our algorithm work even if there are multiple x_0 for which $f(x_0) = 1$. Generally, if our number of x_0's is T, our runtime is $\approx \sqrt{\frac{2^n}{T}}$. This is again the square root of the classical runtime. This algorithm is useful for attacking cryptographic systems, making the search for a 128-bit key take $\approx 2^{64}$ instead of the classical $\approx 2^{128}$.

This concludes the background required for the lecture. From the next chapter onwards, we will dive into the content of the lecture.

4 Hashing

4.1 Symmetric Cryptography

Symmetric Cryptography refers to cryptographic systems where encryption and decryption happen with the same key. It includes for example hash functions, block ciphers as well as symmetric encryption schemes.

4.1.1 Hash Functions

A hash function takes an arbitrary length bit-string and maps it to a fixed length output. It generally looks like this: $H : \{0,1\}^* \to \{0,1\}^n$. Examples of well-known hash functions include MD5, SHA1, SHA3.

We use hash functions for a variety of different purposes:

- They can help identify a long piece of data m from a short hash $H(m)$.
- If we hash a short seed, it can easily be used for random number generation.
- Hash functions need to be hard to invert (even partially). This is for example used in proofs of work for blockchains.
- Time-stamping and signatures, by signing or storing the hash instead of the message itself.
- Building blocks in bigger cryptographic constructions, etc…

Hash functions need to fulfill certain security requirements. They need to possess, among others, collision resistance, one-wayness and pseudorandomness.

4.1.1.1 Collision resistance

Collision resistance means it is hard to find two inputs to the hash function which share the same output. A hash function is collision resistant iff it is hard to find $x \neq x'$ such that $H(x) = H(x')$. A hash function without this resistance would be problematic because finding e.g. two documents with the same hash, you could substitute the documents without anyone noticing.

Collisions of course always exist, because the output is shorter than the input. We just need collisions to be hard to find.

Definition 4.1 (Collision Resistance (simplified))**.** H is (t, ϵ)-collision-resistant iff $\forall t$-time adversaries A: $\Pr[x \neq x' \wedge H(x) = H(x') : (x, x') \leftarrow A] \leq \epsilon$

This means, the probability that A outputs a collision x,x' needs to be smaller than ϵ

> **Warning**
>
> This definition is, unfortunately, impossible to satisfy for reasonable (t, ϵ). Since (x, x') exist, we can just create an adversary that ouputs one hard-coded collision always and instantly. In practice this definition still makes sense, this collision of course still takes infeasibly long to find.

(This can be made formal, see [7]. Alternatively, one can choose to use a variant where the hash function depends on some parameter or key k.) We now ask ourselves: How secure can a hash function be in the best case, classically or quantumly?

4.2 The Birthday Attack

The birthday problem is about the chance of two people in a room having the same birthday. For a surprisingly low number of people, this chance starts to get surprisingly high. The probability for two people in the room to have the same birthday is $\approx \frac{n^2}{2 \cdot 365}$ (as long as n is not too large).

Correspondingly, the birthday attack against collision resistance works as follows: We first pick $2^{\frac{n}{2}}$ different values $x_i \xleftarrow{\$} \{0,1\}^m$. With our collision defined as $H : \{0,1\}^m \to \{0,1\}^n$ with $m \gg n$, find $x \neq x'$ with $H(x) = H(x')$. We then compute $h_i := H(x_i)$ for all i. Since these are, essentially, random values, the probability of a collision is similar to before. $\Pr[\text{two } h_i \text{ are equal}] \approx \frac{(2^{n/2})^2}{2^n} \approx 1$. Now we find the collision in the h_i (just find two identical h_i), and output that collision. Thus we can find a collision using $O(2^{n/2})$ evaluations of H, and time $\tilde{O}(2^{n/2})$ (for sorting the h_i's to find n duplicates). Here, it's important to note that $\tilde{O}$ is a "soft" notation, meaning we ignore logarithmic factors.

We now take a look at a variation of this attack. It is not necessarily useful in the classical case, but will make it easier to understand the quantum algorithm we are about to introduce.

- Pick $2^{\frac{n}{2}}$ values x_i
- $h := H(x_i)$ $\mathbb{D} := \{\underline{h_i}\}$, $\mathcal{L}_{\mathbb{D}}(z) := [H(z) \in \mathbb{D}]$ (The square brackets here mean we evaluate whether it is true)
- For random z, $\Pr[\mathcal{L}_{\mathbb{D}}(z) = 1] \approx \frac{|\mathbb{D}|}{2^n} = \frac{2^{\frac{n}{2}}}{2^n} = 2^{-\frac{n}{2}}$
- Search z with $\mathcal{L}_{\mathbb{D}}(z) = 1$ by trying random z's.
- This succeeds after $\approx 2^{\frac{n}{2}}$ tries. If we found it, then the hash of z is in $\mathbb{D}$, which means $H(z) = h_i = H(x_i)$.
- Find x_i by table lookup. Output the collision (z, x_i).

In short, we pick a few values out of all possible ones, and hash them. All the hashes go into the set $\mathbb{D}$. This set is analogous to the room that we pick two people out of in the birthday problem. Only in this case, the hashes of the x_i are the birthdays, so to speak. We then randomly pick two z's to compare. This search is faster the more items are in the set $\mathbb{D}$. However, generating $\mathbb{D}$ takes time. Therefore, there is a trade-off. If $\mathbb{D}$ contains 2^{cn} and not $2^{n/2}$ many items, then search needs $2^{(1-c)n}$ iterations. This means, for us, that the number of evaluations is equal to $O(2^{cn} + 2^{(1-c)n}) = O(2^{max(c,1-c)\cdot n})$. (The first term being for populating $\mathbb{D}$, and the second term for searching). This is optimal for $c = \frac{1}{2}$

All in all, we found collisions using $O(2^{\frac{n}{2}})$ evaluations and $\tilde{O}(2^{n/2})$ time (assuming a suitable datastructure for representing the set $\mathbb{D}$) of H.

4.3 BHT algorithm (quantum)

We could now optimize this algorithm. In the quantum world, search is faster. We can use Grover for search. Search being faster, however has lost us our optimum point for the trade-off. Remember, our runtime is the maximum of the two. We therefore adjust by making the set $\mathbb{D}$ smaller in return. Parametrized with the factor c our algorithm looks like this.

- Pick $2^{c\times n}$ values x_i
- $h_i := H(x_i)$ $\mathbb{D} := \{\underline{d_i}\}$, $\mathcal{L}_{\mathbb{D}}(z) := [H(z) \in \mathbb{D}]$
- For random z, $\Pr[\mathcal{L}_{\mathbb{D}}(z) = 1] \approx \frac{|\mathbb{D}|}{2^n} = \frac{2^{c\times n}}{2^n} = 2^{(c-1))\times n}$
- Search z with $\mathcal{L}_{\mathbb{D}} = 1$ by **by using grover** .
- This succeeds after $\approx \sqrt{\frac{2^n}{|\mathbb{D}|}}$ tries. If we found it, then the hash of z is in $\mathbb{D}$, which means $H(z) = h_i = H(x_i)$.
- Find x_i by table lookup. Output the collision (z, x_i).

Our runtime then becomes $O(2^{cn} + 2^{(1-c)/2\times n}) = O(2^{max(c,(1-c)/2)\cdot n})$. The runtime therefore becomes optimal at $c = \frac{1}{3}$. For that value of c, both the number of elements in $\mathbb{D}$ and the runtime for rover are balanced. We conclude, that the algorithm takes $O(2^{n/3})$ queries as well as $\tilde{O}(2^{n3})$ time. Time here means the number of gates applied and the classical computation steps applied. This is a noticeable speedup over the classical case. We see that quantumly, Hash-functions still make sense, however some hash functions that might have seemed secure become breakable.

5 Block Ciphers

Block ciphers are deterministic symmetric ciphers with fixed length in- and output. A deterministic cipher always produces the same ciphertext given the same message and key, even if you execute it multiple different times. Using block ciphers, encryption and decryption happen with the same key. This makes them symmetric ciphers. What gives block ciphers their name, is that encryption happens in blocks of fixed length. Both message and ciphertexts are of that length.

Symmetric cryptography schemes have the main disadvantage that anyone who can encrypt can also decrypt. These schemes are therefore used in settings where this is not critical, e.g. as part of a bigger protocol or in a local setting only.

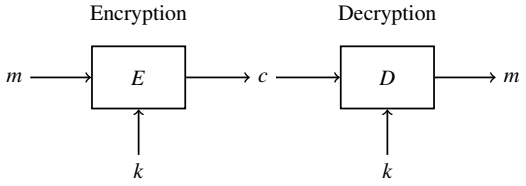


Figure 5.1: Symmetric encryption, schematically

One example for block ciphers is the Advanced Encryption Standard (AES), with a block length of 128 bit.

Block ciphers are then used in a so-called mode of operation. These modes of operation determine how block ciphers are applied to longer messages (In Cipher block chaining for example, messages are XORed with the previous block's ciphertext). Using block ciphers in such a mode results in another symmetric encryption scheme which allows encrypting longer messages and is more secure (like IND-CPA, IND-CCA, although more on these later).

We will ask the following questions in this lecture:

1. What security properties should block ciphers have?
2. What constructions of block ciphers are there?
3. What quantum attacks are there?

5.1 Security of block ciphers

We want block ciphers to be a so-called strong pseudorandom permutation. This means, intuitively, that the block cipher encryption looks like a random permutation, as long as the

key is not known to us. (In this case permutation does not mean a reordering of bits, but a bijective function).

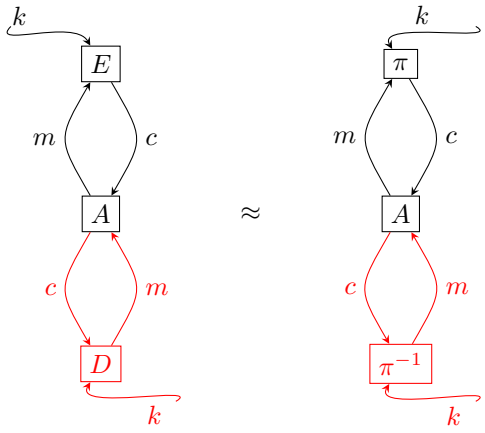


Figure 5.2: A game for pseudorandom permutations

Now, how do we define two things looking the same? A common proof technique in cryptography is specifying a game. If the adversary playing the game is not able to win, our algorithm/idea are safe from attack. In the case of block ciphers we construct the game as follows: The adversary is given a black box that encrypts any plain text given to it. The adversary can encrypt things as often as it would like, but then has to decide whether the function was a random permutation or the encryption scheme. If it can guess no better than 50%, the scheme is considered secure. We can also give the adversary the additional power of decrypting any ciphertext it wants (in the diagram shown in red). If it still cannot decide reliably we call this strong PRP.

Definition 5.1 (Pseudorandom Permutation)**.** (E, D) is a $(t, q, \epsilon)-$strong pseudorandom Permutation (PRP) iff:

- for all keys k: $E(k, \cdot)$ is a permutation
- for all k: $D(k, \cdot)$ is the inverse of $E(k, \cdot)$
- for all t-time, q-query quantum adversaries A: $|Pr[b = 1 : k \xleftarrow{\$} \mathcal{K}, b \xleftarrow{\$} A^{E(k,\cdot),D(k,\cdot)}()] - Pr[b = 1 : \pi \leftarrow Perm(\mathcal{M} \to \mathcal{M}), b \leftarrow A^{\pi,\pi^{-1}}()] \leq \epsilon$. Here $\mathcal{K}$ is key space, $\mathcal{M}$=$\mathcal{C}$ are message space and ciphertext space.

In this definition, we give access to the encryption function to the adversary. We denote this by writing A^E. This means it can query it as often as it wants. K denotes the key space, and $M = C$ denotes the message/ciphertext space. Note that this is two definitions in one,

depending on if you read the red part or not.

5.1.1 Getting a Strong Pseudorandom Permutation

Different PRPs choose different approaches. One you might have heard of is the Advanced Encryption Standard (AES). In AES, the message is fed through lots of very simple functions. These are called rounds, and each round depends on (some parts of) the key. Given the key, each round is easily inverted. The round itself might just shift some bits, or switch them out. In combination, these form a very secure algorithm.

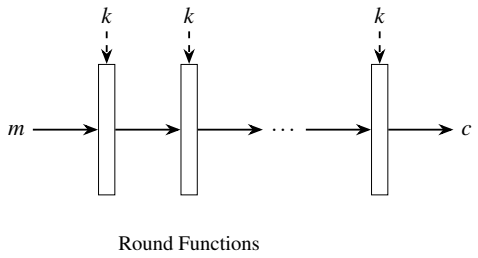


Figure 5.3: A schematic of AES rounds

5.1.2 Feistel networks

A different approach is called a Feistel network. Let us say our message space is $\mathcal{M} = \{0,1\}^{2n}$ and we have a function $F_k : \{0,1\}^n \to \{0,1\}^n$, which is random looking, but depends on the key. This function has more lax requirements than a permutation, because it does not need to come with a decryption, nor does it even need to be injective at all. Still, we will manage to build a permutation out of it. We simply split our message into two halves: m_1 and m_2. We then take these halves and pass them through functions F_k (they need not all be the same), alternating between m_1 and m_2, like shown below. This is known as a Feistel network. If you repeat this long enough, this becomes a good block cipher. This is classically well understood, if F is a pseudorandom function, and the Feistel network has 3/4 rounds, you get a PRP/strong PRP. The same holds in the postquantum setting, but not when we allow superposition queries (see Section N/A).

Further reading:

- [5] shows classical security of 3/4 round Feistel.
- [[2]][4] show that 3/4 rounds are not sufficient in the quantum case (with superposition queries).

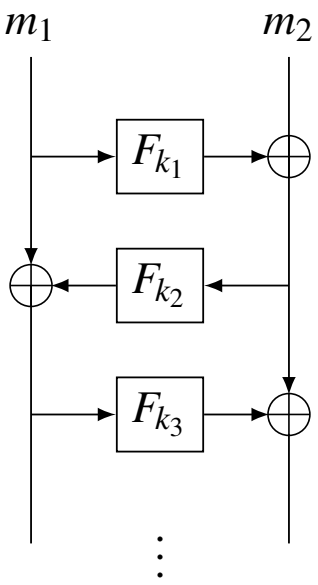


Figure 5.4: A Feistel Encryption ($\oplus$ means XOR)

To decrypt, you only need to turn the procedure on its head. We do everything in reverse.

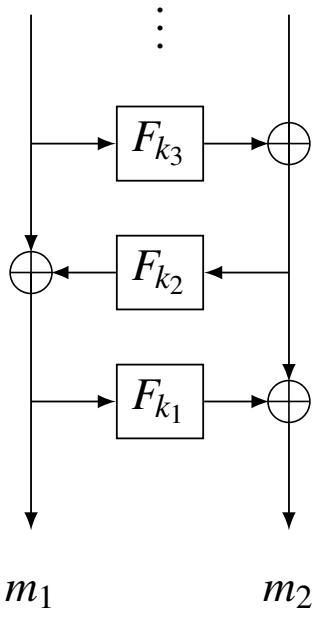


Figure 5.5: A Feistel Decryption

5.1.3 Even-Mansour

We start off with a permutation P, as follows. $P : \{0,1\}^n \to \{0,1\}^n = \mathcal{M}$. This permutation is publicly known, as is its inverse. We also assume that it behaves like a random permutation (whatever that means formally). Constructing a block cipher from it is the next step. The block cipger has key space $\{0,1\}^{2n}$; each key consists of two halves $k_1, k_2 \in \{0,1\}^n$. Encrypting works by first XORing the message with k_1, permuting it, and then XORing it with k_2 ($k_1, k_2 \in \{0,1\}^n$). Similarly, we decrypt by XORing with k_2, using the permutation's inverse, and then XORing with k_1. More formally: $E(k = k_1 \parallel k_2, m) = P(m \oplus k_1) \oplus k_2$ and $D(k = k_1 \parallel k_2, c) = P^{-1}(c \oplus k_2) \oplus k_1$.

Classically, Even-Mansour is a strong PRP for an adversary making less than $2^{n/2}$ queries. There exist attacks that break it in around that many queries. These are key recovery attacks, making it a relatively strong attack.

5.1.4 Attacking EM with quantum computers

This will work similarly to our attack on hash functions. We start of by picking $2^{n/3}$ values m_i. We also construct our set $\mathbb{D} = \{d_i\}$, and to go along with it, our function $\mathcal{L}_{\mathbb{D}}(z) = [P(z) \oplus P(\bar{z}) \in \mathbb{D}]$. In this scenario our key is kl, and $d_i = E_{kl}(m_i) \oplus E_{kl}(\bar{m_i})$. Here $\bar{m_i}$ is the bitwise negation of m_i. Note that since we can compute E_{kl} only by invoking the encryption oracle, we cannot evalate functions involving E_{kl} in superposition, and thus cannot run Grover with such functions. (E.g. we cannot use Grover to search for some m with $E_{kl}(m)=0.) However, P is a public permutation, i.e. completely known to the adversary, thus we can apply Grover to functions involving P.

For any random z, our probability for $L_{\mathbb{D}}(z)$ to evaluate to 1 is very low. Both $P(z)$ and $P(\bar{z})$ are essentially random. Thus, we once again have a chance of success of $\frac{|\mathbb{D}|}{2^n} = \frac{2^{\frac{n}{3}}}{2^n} = 2^{-\frac{2}{3}n}$. We then find the z with Grover. This takes us $\sqrt{\frac{1}{2^{-\frac{2}{3}n}}} = \sqrt{2^{\frac{2}{3}n}} = 2^{\frac{n}{3}}$ evaluations of $L_{\mathbb{D}}$. Luckily, we do not have to balance the two parts of the algorithm, because they already are.

We now have a z with $P(z) \oplus P(\bar{z}) \in \mathbb{D}$. We want to find (using a table), an m_i, such that $E_{kl}(m_i) \oplus E_{kl}(\bar{m_i}) = d_z$. This necessarily exists, because of $P(z) \oplus P(\bar{z})$ being in $\mathbb{D}$. If we now use the definition of EM, we see that $P(z) \oplus P(\bar{z}) = P(m_i \oplus k) \oplus l \oplus P(\bar{m}_i \oplus k) \oplus l$. We can get rid of the ls, and then z could have the following values: Either $z = m_i \oplus k$, or $z = \bar{m}_i \oplus k$. For random P, these are the only cases, up to a small probability. (Proof ommitted.) Turning the equations around, our key is $k = m_i \oplus z$, or $k = \bar{m}_i \oplus z$. Getting l is now also possible: we evaluate (classically) $E_{kl}(0) \oplus P(k) = P(0 \oplus k) \oplus l \oplus P(k) = l$. All that is left is to check whether kl is correct, by letting the oracle encrypt something random for us, and then encrypting it ourselves to compare.

In summary, we used $O(2^{n/3})$ queries to recover the key. Some, like the E_{kl}-queries were classical, and the P-queries we used were quantum.

6 Superposition attacks

While before, we looked at blockciphers, and considered the security of them with and without the existence of quantum computers. Quantum computers weakened the algorithm we looked at but, without completely breaking it. This is very typical of symmetric crypto. Today we will look at a situation where quantum computers can make a more drastic difference, namely superposition attacks.

Recall attacks on block ciphers as we have seen so far:

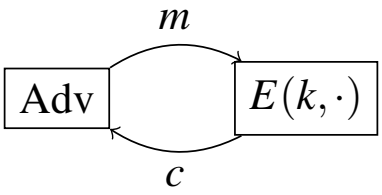


Figure 6.1: Chosen plaintext attack

In this attack, the adversary can asc to encrypt any m's with a certain fixed key k (unkown to the adversary). Now this exists with many different variants, the common theme is the adversary can talk to something to get information. This may be called challenger, oracle, or similar. Importantly, the message that is sent to the encryption oracle is classical, while the adversary themselves is a quantum computer internally.

This is because the real world scenario involves a classical crypto system (the honest parties), which is attacked by an adversary who has a quantum computer. Considering this is typical for post-quantum cryptography.

The encryption oracle represents an interaction with the protocol execution. In reality, the adversary might be able to inject messages and/or trick honest parties into encrypting them. We represent this using the oracle.

All of this means that the encryption oracle is classical.

Recall the attack against EM. First, the adversary ran a large number of queries to the encryption oracle, and in a second phase ran Grover's algorithm. The predicate that we are searching ($\mathbb{L}_{\mathbb{D}}(z) = [P(z) \oplus P(\bar{z}) \in \mathbb{D}]$) is queried in superposition. This is allowed because we do not query the oracle but the permutation P in superposition.

Since P represents a *publicly known* permutation, the adversary is able to construct a circuit to run it on their quantum computer. And, while the adversary could implement a quantum version of the encryption algorithm, it does not know the key k. This means that part has to happen in the system of the honest parties, and therfore classically.

In summary, when we say we do an attack with classical queries (or without superposition queries) we typically mean the following.

- When the adversary performs a query that computes a function involving the secrets not known to the adversary (e.g. E_{kl} in Section 5.1.4) the query is a classical query. (*Cannot* do superposition queries).
- When the adversary performs a query that computes a function involving only things known to the adversary (e.g. P in Section 5.1.4) the query is allowed to be a superposition query. (*Can* do superposition queries.)

CBUt somtimes, we can also consider "attacks with superposition queries". Here all queries are allowed to be superposition queries (e.g. also E_{kl}). Here are some arguments as to why.

- Due to miniaturization in chip technology, the honest party may accidentally execute things in superposition. For example if you get classical transistors down to one atom, they may behave quantumly. This is maybe a bit far fetched, but the argument has been made in the literature.
- We might in the future want to (honestly) use the encryption scheme on a quantum computer on quantum data. As a counterargument: There is something called quantum encryption for quantum data. Just use that!
- Sometimes, in security proofs, you do reductions that transform adversaries in complex ways. E.g., one adversary runs another adversary internally in superposition (which might run E). We see examples of such things in Sec N/A.

6.1 Extending Strong Permutations to quantum

Recall: Definition 5.1 of (Strong) PRPs. We will now define this for superposition queries.

Definition 6.1. (E, D) is a (t, ϵ)-strong qPRP iff:

- for all keys k: $E(k, \cdot)$ is a permutation
- for all k: $D(k, \cdot)$ is the inverse of $E(k, \cdot)$
- for all t-time quantum adversaries A: $|Pr[b = 1 : k \xleftarrow{\$} \mathcal{K}, b \xleftarrow{\$} A^{E(k,\cdot),D(k,\cdot)}()] - Pr[b = 1 : \pi \leftarrow Perm(\mathcal{M} \to \mathcal{M}), b \leftarrow A^{\pi,\pi^{-1}}()]| \leq \epsilon$. Here $\mathcal{K}$ is key space, $\mathcal{M}$=$\mathcal{C}$ are message space and ciphertext space.

Here all oracles $(E(k, \cdot), D(k, \cdot), \pi, \pi^{-1})$ are given to the adversary as quantum oracles. (A.k.a. superposition oracles.)

In the classical case, oracles are given a bit of data (e.g. a bitstring or a number) and then return their answer to that query (e.g. another bitstring or a bit). In the quantum case, we cannot do that anymore, since we now want a superposition query to be possible. Instead, when the adversary makes a query to a quantum oracle $\mathbb{O}$ (where $\mathbb{O}$ is a function $\mathbb{O} : \{0,1\}^n \to \{0,1\}^m$), the adversary provides two registers X and Y consisting of n and m qubits, and then the oracle applies the unitary $(U_{\mathbb{O}} |x, y\rangle \to |x, y \oplus \mathcal{O}(x)\rangle)$ applied to them.

6.2 Quantum-attacking Even-Mansour

This is quite a bit more powerful. Even-Mansour for example is completeley broken by using superposition-attacks. We get a key recovery with $\mathbb{O}(n)$ queries.

We will do this using Simon's algorithm. This algorithm solves the abstract problem of finding a function period. Let the classical function $f : \{0,1\}^n \to \{0,1\}^m$ be t-periodic with respect to $\oplus$. That is, $f(x) = f(x \oplus t)$. We want the period t to be the only period you can find, since the task is finding it. (This, and "normal" period finding are special cases of "hidden sub-group problems"). This problem is exponentially hard classically.

6.2.1 Simon's algorithm

We start off with two registers, one with n and m qubits respectively, and initialize them with zeroes. As a natural start, we apply a Hadamard to the first register, producing a uniform superposition over all possible values of this register ($2^{-n/2} \sum_x |x, 0\rangle$). We then use the function $\mathrm{U_f}$ to evaluate our function f in superposition ($2^{-n/2} \sum_x |x, f(x)\rangle$). As a final step we measure the lower wire, netting us a value y. The upper wire now contains the sum of all possible amplitudes of values x for which $f(x) = y$ ($forxs.t.f(x) = y : \alpha \sum_x |x, f(x)\rangle$). This means because of the periodicity, we can write the state as the superposition of the two points where the function are the same: $\alpha |x_0, f(x_0)\rangle + \alpha |x_0 \oplus t, f(x_0 \oplus t)\rangle$. This also makes it clear that alpha is $1/\sqrt{2}$. Therefore, we can write the state as: $1\sqrt{2}(|x_0\rangle + |x_0 \oplus t\rangle) \otimes |y\rangle$. The first part being on the top wire, and $|y\rangle$ on the bottom wire. Here, we apply the Hadamard because it has similar properties to the DFT.

A result we are going to use here, is that $H^{\otimes n} |x\rangle = 2^{-n/2} \sum_x (-1)^{xz} |z\rangle$ (*).

We then get the following state Ψ_1 on the first wire: $1/sqrt2H^{\otimes n} |x_0\rangle + 1\sqrt{2}H^{\otimes n} |x_0 \oplus t\rangle \overset{*}{=} \frac{2^{n/2}}{\sqrt{2}}(\sum_z (-1)^{x_0 \cdot z} |z\rangle + (-1)^{x_0 \cdot z + t \cdot z} |z\rangle)$.

Note: Here the inner product $(\cdot)$ is defined as: $a \cdot b = \sum a_i b_i \mod 2$.

When determining the content of the sum in the parantheses, we have to differentiate between two cases:

- $\pm 2 * |z\rangle$ (if $t \cdot z = 0$)
- 0 (if $t \cdot z = 1$)

This is equal to $\beta \sum_{zs.t.t \cdot z = 0} \pm |z\rangle$, meaning the outcome z satisfies $t \cdot z = 0$. In the end, this quantum circuit gives us a uniformly random z with $t \cdot z = 0$. This linear equation with one unknown, for which we can solve.

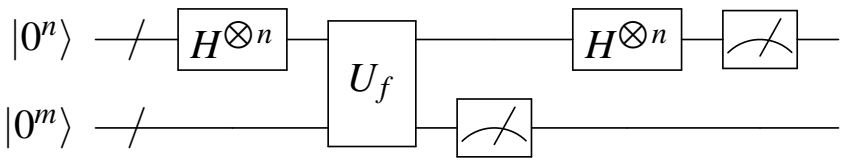


Figure 6.2: Simon's algo

6.2.2 The attack

Recall that Even-Mansour blockcipher was defined as $E(kl, m) = P(m \oplus k) \oplus l$. We define a function $f(x) := P(x) \oplus E(kl, x)$. Plugging in the definition of E gives us $f(x) := P(x) \oplus P(x \oplus k) \oplus l$. Does this function have a period? Indeed it does. Plugging $x \oplus k$ into f shows us that $f(x) := P(x \oplus k) \oplus P(x \oplus k \oplus k) \oplus l = f(x)$, meaning k is our period, and f is k-periodic.

We now need to construct a quantum circuit to compute $\mathrm{U_f}$.

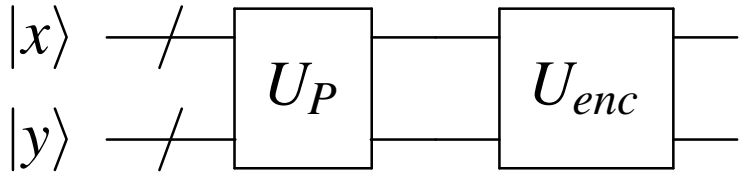


Figure 6.3: U_f as a quantum circuit

This is surprisingly simple. After applying U_P, the top wire has $|x\rangle$ on it, and $|y \oplus P(x)\rangle$ on the bottom wire. Now applying U_{enc} gives us still $|x\rangle$ on the top wire. The bottom wire however, has $|y \oplus P(x) \oplus E(kl, x)\rangle$. You might already recognize, that is exactly $|y \oplus f(x)\rangle$. That means

this computes U_f. Now, Simon's algorithm finds periodic k of f in $O(|k|) = O(n)$ evaluations of U_f, so also in just as many queries.

Though we do not have the key, we have half of it. Computing $E(kl, 0) \oplus P(k)$ (using an encryption query), which is equal to $P(0 \oplus k) \oplus l \oplus P(k)$. The k's cancel out, and we are left with l.

In summary, we computed the whole key kl in a linear number of queries. Therefore, following our definition, EM is not a qPRP.

	Best attack	Security proof
class	$2^{n/2}$	$2^{n/2}$
PRP	$2^{n/3}$	$2^{n/3}$
qPRP	n	-

This table shows the best attacks we now, and the security proofs for them. This means how fast we can currently break it, and whether this is already the theoretical maximum. This is the case for the first two attacks, with the runtime of the quantum attack already being so good no one has bothered to show if it can be done faster.

It should be noted that the classical and PRP attacks have large memory requirements. For the quantum evaluations, we need this memory to be quantum readable. This QROM is a steep requirement that bears to keep in mind.

The PRP attack can also be done in polynomial memory if you use a different attack that we will not cover here.

7 Cryptographic proofs – example

We now begin studying how we can demonstrate the security of an algorithm E' given that we know the security of another related algorithm E from which it is derived. Recall the definition of a (t, q, ϵ)-PRP (Definition 5.1). There we quantify over t-time q-query adversaries.

The *runtime* of an algorithm is defined as the number of steps that it runs in the worst case, without counting the time spent inside an oracle. Rather, every time an oracle is **directly** called by the algorithm, its *query count* is increased. This excludes calls to other oracles that are called by the first oracle. We illustrate how we reason about this in the following theorem and proof.

Theorem 7.1. *Let $E(k,m)$ be a (t,q,ϵ)-PRP. Then $E'(k,m) := rev(E(k,rev(m)))$ is a (t',q',ϵ')-PRP, where $t' = t - 2qt_{rev}$, $q' = q$, $\epsilon' = \epsilon$, and $rev(\mathcal{M})$ reverts the order of the bits of $\mathcal{M}$.*

Proof. This proof will proceed in a series of steps. First, we will fix a t'-time, q'-query adversary A attacking E', and we will define two games:

- Game 1: $k \xleftarrow{\$} \mathcal{K}, b \leftarrow A^{E'(k,\cdot)}()$
- Game 2: $\pi \xleftarrow{\$} Perm(\mathcal{M} \to \mathcal{M}), b \leftarrow A^{\pi}()$

These are the games from the definition of a PRP. Define a second adversary $B^{\mathcal{O}}$ that returns A^F, where $F(m) := rev(\mathcal{O}(rev(m)))$. Then, define the following games:

- Game 3: $k \xleftarrow{\$} \mathcal{K}, b \leftarrow B^{E(k,\cdot)}()$
- Game 4: $\pi \xleftarrow{\$} Perm(\mathcal{M} \to \mathcal{M}), b \leftarrow B^{\pi}()$

These are also the games from the definition of a PRP, but for an adversary B attacking the algorithm E. The query count of B is the same as the query count of A because B simulates A, and whenever A makes a query, B needs to evaluate $rev(\mathcal{O}(rev(m)))$, which contains one query to $\mathcal{O}$. So B queries $\mathcal{O}$ once for each query of A.

Similarly, the runtime of B can be obtained by the runtime of A plus the time spent evaluating $rev(\mathcal{O}(rev(m)))$ q' times, so $t'+2q't_{rev}$. Since $q = q'$ and $t'+2q't_{rev} = (t-2q't_{rev})+2q't_{rev} = t$, B is a t-time q-query adversary. Since E is a (t,q,ϵ)-PRP by assumption, we have $|\Pr[b = 1 : \text{Game 3}] - \Pr[b = 1 : \text{Game 4}]| \leq \epsilon = \epsilon'$.

If we unfold/inline the definition of B in Game 3 and simplify a bit, we get:

- Game $3_{\text{refactored}}$: $k \xleftarrow{\$} \mathcal{K}, b \leftarrow A^{E(k,\cdot)}()$,

where we used that $rev(E'(k,rev(m))) = E(k,m)$ to rewrite the oracle call of A. Since this is just rewriting the definition of Game 3, we have that $\Pr[b = 1 : \text{Game 3}] = \Pr[b = 1 : \text{Game } 3_{\text{refactored}}] = \Pr[b = 1 : \text{Game 1}]$. Similarly, we define:

- Game $4_{\text{refactored}}$: $\pi \xleftarrow{\$} Perm(\mathcal{M} \to \mathcal{M}), b \leftarrow A^{F}()$,

where $F(x) = rev(\pi(rev(x)))$. Since π is a random permutation, so is F, and thus we have $\Pr[b = 1 : \text{Game 4}] = \Pr[b = 1 : \text{Game } 4_{\text{refactored}}] = \Pr[b = 1 : \text{Game 2}]$. This finally leads us to $|\Pr[b = 1 : \text{Game 1}] - \Pr[b = 1 : \text{Game 2}]| = |\Pr[b = 1 : \text{Game 3}] - Pr[b = 1 : \text{Game 4}]| \leq \epsilon$. Thus A has success probability $\leq \epsilon'$ in attacking E', so E' is a (t',q',ϵ')-PRP. ■

Bibliography

[1] Lov K Grover. "A fast quantum mechanical algorithm for database search". In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 1996, pp. 212–219.

[2] Gembu Ito et al. "Quantum chosen-ciphertext attacks against Feistel ciphers". In: *Topics in Cryptology–CT-RSA 2019: The Cryptographers' Track at the RSA Conference 2019, San Francisco, CA, USA, March 4–8, 2019, Proceedings*. Springer. 2019, pp. 391–411.

[3] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography: principles and protocols*. Chapman and hall/CRC, 2007.

[4] Hidenori Kuwakado and Masakatu Morii. "Quantum distinguisher between the 3-round Feistel cipher and the random permutation". In: *2010 IEEE international symposium on information theory*. IEEE. 2010, pp. 2682–2685.

[5] Michael Luby and Charles Rackoff. "How to construct pseudorandom permutations from pseudorandom functions". In: *SIAM Journal on Computing* 17.2 (1988), pp. 373–386.

[6] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010.

[7] Phillip Rogaway. "Formalizing human ignorance: Collision-resistant hashing without the keys". In: *International Conference on Cryptology in Vietnam*. Springer. 2006, pp. 211–228.

[8] Mike Rosulek. *The Joy of Cryptography*. https://joyofcryptography.com. 2021. URL: https://joyofcryptography.com.

[9] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM Journal on Computing* 26.5 (Oct. 1997), pp. 1484–1509. ISSN: 1095-7111. DOI: 10.1137/s0097539795293172. URL: http://dx.doi.org/10.1137/S0097539795293172.