HAW HAMBURG

Bachelorarbeit

Lorenzo Senarigo

Unclonable Secret Sharing: Analysis and Prototype Implementation

Faculty of Engineering and Computer Science
Department Computer Science

Lorenzo Senarigo

Unclonable Secret Sharing: Analysis and Prototype Implementation

Bachelor thesis submitted for examination in Bachelor´s degree
in the study course *Bachelor of Science European Computer Science*
at the Department Computer Science
at the Faculty of Engineering and Computer Science
at University of Applied Science Hamburg

Supervisor: Prof. Dr. Heike Neumann
Supervisor: Prof. Dr. Klaus-Peter Kossakowski

Submitted on: 4th January 2025

Lorenzo Senarigo

Thema der Arbeit

Unclonable Secret Sharing: Analysis and Prototype Implementation

Stichworte

Unclonable cryptography, secret sharing, unclonable secret sharing, quantum computing, qiskit

Kurzzusammenfassung

Die Quantenmechanik und insbesondere das No-Cloning-Theorem bieten die faszinierende Möglichkeit, eine neue Familie kryptographischer Primitive mit der Eigenschaft der Unklonbarkeit zu schaffen. In Verbindung mit Secret Sharing führte dies zur theoretischen Konstruktion von unklonbarem Secret Sharing, einem vielversprechenden Ansatz, der die Wahrscheinlichkeit begrenzt, dass Anteile eines Geheimnisses zu zwei gleichzeitig korrekten Rekonstruktionen des Geheimnisses führen. Diese Arbeit untersucht diese Konstruktion, indem eine praktische Implementierung mit `qiskit` entwickelt wird. Durch experimentelle Tests dieser Implementierung stellen wir fest, dass die vorgeschlagene Konstruktion für unklonbares Secret Sharing die Funktionsanforderungen eines allgemeinen Secret-Sharing-Schemas erfüllt und dass ihre Unklonbarkeitseigenschaft gegen ausgewählte Angriffe bestand hat.

Lorenzo Senarigo

Title of Thesis

Unclonable Secret Sharing: Analysis and Prototype Implementation

Keywords

Unclonable cryptography, secret sharing, unclonable secret sharing, quantum computing, qiskit

Abstract Quantum mechanics and specifically the no-cloning theorem offer the stimulating possibility of creating a new family of cryptographic primitives with the property

of unclonability. In conjuction with secret sharing, this approach led to the theoretical construction of unclonable secret sharing, a promising result that limits the probability that shares of a secret lead to two simultaneously correct reconstructions of the secret. This work investigates this construction by designing a practical implementation with `qiskit`. By experimentally testing this implementation, we find that the proposed construction for unclonable secret sharing satisfies the functionality requirements of a general secret sharing scheme and that its unclonability property holds against selected attacks.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abbreviations

QM Quantum Mechanics.

SS Secret Sharing.

USS Unclonable Secret Sharing.

1 Introduction

1.1 Motivation

The convergence of quantum mechanics and computer science into the field of quantum computing has, ever since its birth, been a double-edged sword for cryptographers.
On the one hand, the laws of quantum physics have been employed to create cryptographic primitives of information-theoretical security, that contrast all the cryptographic systems that rely, for instance, on one-way functions and whose security proof is dependent on the ever-so-problematic problem of $P = NP$,[9] and hardness assumptions, thus sparking the field of quantum cryptography. On the other hand, the power of quantum computing allowed for a prospective demolition of some of classical cryptography's pillars.
Even if conceived as early as the late 1960s with Wiesner's "Conjugate Coding"[29], only published in 1983, quantum cryptography established itself after Bennett and Brassard's paper on secure quantum key distribution was published, [3]. Since then quantum cryptography has kept improving on the foundations laid by BB84, trying to overcome the many technical obstacles on the way to security and by expanding to other cryptographical fields, such as quantum digital signatures. On the other hand, quantum computing also enabled advancements in the resolution of number-theoretic problems on which the security of certain widely employed cryptographic primitives relied. Specifically, when Shor provided a quantum polynomial-time solutions to the integer factorization problem, the discrete logarithm problem and the elliptic curve discrete-logarithm problem, he rendered RSA, Diffie-Hellman key-exchange and elliptic curve cryptography useless if faced by a sufficiently powerful quantum computer [25]. Quantum computing also had a significant impact on secret sharing, a cryptographic primitive allowing distribution of a secret to a set of parties, in a way that prevents them from gaining any information on the secret unless a certain amount or configuration of their partial information is pooled. Secret sharing is a field of cryptography that is not threatened by quantum computers,

as quantum-safe secret sharing schemes exist; it can instead be improved by employing quantum computing tools, in order to overcome weaknesses of classical secret sharing. Specifically, promising results have been recently published introducing a so-called unclonable secret sharing scheme (sometiems abbreviated in USS) [1]. The promise of these scheme is to distribute a secret through quantum shares and limit the probability that two parties, who have gained possession of the necessary shares, could both reconstruct the correct secret. Although promising, this approach is yet vastly unexplored

1.2 Objective

This work stems from Ananth et al. initial research in the field of unclonable secret sharing, [1]. Their work provides the foundations for building secret sharing schemes that prevent the share-receiving parties from creating local clones of their share that can be employed by adversaries for recovering the secret; additionally, it connects USS with other relevant problems in cryptography and quantum computing, such as unclonable encryption and instantaneous non-local computation.
Results are achieved proving possibility or impossibility of both information-theoretic and computational unclonable secret sharing scheme depending on quantum resources available for adversaries and, when possible, a construction of an unclonable secret sharing scheme is provided.
The main goal of this work is to extend the theoretical results of Ananth et al. by providing an implementation of the construction they propose for unclonable secret sharing against adversaries equipped with limited quantum resources, in the form of entanglement.
The first objective is to produce a functional implementation by using IBM's library `qiskit`, that introduces Python-based tools to write code for quantum computers and simulate its execution.
Through this implementation, we aim at experimentally testing whether unclonable secret sharing satifies both the functional requirements of any secret sharing scheme and the unclonability property.
More precisely, our objective is first to verify that the proposed USS primitive satisfies the functional requirements specified in table 1.1. The testing procedure is detailed in section 5; generally, we define the expected behaviour of the program and then validate it through either automated tests or manual tests aided by data analysis.

REQ1	Generate shares correctly
REQ2	Generate uniformly distributed shares
REQ3	Handle uncorrect inputs appropriately
REQ4	The secret is correctly reconstructed from a valid set of shares
REQ5	The reconstruction procedure has consistent outputs for the same inputs
REQ6	Subsets of shares do not leak information about the secret

Tabelle 1.1: Functional requirements of USS

Furthermore, our goal is to test the unclonability property of the proposed construction. To accomplish this purpose we subject our implementation to two attack strategies that might be enacted by adversary parties. These strategies deviate from the boundaries set by the original paper, since they interfere with the scheme's proper functionality through share destruction, but they nonetheless constitute a basis for unclonability testing. Through these experiments, we aim to demonstrate that the probability of two attackers both reconstructing the secret correctly is upper-bounded by $\frac{1}{2}$ plus a negligible term.

1.3 Structure

The work starts from chapter 2, providing an introduction to quantum mechanics and quantum computing. Here the basic concepts of quantum mechanics are presented, such as state representation and measurements, quantum computing is introduced with quantum logic gates and quantum phenomena of high relevancy for cryptography are presented.

Chapter 3 goes over the fundamental notions of secret sharing and an overview of the secret sharing schemes landscape is discussed. Then, unclonable secret sharing is formally defined and results on its theoretical existence and security are presented; finally, a construction of unclonable secret sharing is described.

Chapter 4 is dedicated to discussing the implementation details of USS. It generally addresses the main challenges and paradigms in simulation of quantum computers on classical machines. To follow, it provides an account of the main design decisions taken in constructing a practical implementation of unclonable secret sharing.

Chapter 5 details the validation process of the implementation. It specifies the functional requirements of a secret sharing scheme and the testing strategy to verify that our implementation satisfies them, then it describes adversary configurations that demonstrate the unclonability property of the USS scheme.

2 Fundamentals of quantum computing

Unclonable secret sharing is built from the fundamental concepts of quantum computing, therefore familiarity with the framework of quantum mechanics is needed to understand its implementation and functionality.
This chapter outlines the required concepts, as a preliminary to USS itself. The content is primarily based on [22], which provides a detailed account of quantum information theory and computation.

2.1 Qubits

The fundamental information-carrying unit in quantum computing is the qubit, the quantum counterpart of the bit. A bit is a simple object whose state is either 0 or 1 at any given time, while a qubit can be in a superposition of two basis states and this is formally represented by equation 2.1

$$|v\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{2.1}$$

where $|0\rangle$ and $|1\rangle$ are the so called *computational* basis states.
This formulation of the state of a qubit is in fact equivalent to stating that a qubit is a vector in a two-dimensional complex vector space, whose basis is $\{|0\rangle, |1\rangle\}$. This space is called Hilbert space and denoted by $\mathcal{H}^n$ with n being the number of dimensions.
In general (infinite-dimensional case), a Hilbert space is more than a complex vector space, but a finite-dimensional Hilbert space is equivalent to a complex vector space with inner product [1].
In the classical notation of linear algebra, using column vectors, $|0\rangle$ and $|1\rangle$ represent the canonical basis of $\mathcal{H}^2$, thus

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2.2}$$

[1] [22] pag. 66

and therefore a general qubit of form 2.1 has matrix representation

$$|v\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{2.3}$$

Another fundamental difference between bits and qubits is in the behaviour when measured. Anytime a bit is measured, its state is always known, but when measurement on a qubit is performed the result is 0 with probability $|\alpha|^2$ and 1 with probability $|\beta|^2$, and naturally $|\alpha|^2 + |\beta|^2 = 1$ must always hold true; additionally and importantly, the state of the qubit is changed after the measurement, becoming $|0\rangle$ if the measurement result was 0 and 1 otherwise; this behaviour of qubit, sometimes referred to as *collapse*, makes it impossible to even approximate α and β with multiple measurements on the same qubit.

Quantum computation requires operating on systems of multiple qubits. In linear algebra, the way to combine two vector spaces is the *tensor product* and this works for Hilbert spaces too, therefore a system of two qubits, whose states are vectors respectively in Hilbert spaces $\mathcal{H}_1$ and $\mathcal{H}_2$, will have state space $\mathcal{H}_1 \otimes \mathcal{H}_2$. The elements of $\mathcal{H}_1 \otimes \mathcal{H}_2$ are linear combinations of tensor products of elements $|v\rangle \otimes |w\rangle$, $|v\rangle \in \mathcal{H}_1, |w\rangle \in \mathcal{H}_2$.

Tensor product respects two properties:

- $|v\rangle \otimes (|u\rangle + |w\rangle) = (|v\rangle \otimes |u\rangle) + (|v\rangle \otimes |w\rangle)$
- $\alpha\,|v\rangle \otimes |w\rangle = |v\rangle \otimes \alpha\,|w\rangle = \alpha(|v\rangle \otimes |w\rangle)$

The tensor product of two vectors in $\mathcal{H}^2$ is then:

$$\begin{aligned} |v\rangle \otimes |w\rangle &= (\alpha\,|0\rangle + \beta\,|1\rangle) \otimes (\gamma\,|0\rangle + \delta\,|1\rangle) && (2.4)\\ &= \alpha\gamma\ |0\rangle \otimes |0\rangle + \alpha\delta\ |0\rangle \otimes |1\rangle + \beta\gamma\ |1\rangle \otimes |0\rangle + \beta\delta\ |1\rangle \otimes |1\rangle && (2.5)\\ &= \alpha\gamma\ |00\rangle + \alpha\delta\ |01\rangle + \beta\gamma\ |10\rangle + \beta\delta\ |11\rangle && (2.6) \end{aligned}$$

Where $|a\rangle \otimes |b\rangle$ is written as $|ab\rangle$, as conventional in QM notation. By rewriting the the coefficients in 2.6, the most common representation of the state of a two qubit system follows:

$$|\psi\rangle = \alpha\,|00\rangle + \beta\,|01\rangle + \gamma\,|10\rangle + \delta\,|11\rangle \tag{2.7}$$

As an anticipation of section 2.3, it should be said that, not every two-qubit state can be expressed as a tensor product of two individual qubit states: in this case, the state is said to be *entangled*.

2.2 Quantum gates

The concept of quantum computation is to map a classical problem to a set of quantum operations that create a state that can be measured and produces a result that represents the solution to the problem; similarly to classical logic gates, operations in the quantum computing setting are represented by quantum logic gates.
Mathematically quantum gates are $n \times n$ matrices, with varying dimensions depending on the operation they perform: one-qubit gates are 2×2 matrices, two-qubit gates are 4×4 matrices and so on. There is one constraint, and that is to be linear operators.
Some quantum gates can be conceptually comparable to classical gates, for instance the quantum `NOT` gate. The matrix representation of of `NOT`, also denoted by X, is

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{2.8}$$

recalling 2.1, we can write

$$|v\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{2.9}$$

and then describe the action of `NOT`, with a matrix-vector multiplication

$$\texttt{NOT}(v) = X\,|v\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \beta \\ \alpha \end{pmatrix} \tag{2.10}$$

The gate X is part of a set of four gates, called *Pauli matrices*, that often recur in quantum computation.

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad Y = \begin{pmatrix} 0 & i \\ -i & 0 \end{pmatrix} \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{2.11}$$

Generally, every $n \times n$ matrix is a valid operation on qubits, as long as it satisfies one property: recalling that, for a state in the form 2.1, $|\alpha|^2 + |\beta|^2 = 1$, this has to be preserved by the operation on that qubit; this is true for every operation U that satisfies $UU^\dagger = I$; $U^\dagger$ is called adjoint and it is U transposed and complex-conjugated.
In addition to the Pauli matices, a recurring 1-bit gate in quantum computation is the Hadamard gate, with matrix representation

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2.12}$$

H puts a qubit in either of the computational basis states in superposition of them, specifically by mapping $|0\rangle$ to $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$ and $|1\rangle$ to $|-\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$; $\{|+\rangle, |-\rangle\}$ constitutes an alternative basis for the Hilbert space, known as x-basis. The Hadamard gate is also fundamental in creating entangled qubits, that have many practical applications.

Another commonly employed quantum gate is the controlled not, or `CNOT` gate. `CNOT` is a two-qubit gate that flips the *target* qubit if the *control* qubit is 1 and its matrix representation is

$$\texttt{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.13}$$

The action of `CNOT` can be synthetised in $|a, b\rangle \rightarrow |a, a \bigoplus b\rangle$. By applying an Hadamard

Gate	Notation	Matrix
NOT (Pauli-X)		$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli-Z		$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Hadamard		$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
CNOT (Controlled NOT)		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

Abbildung 2.1: Quantum gates

gate to one qubit, and then a `CNOT` gate with this qubit as control, and another as target, a pair of entangled qubits is created.

Figure 2.1 sums up the matrix representation of the main quantum gates and their corresponding graphical representation in circuits.

2.3 No-cloning theorem and entanglement

One of the strengths of quantum computing is the exploitation of properties of particles that would be indescribable under the assumptions and formalism of classical physics, but that are explained by quantum mechanics: these properties include the impossibility

to clone a qubit and the entanglement between two or more qubits.
One of the fundamentals of quantum mechanics is the no-cloning theorem, expressing the impossibility to copy an arbitrary and unknown quantum state. This section provides an overview of the idea behind principle; a detailed proof can be found in [22][2].
When cloning a qubit, we start with $|v\rangle = \alpha|0\rangle + \beta|1\rangle$ and another qubit that would store the copy of $|v\rangle$, initialiazed as $|0\rangle$: as a two-qubit system is therefore composed, its state can be written as

$$|v\rangle|0\rangle = \alpha|00\rangle + \beta|10\rangle \tag{2.14}$$

If a `CNOT` gate was applied to this state it would result in the following state:

$$\texttt{CNOT}(\alpha|00\rangle + \beta|10\rangle) = \alpha|00\rangle + \beta|11\rangle \tag{2.15}$$

If the qubit to copy was in state $|0\rangle$ or $|1\rangle$, the system would successfully produce two qubits in the same state, because $\texttt{CNOT}|00\rangle = |00\rangle\,(\alpha = 1, \beta = 0)$ and $\texttt{CNOT}|10\rangle = |11\rangle\,(\alpha = 0, \beta = 1)$. Nevertheless, this represents only a special case of qubit cloning: this copying circuit, when given a generic $|v\rangle$ results in $|v\rangle\ |v\rangle$ that, when expanded, equates to

$$|v\rangle|v\rangle = \alpha^2|00\rangle + \alpha\beta|01\rangle + \alpha\beta|10\rangle + \beta^2|11\rangle \tag{2.16}$$

This expression is, in turn, equal to 2.15 only if $ab = 0$: this shows that cloning a qubit is only possible when its state is either $|0\rangle$ or $|1\rangle$.
The other quantum phenomenon that is often exploited in quantum computing is that of entanglement, that is the key to quantum teleportation. entanglement is the property of a two(or more)-qubit system, whose state cannot be written as a tensor product between two one-qubit states. For a more formal definition, let's look at equations 2.5 and 2.4 but reversing our logic. First of all, rewrite 2.5 so that every term only has one coefficient

$$|\phi\rangle = \alpha|0\rangle \otimes |0\rangle + \beta|0\rangle \otimes |1\rangle + \gamma|1\rangle \otimes |0\rangle + \delta|1\rangle \otimes |1\rangle \tag{2.17}$$

It is said that the state $|\phi\rangle$ is not entangled if four constants a, b, c, d exist such that

$$\alpha|0\rangle \otimes |0\rangle + \beta|0\rangle \otimes |1\rangle + \gamma|1\rangle \otimes |0\rangle + \delta|1\rangle \otimes |1\rangle = (a|0\rangle + b|1\rangle) \otimes (c|0\rangle + d|1\rangle) \tag{2.18}$$

This is also called a *separable* state, with the opposite being an *entangled* state.
Entanglement between two qubits has consequences on their behaviour when measured, by creating a correlation between the distinct measurements perfomed on the two qubits,

[2] Page 532

even when physically separated. This correlation is especially observable on the four two-qubits entangled states known as *Bell states*: these are maximally entangled states and this, in turn, implies perfect correlation upon measurement. The four Bell states can be expressed concisely by

$$|\beta_{x,y}\rangle \equiv \frac{|0,y\rangle + (-1)^x |1,\overline{y}\rangle}{\sqrt{2}} \tag{2.19}$$

Entanglement is exploited for the operation known as quantum teleportation, that allows a party(Alice) to teleport an unknown quantum state to another party(Bob) without a quantum communication channel as long as they can pre-share a pair of entangled qubits and a classical communication channel.

Essentially, quantum teleportation of qubit $|\phi\rangle$ is achieved by operating on Alice's two qubits, first with a `CNOT` gate conditioned on $|\phi\rangle$ to be transmitted and then with $H|\phi\rangle$. Then Alice measures her two qubit, obtaining two classical bits a, b and sends them to Bob over the classical channel: at this point Bob has to apply $Z^a X^b$ to his half of the entangled qubits and this will be in state $|\phi\rangle$; note that, while it may seem otherwise, the no-cloning theorem is not infringed because Alice's original qubit is collapsed by the measurement and therefore only one qubit is left in state $|\phi\rangle$.

A more detailed explanation of quantum teleportation is included in the appendix.

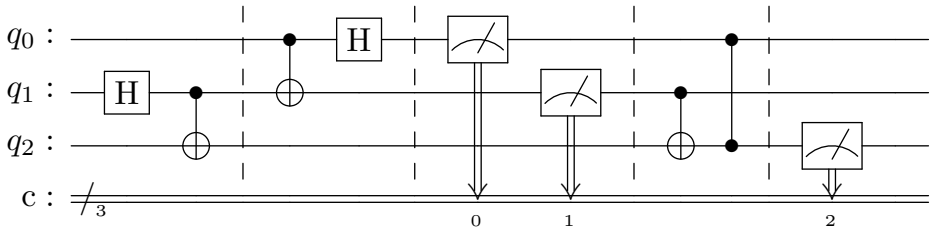

Abbildung 2.2: Complete circuit for quantum teleportation. Qubits q_0, q_1 belong to Alice, q_2 to Bob

2.4 Quantum measurement

In section 2.1, the behavior of measures on a quantum system was briefly described: in this section a more detailed explanation will be provided, as this is of high relevance for quantum cryptography.

As a preliminary, two operations on Hilbert spaces have to be introduced: the *inner* and

the *outer* product.
The inner product between two vectors $|v\rangle, |w\rangle$ in $\mathcal{H}^n$ space is denoted by $\langle v|w\rangle$ and, if $|v\rangle = \{v_1, \ldots, v_n\}^T, |w\rangle = \{w_1, \ldots, w_n\}^T$ with coefficients written with regards to the same basis , the inner product is calculated as

$$\langle v|w\rangle = (w_1^*, \ldots, w_n^*) * \begin{pmatrix} v_1 \\ \ldots \\ v_n \end{pmatrix} \tag{2.20}$$

where w_i^* is the complex conjugate of the i-th element of $|w\rangle$. The result of the inner product is thus a complex number.
For instance, let's define $|+\rangle = H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \end{pmatrix}^T$ and compute $\langle 0|+\rangle$.

$$\begin{pmatrix} 1 & 0 \end{pmatrix} \cdot \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} (1 \cdot 1 + 0 \cdot 1) = \frac{1}{\sqrt{2}} \tag{2.21}$$

The outer product is instead denoted by $|v\rangle\langle w|$, and by rules of matrix multiplication, for two n-dimensional vectors, the result is an $n * n$ matrix whose elements are defined as follows:

$$|v\rangle\langle w| = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix} \begin{pmatrix} w_1^* & w_2^* & \cdots & w_m^* \end{pmatrix} = \begin{pmatrix} v_1 w_1^* & v_1 w_2^* & \cdots & v_1 w_n^* \\ v_2 w_1^* & v_2 w_2^* & \cdots & v_2 w_n^* \\ \vdots & \vdots & \ddots & \vdots \\ v_n w_1^* & v_n w_2^* & \cdots & v_n w_n^* \end{pmatrix} \tag{2.22}$$

To reuse the previous example, it is clear that

$$|0\rangle\langle +| = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & 0 \end{pmatrix} \tag{2.23}$$

These two operations are a key tool in describing the theory of quantum measurements. As anticipated in section 2.1, quantum measurements have an effect on the state of the system and this corresponds to them being described as operators: more precisely a measurement is described by a set of operators M_m where m is the measurement result. The fact that the probability of each different outcome must sum to one is expressed by the condition $\sum_m M_m^\dagger M_m = I$, with $M_m^\dagger$ being the conjugate transpose of M.

The probability of outcome m when measuring qubit ϕ is determined by

$$p(m) = \langle\phi|M_m^\dagger M_m|\phi\rangle \tag{2.24}$$

After measurement the system is left in state

$$\frac{M_m\,|\psi\rangle}{\sqrt{\langle\psi|\,M_m^\dagger M_m\,|\psi\rangle}} \tag{2.25}$$

In the specific case of measuring on the computational basis, a measurement is represented by two operators

$$M_0 = |0\rangle\langle 0| = \begin{pmatrix}1 & 0\\ 0 & 0\end{pmatrix} \quad M_1 = |1\rangle\langle 1| = \begin{pmatrix}0 & 0\\ 0 & 1\end{pmatrix} \tag{2.26}$$

Note that for both $M_m = M_m^\dagger$ and $M_m^2 = M_m$ holds true.
Considering this, it can be made clear why a qubit with state $|\phi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ collapses to $|0\rangle$ with probability $|\alpha|^2$ and similarly for $|1\rangle$.

$$p(0) = \langle\phi|M_0^\dagger M_0|\phi\rangle = \langle\phi|\,|0\rangle\langle 0|\,|\phi\rangle = \langle\phi|0\rangle\langle 0|\phi\rangle = \langle 0|\phi\rangle^2 \overset{2.20}{=} |\alpha|^2 \tag{2.27}$$

Mind that inner product is not *generally* commutative, but it happens to be for $\langle 0|\phi\rangle$; with the same reasoning, it is straightforward to prove that $p(1) = |\beta|^2$. By substituting the appropriate expressions in 2.25, it is also explained why a system is left in state $|0\rangle$ when the measurement result is 0 and equally for $|1\rangle$:

$$\frac{M_0\,|\phi\rangle}{\sqrt{\langle\phi|M_0^\dagger M_0|\phi\rangle}} = \frac{|0\rangle\langle 0|\,|\phi\rangle}{|\alpha|} = \frac{\alpha}{|\alpha|}\,|0\rangle \tag{2.28}$$

The term $\frac{\alpha}{|\alpha|}$ is sometimes referred to as phase and it can be ignored in this case. The corresponding result is trivial to prove for a measurement with result 1, leaving the system in state $|0\rangle$.
For cryptographic purposes, it is especially relevant when vectors in the x-basis($|+\rangle$, $|-\rangle$) are measured in the computational basis: as these vectors are in an equal superposition of $|0\rangle$ and $|1\rangle$, a measurement has an equal probability of resulting in 0 or 1. This property has been the cornerstone first of Wiesner's encoding [29], then of BB84 quantum key distribution [3].

3 Fundamentals of secret sharing

3.1 Notation and preliminaries

Secret Sharing is a cryptographic primitive with one specific aim: that is, a piece of secret data has to be distributed between a set of parties and these parties can only gain knowledge of this secret when a previously-established number of them combines the partial knowledge they received at the start of the process.
This primitive has many real-world applications, some of which are listed next: joint approval of decisions, for instance a board of directors coming together to approve a decision, improving security of IoT networks, electronic voting, safeguarding a private key by distributing it over different physical components and many others.
Many implementations of secret sharing, or secret sharing schemes, exist. Generally, secret sharing is described as a *dealer* distributing a secret s to a set of parties, denoted by $\mathcal{P} = \{P_1, \ldots, P_n\}$ and only subsets of $\mathcal{P}$ who are considered *qualified* can reconstruct s.
On a more formal level, what a scheme does is defining an *access structure* $\Gamma \subseteq 2^{\mathcal{P}}$, where $2^{\mathcal{P}}$ is the set of subsets of $\mathcal{P}$ and Γ represents all the authorized subsets in $\mathcal{P}$; additionally, an access structure is monotone if for $A \in \Gamma$ and , $A' \supseteq \Rightarrow A' \in \Gamma$.
Existence of a secret sharing is subjugated to the monotony of its access structure [20].
A secret sharing scheme for a monotone Γ is summarized as two operations:

- $\mathsf{share}(s) \rightarrow \sigma_1, \ldots, \sigma_n$ that takes the secret s as input and outputs n shares
- $\mathsf{reconstruct}(\{\sigma_i\}) \rightarrow s'$ that takes some shares as input and outputs $s' = s$ if and only if $\{P_i\}$ is an authorized set

Additionally, these two operations need to satisfy two constraints [20]:

- For every secret and every qualified set, $\mathsf{reconstruct}$ correctly retrieves the secret with probability ≥ 1 minus negligible

- The probability of reconstructing a correct secret from an unauthorized set of shares is $\leq \frac{1}{n}$ plus negligible

When an unauthorized group of shares reveals no information at all, it is said that the scheme is *perfectly private*, a notion equivalent to information-theoretic security.
A subset of secret sharing schemes is composed of (k, n)-threshold secret sharing schemes, where not all shares are required for reconstruction, but only $k < n$ are.
To this group belong the first two, independently-developed and contemporary secret sharing schemes: Shamir's secret sharing and Blakley's secret sharing, both relying on two different geometrical intuitions. Blakley's secret is a point in n-dimensional space and the shares are (n-1)-dimensional hyperplanes [4]; Shamir, on the other hand, relies on polynomial curves in two-dimensional space over a finite field of a prime number: considering that a curve of degree $k-1$ is uniquely determined by k value, a random polynomial is selected with the secret being its intersection with the y axis and the shares being n random points on this curve. This intuition makes Shamir's secret sharing perfectly private, as knowledge of $k-1$ does not leak any information on the curve.

3.2 Advancements in secret sharing

Shamir's construction of secret sharing is information-theoretically secure; nonetheless, oftentimes, real-world secret sharing applications require security guarantees that shamir's scheme cannot provide. Think for instance of a setting where the dealer is dishonest and distributes faulty shares or one where the distribution occurs over an insecure network and an attack tampers with the shares, or an attacker trying to corrupt shares over a long time frame to render a secret unaccessible: these are only some of the weaknesses regular SS could be vulnerable to.
Over time, more advanced categories of secret sharing schemes have been designed as an answer to the limitations of the first schemes when it comes to dishonest parties or specific types of attacker: in this section, we will concisely present some of these categories.
Firstly, imagine a situation where an adversary is able to control communications between the parties and a reconstructing user: this attacker could interfere with the shares being transmitted for reconstruction and derail the operations, resulting in a wrong secret being retrieved even by an honest set of parties who believe they sent their non-faulty shares.

It is said that a scheme is *robust* when even if t shares have been corrupted, the secret can be reconstructed correctly as long as all the shares are submitted.
Another adversary configuration could be one where the attacker corrupts communication between the dealer and the parties or the dealer itself is dishonest: this could lead to some parties receiving a faulty share that excludes them from taking part in successful reconstruction operations; as an countermeasure, some schemes allow parties to verify the validity of their shares and are therefore named *verifiable* secret sharing schemes, a notion first introduced by [6].
Secret sharing schemes could also be vulnerable to attackers operating over an extended window of time, trying to gain knowledge of one share after the other rather than instantly intervene during the distribution or the reconstruction phase; clearly, such security is necessary for highly confidential data. Schemes that enable a periodical or upon-request renewal of shares without changing the secret are labeled *proactive*, with the first being designed in [13]. Finally, we introduce a type of scheme that tries to tackle the same vulnerability unclonable secret sharing poses as a countermeasure to, namely the copying of shares.
Imagine Alice has a secret that she wants to protect through secret sharing, so she generates some shares and wants to store them in some third-party storage space: in general, irregardless of the property of the secret sharing scheme employed, copies of the shares can always by kept by the storage providers; this, in turn, allows either the providers to sell their copy to competitors or attackers to obtain each share from every provider.
In [12], traceable secret sharing is introduced as a SS primitive ensuring that, if the storage providers sell their shares to a third party, a proof of this fraud is inevitably produced and therefore they should be deterred from such operation. Nonetheless, this feature offers only limited guarantees as it does not prevent physically copying the shares and if the buying party is not collaborating with Alice, it would be able to reconstruct the secret anyway. It is precisely this problem that USS tries to solve with quantum computing.

3.3 Unclonable Secret Sharing

Similarly to traceable secret sharing, USS aims at overcoming the vulnerability innate to classical secret sharing of the copying of shares, by exploiting quantum computing.
Previously, quantum computing had already been joined with secret sharing but either with the aim of mirroring classical secret sharing for quantum information, [11], [21], or

of ensuring no attacker intervenes during the distribution phase, [14], [19], now, the aim is to split a secret through shares prepared in a a way that prevents external storage providers from locally cloning their shares into two distinct sets, that both allow for reconstruction.
Ananth et al., [1], introduce a general (t, n)-unclonable secret sharing scheme as two operations:

- $\mathsf{Share}(1^\lambda, 1^n, 1^t, m) \to \rho_{R_1 R_2 \ldots R_n}$: n, λ are security parameters, m is the secret bit string and t is the threshold value (ignored for n-out-of-n sharing); this operation outputs registers $R_1, R_2, \ldots, R_n$
- $\mathsf{Reconstruct}(\rho_{R'_{i_1}}, \ldots, \rho_{R'_{i_t}})$: takes t shares $R'_{i_1}, \ldots, R'_{it}$ as input and outputs a secret $\hat{m}$

These operations must allow for correct secret reconstruction with probability 1 minus negligible, formally:

$$\Pr[\text{Reconstruct}(\rho_{i_1}, \ldots, \rho_{i_k}) = m | (p_1, \ldots, p_n) \leftarrow \text{Share}(1^\lambda, 1^\lambda, m) \wedge k \geq t] = 1 - \text{negl}(\lambda).$$

What sets USS apart is the unclonability property, formally expressed in the notion of indistinguishability-based security (short, IND-CPA). To define IND-CPA, [1] introduces a game $\mathsf{Exp}(\{\mathcal{A}_i\}, \mathcal{B}, \mathcal{C})$

$\mathsf{Exp}(\{\mathcal{A}_i\}, \mathcal{B}, \mathcal{C})$

1. ξ is a quantum state over n registers $Aux_1, ..., Aux_n$. $\mathcal{A}_i$ receives register Aux_i for each $i \in [n]$
2. $\mathsf{Adv}= (\{\mathcal{A}_i\}, \mathcal{B}, \mathcal{C}, \xi)$ sends two message m_0 and m_1 of the same length to a challenger
3. The challenger selects a random bit and prepares $(\rho_1, ..., \rho_n) \leftarrow \mathsf{Share}(1^\lambda, 1^n, m_b)$, then send ρ_i to $\mathcal{A}_i$
4. For each $i \in [n]$, $\mathcal{A}_i$ produces a bipartite state $\sigma_{X_i Y_i}$ over registers X_i and Y_i from the share it received, and sends X_i to $\mathcal{B}$ and Y_i to $\mathcal{C}$.
5. B on input the registers $X_1, ..., X_n$, outputs a bit b_g. C on input the registers $Y_1, ..., Y_n$, outputs a bit b_c.

6. Output 1 if $b_g = b$ and $b_c = b$.

Finally, based on this experiment two types of unclonable secret sharing, with different security levels are defined:

- Information-theoretic USS: an n-party unclonable secret sharing has 1-bit unpredictability if
$$Pr\left[1 \leftarrow \mathrm{Expt}_{(\{A_i\},B,C)}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda) \tag{3.1}$$
holds for any non-uniform adversary $\mathrm{Adv} = (\{A_i\}, B, C)$.

- Computational USS: an n-party unclonable secret sharing has 1-bit unpredictability if
$$Pr\left[1 \leftarrow \mathrm{Expt}_{(\{A_i\},B,C)}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda) \tag{3.2}$$
holds for any non-uniform adversary $\mathrm{Adv} = (\{A_i\}, B, C)$ limited to quantum polynomial time resources.

Clearly, the former definition has stronger security than the latter.

3.4 Constructing USS

The construction of a secure unclonable secret sharing is not always possible: specifically, it is dependent on the amount of entanglement shared by the parties receiving the shares. To formalize the adversary entanglement configurations, the concept of *entanglement graph* is introduced.

Definition. *With ρ an n-partite state over quantum registers $X_1, \ldots, X_n$, an entanglement graph $G = (V, E)$ is defined as follows:*

- *G is an undirected graph*
- *$V = \{1, 2, \ldots, n\}$*
- *E contains an edge (u, v) if and only if X_u, X_v are entangled.*

If $P = (P_1, \ldots, P_n)$ are the parties involved in USS, receiving the state $\rho = \rho_1 \otimes \ldots \rho_n$ so that P_i receives ρ_i, it is said that G is the entanglement graph associated with P if G is the graph associated with $(\rho, X_1, \ldots, X_n)$. Finally, we introduce the notation USS_d to represent an unclonable secret sharing scheme that is secure against all adversaries sharing either unlimited or efficient entanglement, whose entaglement graph has at least d connected components.
Before providing a construction for USS in a computational setting, we recount some existence results for some adversary configurations. We omit proofs, to be found in [1].

Theorem. *Information-theoretically USS is impossible for a set of parties P , if G associated with P is connected. In short,* USS_1 *does not exist.*

Theorem. USS_d *exists for every* $d = \omega(\log \lambda)$ *where* λ *is a security parameter*

Finally, a construction for n-out-of-n USS under the conditions of the latter theorem is provided.

1. $\mathsf{Share}(1^\lambda, 1^{(n+1)}, m \in \{0,1\})$
   (a) Sample uniformly random $r_1, \ldots, r_n \in \{0,1\}$ such that $\bigoplus_{i=1}^n r_i = m$
   (b) Sample $\theta_1, \ldots, \theta_n \in \{0,1\}$.
   (c) For each $i \in [n]$, let the i-th share be $\rho_i = H^{\theta_i}|r_i\rangle\langle r_i|H^{\theta_i}$. Let the $(n+1)$-th share be $\rho_{n+1} = (\theta_1, \ldots, \theta_n)$
   (d) Output $(\rho_1, \ldots, \rho_{n+1})$.
2. $\mathsf{Reconstruct}(\rho_1, \ldots, \rho_{n+1})$
   (a) Measure ρ_{n+1} in the computational basis to get $(\theta_1, \ldots, \theta_n)$.
   (b) For every $i \in [n]$, apply H^{θ_i} to ρ_i. Measure the resulting state in the computational basis to get r_i.
   (c) Output $\bigoplus_{i=1}^n r_i = m$.

4 Implementation

4.1 Simulating quantum computers

Programming a quantum computer is all but a trivial task: physical quantum computers(or QPUs) are complex objects, that require strict conditions for execution and that are flawed by noise and qubits losing their quantum properties over time. As cost, complexity and availability severely limit quantum software development, platforms have been produced to allow for simulation of quantum devices on classical machines. This requires working on multiple layers, whose general view is depicted in 4.1.

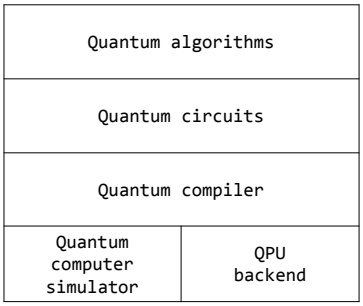

Abbildung 4.1: Quantum computing stack

Generally, quantum programming means specifying a set of quantum operations on qubits, such as logic gates or measurements, to solve a problem; these layers represent the steps necessary to adapt this high-level sequence of instructions to hardware-dependent, executable operations.
At the highest level one finds quantum algorithms and quantum circuits: through a quantum programming language or an integration of another programming language, a programmer codes a sequence of actions or employs quantum subroutines that tackle common problems.
This program is then put through compilation, itself a layered operation. In short, the

three main conceptual issues compilation has to deal with are *decomposition, optimization* and *mapping*, [15]:

Decomposition As said in section 2.2, every $n \times n$ unitary matrix is a valid quantum operation that a programmer can specify; on the other hand, physical QPUs only implement a restricted set of logic gates, such as single-qubit gates and `CNOT`, therefore a compiler has to decompose the sequence of user-specified unitary gates into a sequence of QPU-supported gates.

Optimization Decomposition increases the size and the depth of quantum circuits: this represents a problem, because physical qubits are unstable objects, and over time they lose their quantum properties, [16]. Optimization strategies aim at reducing circuit depth and size with different strategies, such as ancillary qubits, to compensate for short qubit lifetime.

Mapping In addition to specific sets of supported elementary gates, QPUs also often have device-dependent wiring between qubits and not every qubit is connected to another arbitrary qubit; an example is depicted in figure 4.2. The mapping phase creates correspondence between logical qubits and elementary gates from the previous phases to physical qubits and wires contained in the QPU.

Finally, at the lowest level of the quantum computing stack one finds QPUs and simulators. QPUs are ill-suited for first testing quantum software, as they make locating errors harder because of noise, [24]; quantum simulators are therefore ideal substitutes for testing in the ideal setting but not exclusively, as many also allow for simulation of noise too.
There are different approaches to designing simulators, described in-depth by Huang et al. in [15] and Young, Scese and Ebnenasir in [30]. Most simulators are based on the Schrödinger's model: the intuition behind this model is that, considering that a n-qubit state can be written as the sum of 2^n products between a complex coefficient and a basis state, a classical computer can straightforwardly represent quantum states as vectors of coefficients and operations as matrices. Clearly, this approach scales exponentially memory-wise, and simulating around 50 qubits becomes a task beyond most classical supercomputers, [15].

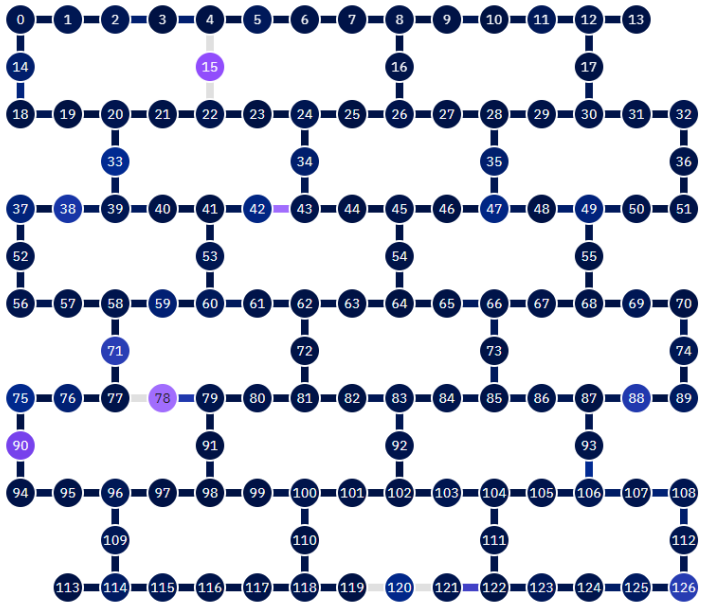

Abbildung 4.2: Wiring of IBM's QPU Quebec. From https://quantum.ibm.com/services/resources?system=ibm_quebec, retrieved December 2024

4.2 Quantum computing framework

There are many tools to be considered when approaching quantum computing platforms. Since not every framework covers all the layers depicted in figure 4.1, we only considered "full-stack"platforms to avoid integration issues between the key components.
Additionally, as quantum computing is an ever-evolving field, it is necessary to rely on ecosystems that are up-to-date, well-maintained and reliable. Fingerhuth et al., [8] provide a detailed analysis of the open-source quantum computing frameworks, both full-stack and non, that have received active maintenance and updates as of 2018; although the field has evolved, the then most established platforms have remained the go-to solution. Of all the ecosystems analysed in [8], for accessibility and ease of use reasons, we discarded "pure"quantum programming languages in favour of Python libraries that allow for quantum computation without requiring familiarity with new syntaxes; mention-worthy exclusions due to this criteria would be ETH Zurich's Silq and Microsoft's Q# (although Q# code can be embedded in Python code).
According to this criteria, the main options were:

- Google's Cirq, [7]

- IBM's Qiskit, [18]
- ETH Zurich's ProjectQ, [27]
- Rigetti's pyQuil, [26]

These frameworks share the same underlying computational paradigm, the discrete gate model as defined by Fingerhuth, that is a generalization of classical computing with qubits and quantum gates (or unitary operations) instead of bits and logic gates. Functionality-wise, all these ecosystems have a similar offering, all the way from Python libraries to a local quantum computer simulator and access to a real quantum machine through cloud services.
Out of all these solutions, ProjectQ adopts an interesting syntax, as seen in 4.1, closely resembling the mathematical formalism of quantum mechanics, but we discarded it mainly for lack of recent updates, with the last release dating back to October 2022 [1] and a reduced documentation when compared to its competitors. Additionally, while ProjectQ grants access to physical backends from other providers, it does not include a proprietary physical quantum computer.
Even though our approach to implementing USS was oriented at a high-level, QPU-indepedent look, this aspect needs to be explored for scaling this application or for a more detailed look at its physical implementation; for these reason, we deemed Cirq and Qiskit better options than pyQuil as they provide integration with many backends from different providers, Cirq with native support[2] and Qiskit with community-maintained packages[3], in addition to respectively Google's and IBM's quantum computers; both also offer access to Microsoft's Azure quantum ecosystem, in turn granting access to multiple backends from diverse providers.
Both Qiskit and Cirq offer local simulators: the Qiskit ecosystem either through the runtime environment that makes fake providers emulating IBM quantum computers available for simulation or through the IBM-maintained, external package `qiskit_aer`, while Cirq either provides its own simulator or is compatible with `qsim`, a C++ based quantum simulator, through a specialized interface. [17] provides a detailed analysis of many simulators performance, including the previously mentioned ones.
For simulating unclonable secret sharing, we decided to rely on Qiskit. Both platforms offer comparable tools, with Cirq having a small edge when it comes to simulation thanks to `qsim`, as [17] shows, even if Qiskit's simulator also has good performance for large

[1] https://github.com/ProjectQ-Framework/ProjectQ/releases [Last visited 05/12/2024]
[2] https://quantumai.google/cirq/hardware
[3] https://www.ibm.com/quantum/ecosystem?tag=Compute+provider

number of qubits. The deciding factors in the decision were quality and availability of documentation and tutorials, a criterion in which we found Qiskit to be superior to its competitor, and also the vast array of visualization tools available with qiskit.

```
from projectq import MainEngine
from projectq.ops import H, Measure

engine = MainEngine()

qubit = engine.allocate_qubit()

H | qubit

Measure | qubit

engine.flush()

result = int(qubit[0])
print("ProjectQ Result:", result)
```

Listing 4.1: ProjectQ

```
import cirq

qubit = cirq.LineQubit(0)

circuit = cirq.Circuit()
circuit.append(cirq.H(qubit))
circuit.append(cirq.M(qubit)

simulator = cirq.Simulator()
result = simulator.run(circuit,
    repetitions=1)



print("Result:", result)
```

Listing 4.2: Cirq

```
from qiskit import QuantumCircuit
from qiskit_aer import Aer


qc = QuantumCircuit(1)


qc.h(0)


qc.measure(0, 0)

backend = Aer.get_backend('qasm_simulator
    ')

result = execute(qc, simulator, shots=1).
    result().get_counts()
```

Listing 4.3: Qiskit

```
from pyquil import Program
from pyquil.gates import H, MEASURE
from pyquil.simulation import
    ReferenceSimulator

p = Program()

ro = p.declare('ro', memory_type='BIT',
    memory_size=1)

p += H(0)
p += MEASURE(0, ro[0])

sim = ReferenceSimulator()
result = sim.run(p, repetitions=1)

print("Result:", result)
```

Listing 4.4: pyQuil

4.3 Components of a Qiskit program

Implementing and simulating USS requires a relatively limited set of quantum gates and operations. In this section we give an overview of the qiskit tools employed in the implementation.
The core object in the Qiskit ecosystem is the `QuantumCircuit`, that abstracts a quantum computer and encompasses quantum and classical memory, in the form of `QuantumRegister` and `ClassicalRegister` objects, respectively storing `Qubit` or `Clbit` objects. Listing 4.5 showcases the different initialization methods for circuits and registers; by default, all qubits in circuits and registers are initiliazed to $|0\rangle$.

```
from qiskit import QuantumCircuit, QuantumRegister,
   ClassicalRegister
from qiskit.circuit import Qubit, Clbit

qc = QuantumCircuit(2) # Create a quantum circuit with 2 qubits
qc1 = QuantumCircuit(2, 2) # Create a quantum circuit with 2
   qubits and 2 classical bits
qc2 = QuantumCircuit(2, 2, name='qc2') # Create a named quantum
    circuit with 2 qubits and 2 classical bits

qreg = QuantumRegister(2) # Create a quantum register with 2
   qubits
qreg1 = QuantumRegister(2, name='qreg1') # Create a named
   quantum register with 2 qubits
qreg2 = QuantumRegister(Qubit() for _ in range(2)) #create
   quantum register from iterable of qubits

creg = ClassicalRegister(2) # Create a classical register with
   2 bits
creg1 = ClassicalRegister(2, name='creg1') # Create a named
   classical register with 2 bits
creg2 = ClassicalRegister(Clbit() for _ in range(2)) #create
   classical register from iterable of clbits
```

```
qc3 = QuantumCircuit(qreg, creg) # Create a quantum circuit
   from quantum register and a classical register

qc4 = QuantumCircuit()
qc4.add_register(qreg, qreg1, creg, creg1) # Dinamically add
   quantum and classical registers to a quantum circuit
```

Listing 4.5: Initialization of ciruits and registers

Specifying a general operation on a circuit is achieved by employing the `append(instruction)` method (or other function that combine different circuits); through the `Instruction` interface, users can define their own operations. In addition to the general method, qiskit provides a range of methods to concisely append the most common operations to a circuit.

```
from qiskit import QuantumCircuit, QuantumRegister,
   ClassicalRegister
from qiskit.circuit.library import HGate, CXGate

qc = QuantumCircuit(2)

# Append an H gate to the first qubit using the append method
h_gate = HGate()
qc.append(h_gate, 0)
qc.h(0)

# Append a CX (CNOT) gate to the circuit using the append
   method
cx_gate = CXGate()
qc.append(cx_gate, [0, 1])
qc.cx(0, 1)

#Pauli matrices
qc.x(0)
qc.z(0)
qc.y(0)
```

```
qc.i(0)
```

Measurement of qubits in qiskit is performed through three method calls on a circuit, `measure(qubit, cbit)` that measures a specified qubit or quantum register into a selected classical bit or register, `measure_all()` that measures all qubits in the circuit and, unless otherwise specified, adds a classical register to save the measurement result and finally `measure_active()` that behaves similarly to `measure_all()` but only measures qubits involved in any operation.

```
from qiskit import QuantumCircuit, QuantumRegister,
   ClassicalRegister

qreg, creg = QuantumRegister(2), ClassicalRegister(2)
qc = QuantumCircuit(qreg, creg)

qc.measure(qreg, creg) # Measure all qubits in qreg and store
   the result in creg

qc1 = QuantumCircuit(2, 2)
qc1.measure_all() # Measure all qubits in the circuit and store
    the result in classical bits

qc2 = QuantumCircuit(3, 3)
qc2.cx(0, 1)
qc2.measure_active() # Measure all active qubits (here 0 and 1)
    and store the result in classical bits
```

Listing 4.6: Measurements in qiskit

To run a circuit and observe the execution result, it is necessary to specify either a simulator or a QPUs to execute the circuit on. Most of our simulations were run on the qiskit Aer simulator[4], provided by the external but IBM-maintained `qiskit_aer` package. A circuit can be run by calling the `run` method on a backend object; among other parameters, this method allows to fix how many times the circuit should be executed (shots). Running a circuit on a backend returns a `Result` object, that encapsulates execution

[4] https://qiskit.github.io/qiskit-aer/

data. To fetch the result of measuring qubits, the `Result.get_counts()` has to be called and it returns a dictionary whose keys are binary strings that represent the classical result of the measurement in little-endian order, and values represent how many times this result occurred. For instance, running the code in listing 4.7 once, results in output `{'0': 529, '1': 495}`.

```
from qiskit import QuantumCircuit
from qiskit_aer import Aer

qc = QuantumCircuit(1, 1)

qc.h(0)

qc.measure(0,0)
backend = Aer.get_backend('qasm_simulator')
res = backend.run(qc, shots=1024).result()
counts = res.get_counts()
```

Listing 4.7: Executing a circuit

4.4 Design

As discussed in section 3.4, unclonable secret sharing can only exist against adversaries limited in quantum resources; so far, only a construction for USS against adversaries who share polynomial entanglement, denoted by $\mathsf{USS}_{\omega \log(\lambda)}$ has been provided.
Implementing this construction requires adapting theoretical quantum operations to the possibilities offered by a concrete quantum programming framework. This imposes some limitations to the accuracy of the simulation, in this case mostly to representation of multi-party quantum communications.
We aim at replicating the basic functionality of a secret sharing scheme, therefore we directly implement two functions, share() and reconstruct(), and additional utility functions.
This section documents the main design decisions that guided the development of the implemented $\mathsf{USS}_{\omega \log(\lambda)}$ scheme.

The validation of the implementation through appropriate tests is discussed in the following section, while the folder structure of the project is explained in appendix A.2.

4.4.1 Share phase

We recall the construction of the sharing phase for an $\mathsf{USS}_{\omega \log(\lambda)}$ scheme:

$\mathsf{Share}(1^\lambda, 1^{(n+1)}, m \in \{0,1\})$:

(a) Sample uniformly random $r_1, \ldots, r_n \in \{0,1\}$ such that $\bigoplus_{i=1}^{n} r_i = m$

(b) Sample $\theta_1, \ldots, \theta_n \in \{0,1\}$.

(c) For each $i \in [n]$, let the i-th share be $\rho_i = H^{\theta_i}|r_i\rangle\langle r_i|H^{\theta_i}$. Let the $(n+1)$-th share be $\rho_{n+1} = (\theta_1, \ldots, \theta_n)$

(d) Output $(\rho_1, \ldots, \rho_{n+1})$.

We distinguish two phases of shares' generation: sampling of random bits for r_i and θ_i and initialization of quantum states depending on the combined values of r_i, θ_i. r_i is the bitstring that actually encodes the secret, that is the `XOR` of all its bits, also known as *parity* of a bit string.
Subsequently, all quantum shares but ρ_n are created according to equation

$$\rho_i = H^{\theta_i} |r_i\rangle \langle r_i| H^{\theta_i} \tag{4.1}$$

This equation represents a quantum state as a matrix, rather than a vector, according to a formalism known as *densitymatrix*.
Since, the effect of a gate U on a density matrix σ, leaving the system in state ρ' is expressed as

$$\sigma' = U\sigma U^\dagger \tag{4.2}$$

it follows that equation 4.1 is an alternative formulation of an application of the H gate to a quantum state σ. As in this protocol, $r_i = \{0,1\}$, it follows that either $\sigma = |0\rangle\langle 0|$ or $\sigma = |1\rangle\langle 1|$, that are respectively the density matrix representations of $|0\rangle$ and $|1\rangle$. Therefore, equation 4.1 means that, to construct any i-th share other than ρ_n, an Hadamard gate is applied to $|0\rangle$ or $|1\rangle$, conditionally on the value of θ_i.
Considering that every qubit in the shares-generating circuit is initiated in state $|0\rangle$, table 4.1, provides a summary of state preparation operations for shares $(\rho_i, \ldots, \rho_{n-1})$.

r_i	θ_i	State vector	Label	Preparation from $\|0\rangle$
0	0	$\begin{pmatrix} 1 \\ 0 \end{pmatrix}$	$\|0\rangle$	
1	0	$\begin{pmatrix} 0 \\ 1 \end{pmatrix}$	$\|1\rangle$	$X\|0\rangle$
0	1	$\begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$	$\|+\rangle$	$H\|0\rangle$
1	1	$\begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix}$	$\|-\rangle$	$HX\|0\rangle$

Tabelle 4.1: Share preparation operations

Preparing ρ_n is straightforward: starting from $n-1$ qubits in state $|0\rangle$, an X is applied to qubit i if $\theta_i = 1$.

The activity diagram in figure 4.3 represents the sequence of operation to be performed during the sharing phase. In a practical, QPU-based implementation, these operations would produce a set of single-qubit quantum states, representing $\rho_1, \ldots, \rho_{n-1}$ and an $(n-1)$-qubit state for ρ_n: realistically, these states would then be transmitted to different physical quantum devices controlled by the recipients.

Multi-device modelling is beyond the functionalities of `qiskit`: operations and memory, either quantum or classical, have to be specified inside a single `QuantumCircuit`, that constitutes the main abstraction of a quantum computer. Clearly, this prevents 1:1 simulation of the communication between quantum devices.

To address this constraint, the implementation stores the generated shares in two separate quantum registers within the same circuit. One register represents shares $\rho_1, \ldots, \rho_{n-1}$, while the other stores ρ_n. As a result, the share() function outputs a `QuantumCircuit` object containing these two explicitly named quantum registers of $(n-1)$-qubits. It is critical that registers are named, as this enables the reconstruct() function to identify and process the appropriate registers during the reconstruction phase.

In addition to the base specifications provided by [1], the share() function has been equipped with additional parameters that act as input values for r and t: this facilitates testing and allows users to employ the RNG they deem most secure.

4.4.2 Reconstruction phase

The reconstruction function, as proposed in [1] behaves as follows:

Reconstruct$(\rho_1, \ldots, \rho_{n+1})$

(a) Measure ρ_{n+1} in the computational basis to get $(\theta_1, \ldots, \theta_n)$.

(b) For every $i \in [n]$, apply H^{θ_i} to ρ_i. Measure the resulting state in the computational basis to get r_i.

(c) Output $\bigoplus_{i=1}^{n} r_i = m$.

In our implementation, this function has to be adjusted in order to match the return type of the sharing function; for this reason, it receives as input a `QuantumCircuit` object, rather than n separate shares, and two strings representing the names of the shareholding registers, through which it can access the shares.
The main task in this function is the conditional application of Hadamard gates to the shares: this step is necessary because measuring $|+\rangle$ or $|-\rangle$ in the computational basis can result in either 0 or 1, therefore the result might not mirror r_i as generated in share();on the other hand, after applying an Hadamard gate to $|+\rangle$ and $|-\rangle$, they are respectively left in state $|0\rangle$ and $|1\rangle$ and, when measured in the computational basis, they will produce an output that correlates perfectly to r_i.
The conditional application of quantum gates, based on measurement result is natively supported by `qiskit` with the `c_if()` instruction, whose use is illustrated in listing 4.8

```
from qiskit import QuantumCircuit, QuantumRegister,
   ClassicalRegister

creg = ClassicalRegister(1)
qreg1 = QuantumRegister(1)
qreg2 = QuantumRegister(1)
circuit = QuantumCircuit(qreg1, qreg2, creg)

#Store measurement result of qreg1 in creg
circuit.measure(qreg1, creg)

#apply an X gate if classical bit creg[0] is 1
```

```
circuit.x(qreg2[0]).c_if(creg[0], 1)
```

Listing 4.8: `c_if` instruction

After measuring the shares in the computational basis, the parity of the resulting bitstring is measured and returned.
The flow of operations for the reconstruction phase is detailed in figure 4.4.

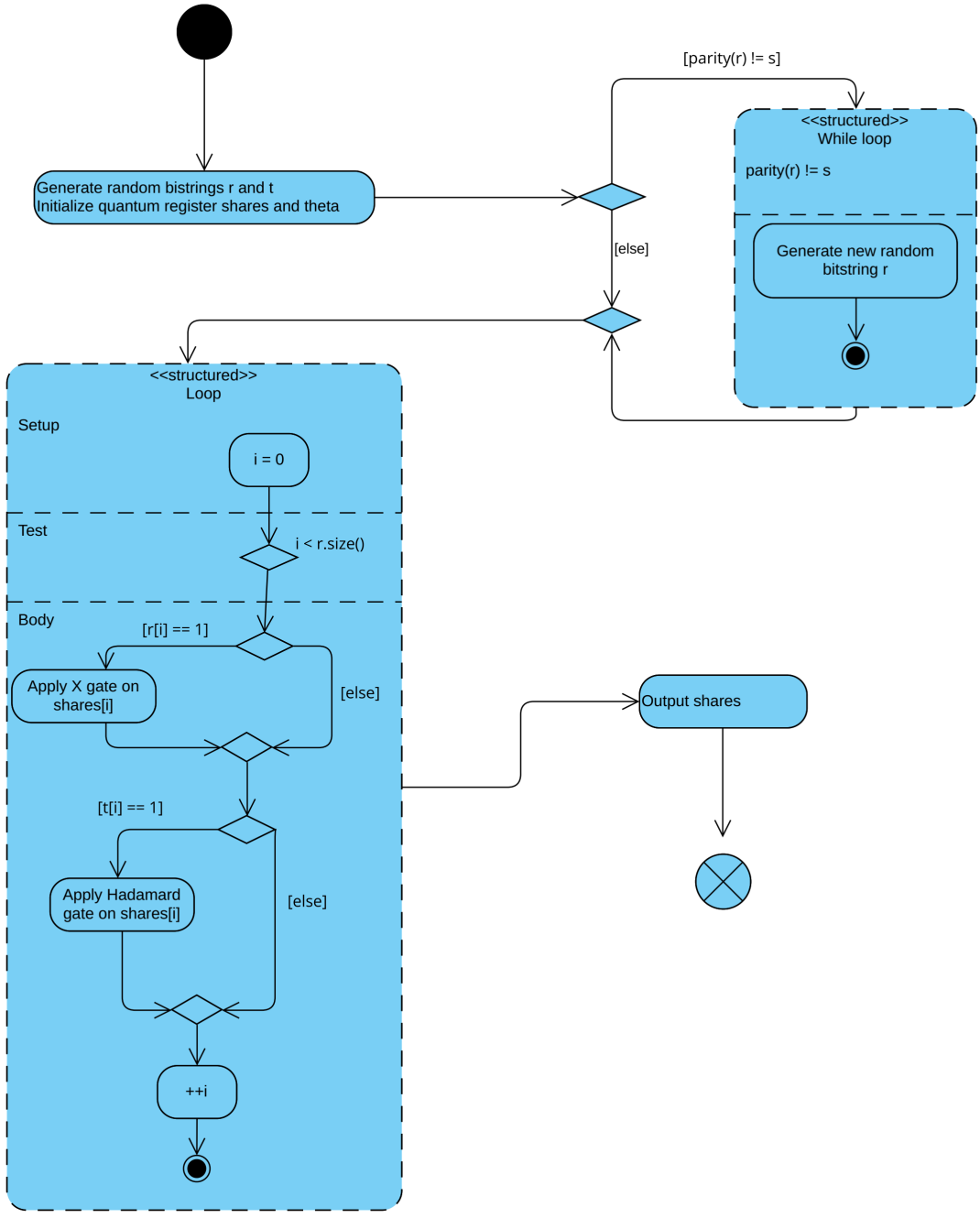

Abbildung 4.3: Activity diagram of the sharing phase

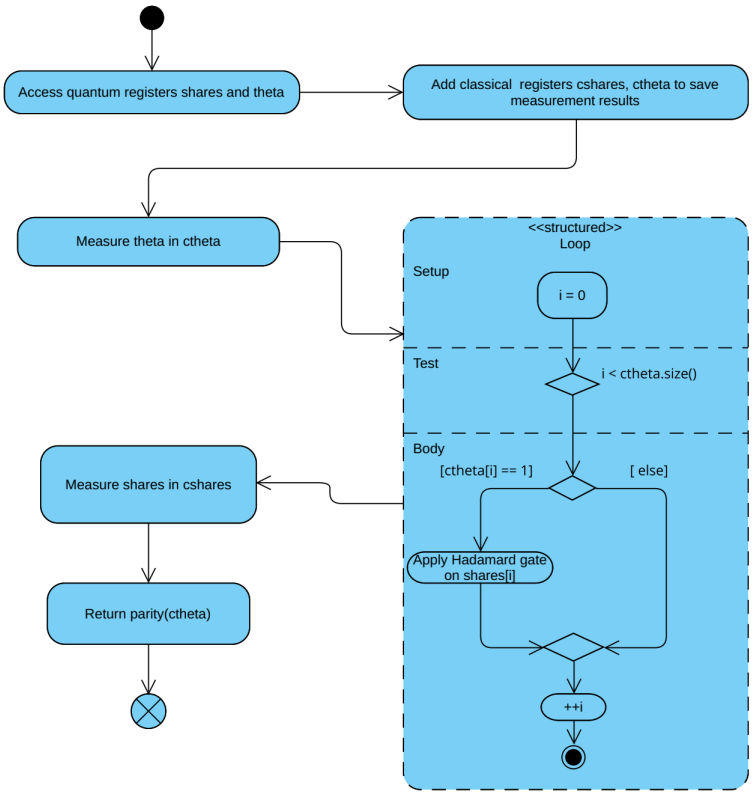

Abbildung 4.4: Activity diagram of the reconstruction phase

5 Testing

This chapter explains the testing strategy and methodology for the implementation of $\mathsf{USS}_{\omega \log(\lambda)}$. The testing phase aims at verifying mainly two aspects of the implementation:

- Validity as a general secret sharing primitive
- Unclonability property against specific adversary strategies

A white-box testing approach is employed: every component of the implementation is tested independently, with both positive test cases to ensure its correct functionality under valid inputs, and negative test cases to confirm appropriate handling of errors and edge cases.
Testing is conducted both through automated tools such as Python's built-in library `unittest` and through manual test cases, designed to verify more complicated scenarios or specific behaviors. Automated tests are collected inside the project folder, inside the `tests` folder, logically grouped by the component that they verify.

5.1 Functionality testing

The first branch of tests aims at ensuring that our implementation of unclonable secret sharing acts as a valid secret sharing primitive; these tests confirm whether the scheme satisfies the functional requirements of secret sharing schemes. An overview of the functional requirements that our implementation must fulfill is provided in table 5.1. The main testing methodology in this phase consists of automated tests, aided by data analysis performed on the execution data of repeated iterations of the sharing phase.

Function	Requirements
Share	Generate shares correctly (REQ1)
	Generate uniformly distributed shares. (REQ2)
	Handle uncorrect inputs appropriately (REQ3)
Reconstruct	The secret is correctly reconstructed from a valid set of shares (REQ4)
	The reconstruction procedure has consistent outputs for the same inputs (REQ5)
	Subsets of shares do not leak information about the secret (REQ6)

Tabelle 5.1: Functional Requirements of Share and Reconstruct Functions in Secret Sharing Schemes

5.1.1 Shares generation

In our model of n-out-of-n USS for bounded adversaries, shares take up the form of quantum states. We distinguish between two type of shares: shares ρ_1 to ρ_{n-1}, that assume one of the states in $\{|0\rangle, |1\rangle, |+\rangle, |-\rangle\}$, and share ρ_n, that is in state $|t\rangle$, with $t \in \{0,1\}^n$. We recall here that the shares generation procedure is dependent on two input bit strings r and t of length $n-1$, randomly sampled with the condition $\mathsf{parity}(\mathsf{r}) = secret$. Consequently, there are different correctness requirements for the two types. For the former type it is required that:

- The correct amount of shares is produced
- The correctly encode the condition $\mathsf{parity}(\mathsf{r}) = secret$
- They are uniformly distributed, that is, given a bit, every bit string r respecting the parity condition is equally probable.

For the latter type, the requirements are:

- It is an n-qubit state
- It is a computational basis state, that is, when measured in the computational basis it collapses to a single basis state $|t\rangle$

RNG testing First, we test that shares are sampled in a uniformly random fashion. Quantum shares are a product of the two bistrings r and t, through the operations described at section 4.4.1; we sample r and t with the `secrets` module, [23], a library providing cryptographically secure random numbers generation.
To verify that this generator satisfies the security standards set for RNG, we subjected it to the test suite proposed by NIST in [2].
A combination of the official NIST implementation of the suite [1] and community-developed alternatives[2] was employed. The testing data was a sequence of 10^6 bits generated with the function `secrets.randbits()`, stored in both string and binary format in files `random_bits.txt` and `random_bits.bin`. Mostly, we relied on the NIST implementation result, but as it presents some shortcomings with usability and accessibility, that prevented us from executing all the tests, we integrated results obtained with the community implementation.

Test Name	Proportion	Result
NIST		
Frequency	99/100	Passed
Block frequency	100/100	Passed
Cumulative sums	99/100 (average)	Passed
Runs	98/100	Passed
Longest run of 1s in a block	99/100	Passed
Binary matrix rank	100/100	Passed
Discrete Fourier transform	97/100	Passed
Non overlapping template matching	98/100 (average)	Passed
Overlapping template matching	100/100	Passed
Serial	99.5/100 (average)	Passed
Linear complexity	97/100	Passed
Community		
Maurer's universal test	1/1	Passed
Random excursions	1/1)	Passed
Random excursions variant	1/1	Passed

Tabelle 5.2: Test result of `secrets.randbits()`

Table 5.2 presents the testing results. While the community implementation acts on the bit sequence as a whole, the NIST implementation splits the bitstream into 100 subse-

[1] https://csrc.nist.gov/Projects/random-bit-generation/Documentation-and-Software
[2] https://github.com/dj-on-github/sp800_22_tests

quences of 10000 bits each, and applies the test on each subsequence: the second column therefore shows the proportion of subsequences that passed the test. The minimum pass rate for each statistical test is approximately 96 for a sample size of 100 subsequences. Some tests, for instance the non overlapping template matching test, looks for a predefined string in the stream: as different strings are looked for, results for such tests are grouped as the average proportion of passing sequences.
The detailed test results are found in files `finalAnalysisReport.txt` and `nist-community-result.txt`.

Correctness of shares generation To verify that shares are correctly generated, two levels of tests are established.
First, the `QuantumCircuit` object returned by share() is inspected, ensuring that it contains two `QuantumRegisters` of appropriate length ($n-1$, with n being the number of parties) and that both these registers have been named. To ensure these requirements are met, the share() is executed for different combinations of `secrets` and `num_parties`, and the returned object is checked against the expected parameters of itself and its contained `QuantumRegisters`. The second level of tests consists in verifying that quantum shares correctly encode the classical information of bit strings r and t.
For share ρ_n it suffices to ensure that it is a computational basis state corresponding to $|t\rangle$. Operationally, this equates to satisfying two conditions: when measured, this state always produces the same result and this result must be equal to t. The verification of this property is carried out through the following operations:

1. Execute share() for every possible bit string t of length 8
2. Measure the quantum state produced in register `theta` from the previous step. `qiskit` natively allows to execute a circuit multiple times in the same configuration: advantage of this tool is taken and this measurement is performed 1000 times
3. Check that every measurement produced the same result and this result equals t

On the other hand, verifying that shares ρ_1 to ρ_{n-1} are correctly generated would require reversing the generation procedure; the testing procedure resembles the operation to verify the correctness of generating ρ_n:

1. Execute share() for every possible pair of bit string r, t, both of length 8

2. Apply an H gate to the i-th qubit in the quantum register containing the shares if $t[i] == 1$
3. Measure the register
4. Verify that each measurement leads to the same result and that this results matches r

5.1.2 Error Handling

The method definition of share() is

```
def share(secret: bool, num_parties: int, r : Bits = None, t :
    Bits= None)
```

with some omitted additional arguments for debugging and visualization. The secret is annotated as a boolean type to highlight that it is a bit.
Based on this parameter, the expected behaviour of the function under unvalid inputs is defined as:

- Return an error if `num_parties` < 2
- Return an error if the size of `r` and `t` mismatch `num_parties`
- Prevent function calls with a secret that is not of type `bool`. As this is accomplished with explicit type checking, an error is returned even for values that can be coerced to `bool`; this also prevents calls such as

  ```
  share(0, 5)      #0 is coerced to False
  share('False', 5)   #'False' is coerced to False
  share("", 5)     # empty string is coerced to False
  ```

On the other hand, *reconstruct*() requires that the circuit representing the shares contains at least two named quantum registers, whose name is also passed to the function. An error has to be raised when any of these components is missing.

5.1.3 Reconstruction correctness and consistency

Correctness The operational correctness of the reconstruction procedure has to be verified: this means checking that, on input the shares saved in a QuantumCircuit by share(secret_bit), reconstruct outputs the correct secret bit.
The concept of the test is to execute the reconstruction procedure for every input configuration, therefore for every pair of n-bit strings, from which shares are generated, and compare the reconstructed secret with the original. The number of qubits to be simulated for n parties USS is $(n-1) \cdot 2$ and since the resources needed for simulation grow exponentially with the number of qubits, the correctness property is tested exclusively for $n = 8$.

Consistency A basic requirement of secret sharing schemes is that its reconstruction procedure be consistent, that is, a definite set of shares is always reconstructed into the same secret.
To verify this property, a similar approach to correctness testing is employed: the reconstruction process is repeated multiple times for the same input and its output is compared to the expected value; as this repetition increases the simulation workload, the number of parties is limited to 5 to reduce the size of the input space. For each input, the reconstruction procedure is executed 50 times.

5.1.4 Automated tests results

The previous tests are automated through Python's `unittest` framework, that allows to define classes encapsulating different test cases as functions. Two classes are defined, respectively testing share() and reconstruct():

TestShare : includes test cases for share

- Invalid input handling: tests that errors are raised when the number of parties is less than 2 (`test_share_valid_n`), when the secret is not interpretable as a boolean (`test_share_secret_type`), or when the provided bitstrings `r` or `t` are of incompatible length with respect to the number of parties (`test_share_r_wrong_length`, `test_share_t_wrong_length`).

- Quantum registers generation: verifies that the correct number of quantum registers with proper names are generated based on the number of parties (`test_registers_generation`).
- ρ_n correctness: ensures that the `theta` register matches the provided bitstring `t` for both secret values of `True` and `False`, by implementing the defined procedure (`test_theta_generation_1`, `test_theta_generation_0`).
- $\rho_1, ..., \rho_n$ correctness: confirms the correctness of the generated `shares` register for even and odd parity cases, ensuring it matches the provided bitstring `r` according to the defined strategy (`test_shares_generation_even`, `test_shares_generation_odd`).

TestReconstruct : includes test cases for reconstruct()

- Invalid input handling: tests that errors are raised when the input circuit does not contain the correct amount of registers or the names are not provided (`test_invalid_input`)
- Correctness of reconstruction: checks that the reconstructed value matches the expected secret for bitstring pairs with parity 0 or 1 (`test_reconstruct_correctness_1`, `test_reconstruct_correctness_0`).
- Consistency of reconstruction: repeats reconstruction multiple times for various bitstring pairs to confirm the results remain consistent and correct(`test_consistency`).

When executed, all the test cases succeed, therefore attesting that the implementation satifies the appointed requirements.

5.1.5 Information leak

In addition to being correct and consistent, a reconstruction function also has to be secure, where by secure it is meant:

- Possessing a subset of authorized shares, does not reduce the dimension of the space of possible secrets

- An unauthorized set of shares does not reveal anything about the secret

Subsets of shares We reason about information gain with subsets of shares first. Given $r, t \in \{0,1\}^{(n-1)}$, the complete set of shares consists of n quantum states: $\{\rho_i\}_{1 \leqslant i \leqslant n-1}$, that can be in either of the four states $\{|0\rangle, |1\rangle, |+\rangle, |-\rangle\}$ and $\theta = |t\rangle$. We distinguish between two types of shares' subsets: either the attacker possesses θ and some of ρ_i, or it only possesses some or all of ρ_i.
We present tests for the latter type first. By the attack definition, an adversary who does not own θ only possesses some shares ρ_i; without loss of generality, it can be imagined that it possesses all ρ_i.
It is known that ρ_i are either computational basis states or x-basis states ($|+\rangle$ and $|-\rangle$): if they were to be measured without any prior operations, due to the nature of quantum measurement, computational basis states would result in a bit β_i equal to r_i with probability 1, meanwhile x-basis states would result in $b_i = r_i$ with probability $\frac{1}{2}$ and $b_i \neq r_i$ with probability $\frac{1}{2}$. Therefore, the bit string representing measurement of ρ_i will have parity equal to the secret if each and every one of the possessed shares is in a computational basis state, otherwise it will have parity= 0 with probability $\frac{1}{2}$ and parity= 1 with probability $\frac{1}{2}$. This sets the reference value for our tests: clearly, some reconstructions with incorrect θ will still result in the correct secret, but over large numbers of iterations this is not expected to happen over 50% of the time.
The testing strategy is discussed below. We abstain from implementation details, providing only the concept.

1. Fix random $r, t = \{0,1\}^{(n-1)}$. This uniquely determines a secret as $parity(r)$.
2. Execute share$(secret, n, r, t)$. This simulates creating quantum states $\{\rho_i\}$ and θ; subsequently execute reconstruct() on the outputs of the sharing phase. This step acts as reference of a reconstruction from a party holding all shares.
3. Sample random $t' = \{0,1\}^{(n-1)}$ according to some strategy, to fill in for t.
4. Execute the reconstruction process with inputs $\{rho_i\}$ and $|t'\rangle$, to simulate the attacker reconstruction

Two strategies were designed: one initializes t' as a bit string of 0s appropriate length, therefore $\theta = |0\rangle^{\otimes n}$ (Naive strategy), the other randomly initializes t' (Random strategy). For each strategy, we create a dataset consisting of 70000 entries (10000 for each possible value of n) and each entry stores the secret, r, t, t' and the secret reconstructed from r

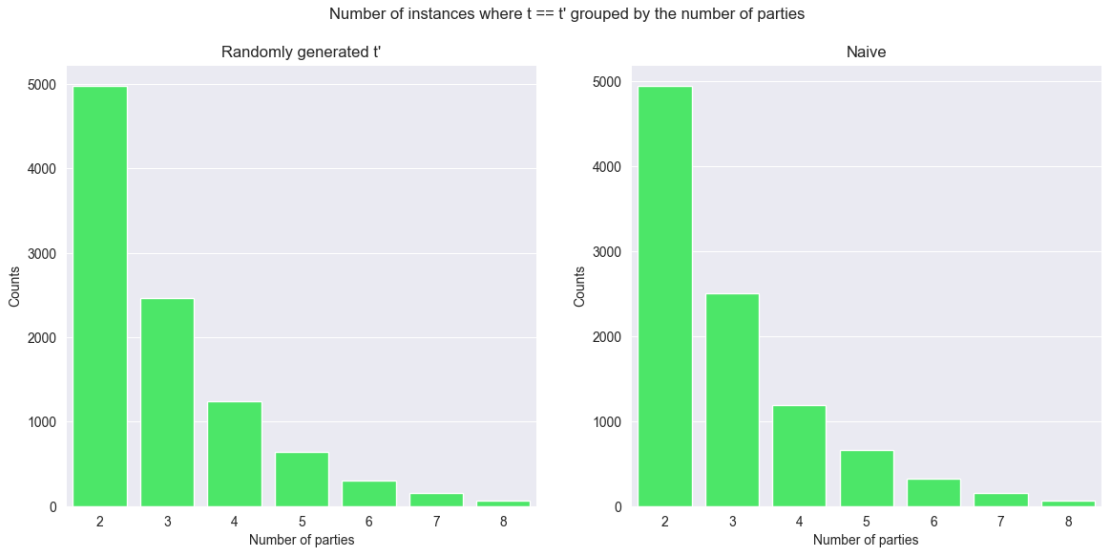

Abbildung 5.1: Number of instances where $t' = t$

and t'.

The hypothesis to be verified is that, whenever it uses $t' \neq t$ as input, the reconstruction process has at most a $\frac{1}{2}$ chance of producing the correct secret. Clearly, if $t = t'$, then the reconstruction process results in the correct secret with probability 1: these cases drop exponentially as n grows, as depicted in figure 5.1. All such cases are not considered for the analysis.

To confirm the hypothesis, the dataset is grouped by the number of parties, then all instances where the reconstructed secret does not match the original secret are counted and compared against the total instances. The test results are reported in tables 5.3 and 5.4 The experimental results confirm the hypothesis.

Number of parties	Successful instances	Total instances	Percentage(%)
2	2463	5050	51.2
3	3749	7500	50.0
4	4456	8807	49.4
5	4648	9342	50.2
6	4875	9682	49.6
7	4947	9850	49.8
8	4908	9939	50.6
Total	30046	60170	49.9

Tabelle 5.3: Succesful reconstructions when $t \neq t'$, 0 strategy

Number of parties	Successful instances	Total instances	Percentage(%)
2	2512	5018	50.1
3	3743	7534	49.7
4	4387	8756	50.1
5	4677	9358	50.0
6	4894	9698	50.5
7	4950	9848	50.3
8	4971	9934	50.3
Total	30134	60146	50.1

Tabelle 5.4: Succesful reconstructions when $t \neq t'$, Random strategy

The other configuration of subsets of shares consists in the attacker possessing θ and some of ρ_i; this knowledge substantially provides access to a subset of bits of the secret bitstring r. Since the parity of a bit string is dependent on all its bits, and the bits are sampled independently of each other, it is expected that possession subsets of shares of this kind does not leak the secret nor reduce the number of possible secrets. Moreover, it is clear that as the number of ρ_i shares included in a subsets increases, no additional information is gained. To verify that subsets of shares including θ do not leak the secret, the reconstruction procedure with the subset as input is simulated. The complete set of shares is generated, then a random subset is sampled and fixed as input for the reconstruction procedure and compared to the output of reconstruction from the complete set. A dataset is created from repeated iterations of the process for each value of n; $n = 2$ not considered, as possessing both θ and ρ_i in this setting is impossible.
The behavior of the program is then statistically analyzed to verify the expectation that the probability of the two reconstructions colliding be at most $\frac{1}{2}$.
First the dataset is filtered based on the condition that the reconstruction from partial set of shares and the complete collide and produce the same result, then the results are grouped by the size difference between the randomly sampled subset and the complete set. Table 5.5 documents the final results of the data analysis. As the probabilities of successful reconstruction from subsets of shares does not significantly deviate from $\frac{1}{2}$, no information leak happens from subsets of shares.

Size difference	Correct reconstructions	Total rec.	Percentage
1	6317	12639	50.0
2	6318	12721	49.7
3	4342	8740	49.7
4	3163	6222	50.8
5	1953	4050	48.2
6	1276	2466	51.7
7	576	1162	49.6
Total	23945	48000	49.9

Tabelle 5.5: Succesful reconstructions from randoms subsets of shares including ρ_n

5.2 Unclonability Testing

In addition to verifying the validity of the proposed USS construction as a regular secret sharing scheme, our objective is to test its unclonability property.
Concisely, this property should ensure that, if shares of a secret are distributed among external storage providers and these providers attempt to copy their share and send it to two different adversaries, the probability that both adversaries reconstruct the correct secret is at most $\frac{1}{2}$ plus a negligible value.
To test this property, we prepare two attack strategies based on the experiment defined in section 3.3. Parties in the strategies are $\mathcal{A}$, the secret holder, P_i the share holding parties, $\mathcal{B}$ and $\mathcal{C}$ who are the adversaries.

- Measure and retransmit strategy

  1. $\mathcal{A}$ selects a secret bit $s \in \{0,1\}$ and prepares $(\rho_1, \ldots, \rho_n) \leftarrow \mathsf{share}(s, n)$

  2. P_i receives ρ_i and it samples a random bit b, then if $b = 0$, it measures ρ_i in the computational basis, otherwise it measures in the x-basis (equivalent to applying an hadamard gate before measuring). Finally, it prepares $\rho_{\mathcal{B}}$ and $\rho_{\mathcal{C}}$ according to the following table:

b	Measurement	$\rho_{\mathcal{B}}$	$\rho_{\mathcal{C}}$
0	0	$\|0\rangle$	$\|0\rangle$
0	1	$\|1\rangle$	$\|1\rangle$
1	0	$\|+\rangle$	$\|+\rangle$
1	1	$\|-\rangle$	$\|-\rangle$

     Both $\mathcal{B}$ and $\mathcal{C}$ receive ρ_n, as it is a computational basis state

3. $\mathcal{B}$ and $\mathcal{C}$ attempt reconstruction with the states they received in step 2

- Teleportation strategy
  1. $\mathcal{A}$ selects a secret bit $b \in \{0,1\}$ and prepares $(\rho_1, \ldots, \rho_n) \leftarrow \mathsf{share}(b, n)$
  2. Each party P_i except for P_n pre-shares a pair of entangled qubits with $\mathcal{B}$
  3. P_i teleports ρ_i to $\mathcal{B}$, meanwhile $\mathcal{C}$ receives a random state between $\{|0\rangle, |1\rangle, |+\rangle, |-\rangle\}$. Similarly to the measurement strategy, they both receive ρ_n
  4. $\mathcal{B}$ and $\mathcal{C}$ attempt reconstruction on the states they received

By simulating these two attack multiple times, it can be verified whether the probability of both $\mathcal{B}$ and $\mathcal{C}$ reconstructing the secret is upper-bounded by $\frac{1}{2}$.

5.2.1 Difference from original adversarial model

One of the cornerstones of unclonable cryptography is represented by cloning games of BB84 states, also known as monogamy-of-entanglement games. Monogamy-of-entaglement games were introduced by Tomamichel et al. in [28], and they were subsequently adopted by Broadbent and Lord to investigate unclonable encryption, [5].
A simplified formulation of a BB84 cloning game, as designed in [28] is the following:

- B and C prepare a tripartite quantum state ρ_{ABC}. For simplicity, it is assumed that this is a 3-qubit state. The state is then split between the three parties, with ρ_A going to A, with ρ_B going to B and similarly for C.
- A samples a random bit θ and based on this, measures ρ_A either in the computational basis or in the x-basis. θ is announced to B and C
- B and C perform a measurement on their part of the quantum state, trying to guess A's measurement outcome.
- B and C win if they both guess A's measurement result.

It is clear that this game is similar to the attack strategies previously defined, if not for the fact that, in these strategies, B and C do not have a role in preparing the state they will play on. Since the original BB84 cloning game can be recast to a situation where, rather than B and C preparing a state, they receive one from P_i who applies the state splitting strategy on the state $|x^\theta\rangle = H^\theta |x\rangle$ it received from A, our strategies can be said to be two different approaches to a cloning game. Figure 5.2 schematizes the game.

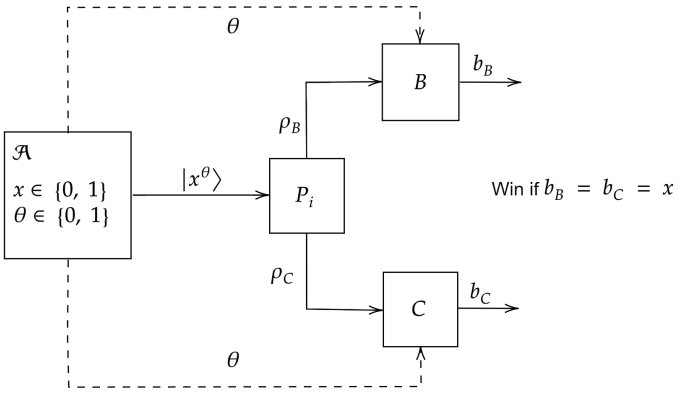

Abbildung 5.2: BB84 cloning game

Our approach to cloning games differs from [1]. When discussing unclonability in function of cloning games, Ananth. et al. investigate strategie with the following boundary: P_i, when receiving a quantum state dependent on x, θ, is not able to create a bipartite state ρ_{BC} that allows both B and C to determine x with a certain degree of confidence. Formally, if σ_i^0, σ_i^1 are defined as the states that B would receive if x was respectively 0 or 1 and ζ_i^0, ζ_i^1 similarly defined for C, either σ_i^0 and σ_i^1 are clearly distinguishable or ζ_i^0 and ζ_i^1 are. This can be quantified by using the trace distance TD between two states as a metric of their distinguishability, [10]: either $\mathsf{TD}(\sigma_i^0, \sigma_i^1) < c$ or $\mathsf{TD}(\zeta_i^0, \zeta_i^1) < c$.
Our strategies violate this boundary, since even if a certain degree of randomness is involved in P_i operations, it cannot be excluded that in certain instances B and C both receive highly-distinguishable states. Additionally, our strategies also interfere with the regular functionality of a secret sharing scheme, since they render the ρ_i shares either by

measuring them, or by modifying them through the teleportation circuit.

5.2.2 Practical testing

The testing is carried out by producing `qiskit` simulations of the attack, where the two adversaries $\mathcal{B}$ and $\mathcal{C}$ are represented as quantum registers inside the main circuit, whose state is appropriately initiated according to the strategy operations. Multiple reconstructions are attempted for different sets of shares, generated by a randomly selected secret for each number of parties that can be simualted. Then, a dataset is created containing the number of parties, the secret, the secret as reconstructed by $\mathcal{B}$ and the secret as reconstructed by $\mathcal{C}$. We denote the output reconstructions of $\mathcal{B}$ and $\mathcal{C}$ with $b_{\mathcal{B}}$ and $b_{\mathcal{C}}$ respectively.
Table 5.6 and figure 5.3 present the results, where column $b_{\mathcal{B}} = b_{\mathcal{C}}$ denotes the instances in the dataset where both $\mathcal{B}$ and $\mathcal{C}$ correctly reconstructed the secret . Overall, the data

Num. of parties	$b_{\mathcal{B}} = b_{\mathcal{C}}$	Total	Percentage
Measure and retransmit strategy			
2	3119	5000	62.4
3	2215	5000	44.3
4	1703	5000	34.1
5	1469	5000	29.4
6	1350	5000	27.0
7	1357	5000	27.1
8	1305	5000	26.1
Total	12518	35000	35.8
Teleportation strategy			
2	2544	5000	50.9
3	2459	5000	49.2
4	2489	5000	49.8
5	2507	5000	50.1
6	2563	5000	51.3
7	2497	5000	50.0
8	2488	5000	49.8
Total	17547	35000	50.1

Tabelle 5.6: Teleportation strategy results

aligns with the expectation that $\mathcal{B}$ and $\mathcal{C}$ both succeed in about 50% of the attempted reconstructions. However, it seems that the unclonability property is violated by the

measure and retransmit strategy for a secret shared between only two parties.

As expected, in the teleportation strategy, B always reconstructs the correct secret

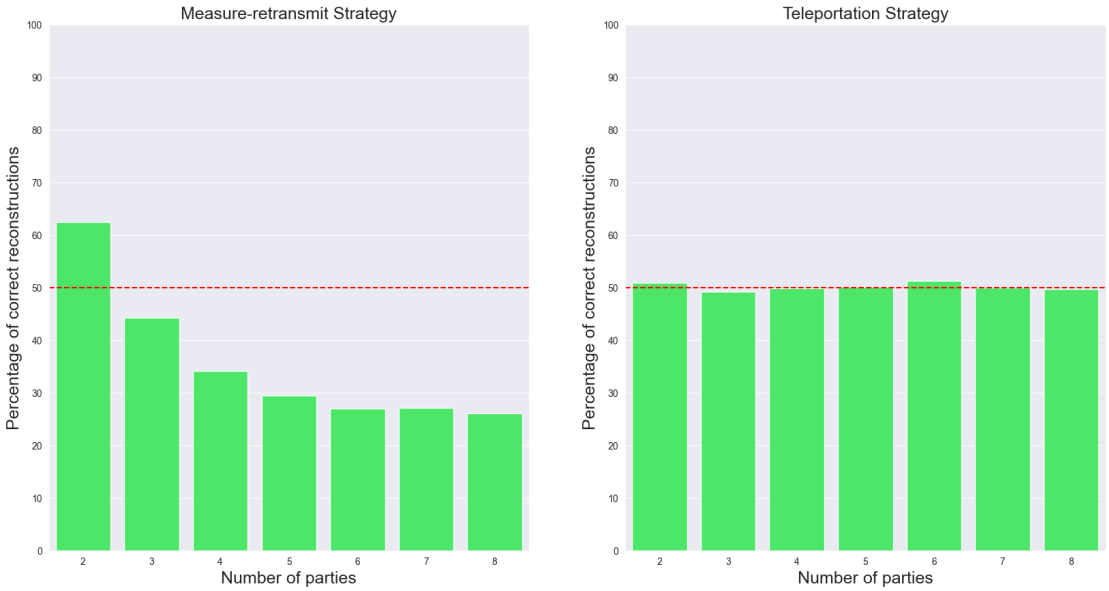

Abbildung 5.3: Percentage of succesful reconstructions with cloned shares

with probability 1, while $\mathcal{C}$ is forced to perform a random guess, resulting in an overall probably of about $\frac{1}{2}$.

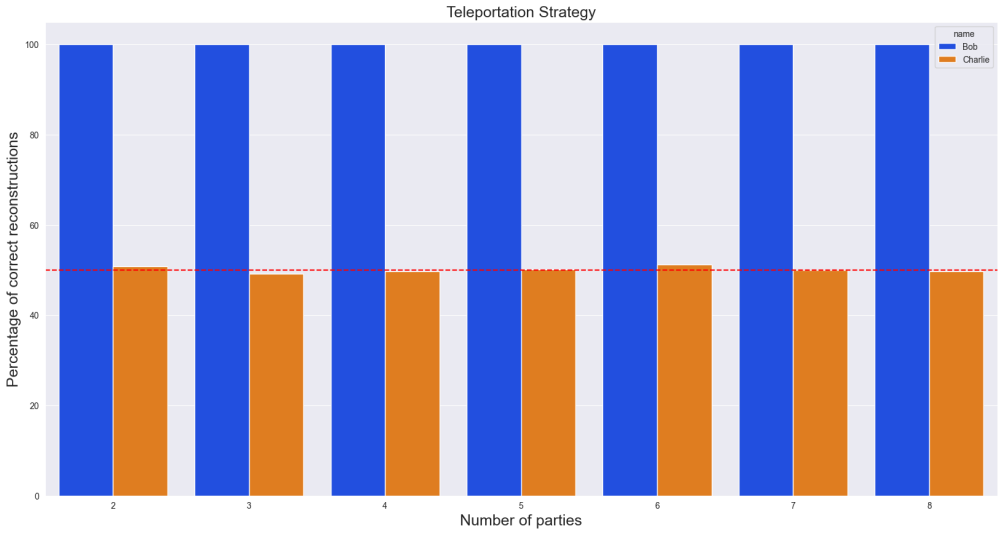

Abbildung 5.4: Comparison between $\mathcal{B}$ and $\mathcal{C}$ reconstructions in the teleportation strategy

Conclusions

Our work positions itself in the context of unclonable secret sharing, a novel and yet relatively unexplored cryptographic primitive that attained promising theoretical results in addressing the unavoidable limitation of classical secret sharing, constituted by the local cloning of shares entrusted to third party storage providers.
The main objective of this work was to design a functional prototype of the construction for unclonable secret sharing against adversaries with limited quantum resources, as described in USS founding paper by Ananth et al., [1]. The quantum programming framework `qiskit` was employed to simulate the unclonable secret sharing procedure and for experimental validation of its functionality.
We found that our implementation satisfies the functional requirements of a secret sharing scheme, as it provides correct, consistent and secure functionalities. Moreover, we experimentally verified that the unclonability property stands against specific attacks, executed in the simulation setting.
These results contribute to the unclonable secret sharing discourse by providing a proof-of-concept of a specific unclonable secret sharing procedure and by assessing its validity as a scheme in the simulation setting, through a combination of automated testing and statistical analysis.
Future developments could focus on more complicated simulations, for instance by testing the unclonability property against adversaries carrying out more advanced attacks, that do not destroy the original shares or that try to increase their winning probability. Additionally, larger-scale simulations could be designed, or the implementation possibilities of unclonable secret sharing on physical, noisy quantum devices could be investigated.

Literaturverzeichnis

[1] Prabhanjan Ananth, Vipul Goyal, Jiahui Liu, and Qipeng Liu. Unclonable secret sharing, 2024.

[2] L E Bassham, III, A L Rukhin, J Soto, J R Nechvatal, M E Smid, E B Barker, S D Leigh, M Levenson, M Vangel, D L Banks, N A Heckert, J F Dray, and S Vo. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical report, 2010.

[3] Charles H. Bennett and Gilles Brassard. Quantum cryptography: Public key distribution and coin tossing. *Theoretical Computer Science*, 560:7–11, December 2014.

[4] G. R. BLAKLEY. Safeguarding cryptographic keys. 1979 International Workshop on Managing Requirements Knowledge, IEEE, 6 1979.

[5] Anne Broadbent and Sébastien Lord. Uncloneable quantum encryption via oracles. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

[6] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, pages 383–395, 1985.

[7] Cirq Developers. Cirq, May 2024.

[8] Mark Fingerhuth, Tomáš Babej, and Peter Wittek. Open source software in quantum computing. *PLOS ONE*, 13(12):e0208561, December 2018.

[9] LANCE FORTNOW. *The Golden Ticket: P, NP, and the Search for the Impossible*. Princeton University Press, 2013.

[10] Alexei Gilchrist, Nathan K. Langford, and Michael A. Nielsen. Distance measures to compare real and ideal quantum processes. *Physical Review A*, 71(6), June 2005.

[11] Daniel Gottesman. Theory of quantum secret sharing. *Physical Review A*, 61(4), March 2000.

[12] Vipul Goyal, Yifan Song, and Akshayaram Srinivasan. Traceable secret sharing and applications. Cryptology ePrint Archive, Paper 2021/871, 2021.

[13] Amir Herzberg, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In Don Coppersmith, editor, *Advances in Cryptology — CRYPT0' 95*, pages 339–352, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[14] Mark Hillery, Vladimír Bužek, and André Berthiaume. Quantum secret sharing. *Physical Review A*, 59(3):1829, 1999.

[15] He-Liang Huang, Xiao-Yue Xu, Chu Guo, Guojing Tian, Shi-Jie Wei, Xiaoming Sun, Wan-Su Bao, and Gui-Lu Long. Near-term quantum computing techniques: Variational quantum algorithms, error mitigation, circuit compilation, benchmarking and classical simulation. *Science China Physics, Mechanics ; Astronomy*, 66(5), 4 2023.

[16] Thomas Häner, Damian S Steiger, Krysta Svore, and Matthias Troyer. A software methodology for compiling quantum programs. *Quantum Science and Technology*, 3(2):020501, February 2018.

[17] Amit Jamadagni, Andreas M. Läuchli, and Cornelius Hempel. Benchmarking quantum computer simulation software packages: State vector simulators, 2024.

[18] Ali Javadi-Abhari, Matthew Treinish, Kevin Krsulich, Christopher J. Wood, Jake Lishman, Julien Gacon, Simon Martiel, Paul D. Nation, Lev S. Bishop, Andrew W. Cross, Blake R. Johnson, and Jay M. Gambetta. Quantum computing with qiskit, 2024.

[19] Anders Karlsson, Masato Koashi, and Nobuyuki Imoto. Quantum entanglement for secret sharing and secret splitting. *Phys. Rev. A*, 59:162–168, Jan 1999.

[20] Stephan Krenn and Thomas Lorünser. *An Introduction to Secret Sharing*. Springer International Publishing, 2023.

[21] He Lu, Zhen Zhang, Luo-Kan Chen, Zheng-Da Li, Chang Liu, Li Li, Nai-Le Liu, Xiongfeng Ma, Yu-Ao Chen, and Jian-Wei Pan. Secret sharing of a quantum state. *Physical Review Letters*, 117(3), 7 2016.

[22] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[23] Python Software Foundation. *secrets — Generate secure random numbers for managing secrets*, 2024. Accessed: 2024-12-22.

[24] Manuel A. Serrano, José A. Cruz-Lemus, Ricardo Perez-Castillo, and Mario Piattini. Quantum software components and platforms: Overview and quality assessment. *ACM Comput. Surv.*, 55(8), December 2022.

[25] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, October 1997.

[26] Robert S. Smith, Michael J. Curtis, and William J. Zeng. A practical quantum instruction set architecture, 2016.

[27] Damian S. Steiger, Thomas Häner, and Matthias Troyer. Projectq: an open source software framework for quantum computing. *Quantum*, 2:49, January 2018.

[28] Marco Tomamichel, Serge Fehr, Jędrzej Kaniewski, and Stephanie Wehner. A monogamy-of-entanglement game with applications to device-independent quantum cryptography. *New Journal of Physics*, 15(10):103002, October 2013.

[29] Stephen Wiesner. Conjugate coding. *SIGACT News*, 15(1):78–88, January 1983.

[30] Kieran Young, Marcus Scese, and Ali Ebnenasir. Simulating quantum computations on classical machines: A survey, 2023.

A Appendix

A.1 Quantum Teleportation

Alice possesses two qubits: the quantum state to be teleported $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, and one qubit of an entangled pair. The entangled pair shared between Alice and Bob is given by the Bell state:

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle). \tag{A.1}$$

The combined initial state of the system is:

$$|\Psi_{\text{initial}}\rangle = |\psi\rangle \otimes |\Phi^+\rangle = (\alpha|0\rangle + \beta|1\rangle) \otimes \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle). \tag{A.2}$$

Alice applies a CNOT gate, with $|\psi\rangle$ as control qubit and target her half of the entangled pair $|\psi\rangle$ as the control. This transforms the state to:

$$|\Psi_1\rangle = \frac{1}{\sqrt{2}}\big(|0\rangle(\alpha|00\rangle + \beta|11\rangle) + |1\rangle(\alpha|10\rangle + \beta|01\rangle)\big). \tag{A.3}$$

Next, Alice applies a Hadamard gate to the input qubit, resulting in the state:

$$\begin{aligned}|\Psi_2\rangle = \frac{1}{2}\big(&|00\rangle(\alpha|00\rangle + \beta|11\rangle) + |01\rangle(\alpha|10\rangle + \beta|01\rangle) \\ &+ |10\rangle(\alpha|00\rangle - \beta|11\rangle) + |11\rangle(\alpha|10\rangle - \beta|01\rangle)\big).\end{aligned} \tag{A.4}$$

Alice then measures her two qubits in the computational basis, collapsing the state to one of the four possible outcomes. The result of her measurement is communicated to Bob via a classical channel.
Depending on Alice's measurement outcome, Bob's qubit is left in one of four states. Using the classical information sent by Alice, Bob applies a corresponding Pauli operation to recover the original state $|\psi\rangle$:

- If Alice measures $|00\rangle$, Bob's qubit is $|\psi\rangle$ (no operation needed).
- If Alice measures $|01\rangle$, Bob applies X gate.
- If Alice measures $|10\rangle$, Bob applies Z gate.
- If Alice measures $|11\rangle$, Bob applies XZ gates.

A.2 Content of the DVD

The attached DVD contains

- PDF file of this thesis
- Project files of the unclonable secret sharing scheme implementation. The project structure is

  USS Root folder

  src Source code folder

  uss Unclonable secret sharing module

  core.py share() and reconstruct() function

  sharesgenerator.py Specialized subroutines for creating the quantum shares

  utils Utility functions

  tests Automated test cases

  data Dataset and logs folder

  finalAnalysisReport.txt Report of NIST RNG test suite

  information_leak_naive.csv Dataset to test information leak with subsets of shares not including ρ_n and naive strategy

  information_leak_random.csv Dataset to test information leak with subsets of shares not including ρ_n and random strategy

measure_retransmit_strategy.csv Dataset of attacks by measure and retransmit strategy

nist-community-result.txt Report of the community implementation of NIST RNG test suite

teleportation_strategy.csv Dataset of attacks by teleportation strategy

subsets.csv Dataset of reconstructions with partial sets of shares including ρ_n

notebooks Jupyter notebooks for creating the datasets

Erklärung zur selbständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original