UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN MATHÉMATIQUES ET INFORMATIQUE APPLIQUÉES

PAR
NASR AKRAM

LA VULNÉRABILITÉ INHÉRENTE DU CRYPTOSYSTÈME RSA EXPOSÉE PAR LA CRYPTANALYSE QUANTIQUE

REMERCIEMENTS

Avant tout développement sur ce mémoire, il me semble indispensable d'exprimer mes sincères remerciements aux personnes qui ont rendu possible la réalisation de ce travail.

Je tiens à exprimer ma profonde gratitude à mon directeur de recherche, M. Mhamed Mesfioui, pour sa précieuse supervision, son soutien inébranlable et ses conseils éclairés tout au long de ce parcours. Sa grande expertise et son approche pédagogique ont grandement contribué à l'élaboration de ce travail.

De même, mes remerciements vont à mon codirecteur de recherche, M. François Meunier, dont les orientations, les critiques constructives et l'accompagnement rigoureux ont été d'une aide inestimable. Sa vision critique et son engagement envers l'excellence académique m'ont permis de mener à bien cette recherche.

Cette expérience a été enrichissante et formatrice, et c'est en grande partie grâce à leur encadrement exemplaire, leur disponibilité et leur encouragement constant. Leurs qualités humaines et professionnelles ont été pour moi une source d'inspiration et de motivation tout au long de mon parcours de recherche.

Je tiens également à remercier toutes les personnes qui, de près ou de loin, ont contribué à la réalisation de ce mémoire. Leur soutien, sous quelque forme que ce soit, a été un pilier dans l'achèvement de ce travail.

Enfin, je voudrais exprimer ma gratitude à tous ceux qui, par leur présence, leurs paroles ou leurs pensées, m'ont soutenu et encouragé durant ces années d'étude.

RÉSUMÉ

Ce mémoire examine en profondeur la menace croissante que représente l'informatique quantique pour la sécurité des cryptosystèmes classiques, en se concentrant particulièrement sur le système RSA (Rivest, Shamir, Adleman), le cryptosystème asymétrique le plus utilisé et reconnu pour sa robustesse. Malgré sa réputation de résistance aux attaques par les technologies informatiques classiques, notre recherche révèle des vulnérabilités préoccupantes.

Le cryptosystème RSA, de même que d'autres systèmes tels que l'AES et le SHA, ont été analysés minutieusement. Nous avons exploré les subtilités de l'informatique quantique, notamment les phénomènes d'intrication, de superposition, et les *bits* quantiques ou *qubits*. Cette exploration a permis de comprendre comment ces éléments pourraient être exploités pour menacer la sécurité des systèmes informatiques actuels.

Un aspect central de notre étude met en lumière un progrès mathématique majeur facilité par l'informatique quantique : la capacité à résoudre efficacement des problèmes mathématiques complexes qui sont extrêmement difficiles, voire pratiquement impossibles, pour les ordinateurs classiques. En particulier, nous avons examiné la manière dont les algorithmes quantiques, tels que l'algorithme de Shor, peuvent théoriquement factoriser les grandes clés publiques RSA. Alors que la factorisation de petites clés est réalisable avec des ordinateurs classiques, la factorisation de clés complexes comme RSA-2048 reste hors de portée, soulignant une vulnérabilité potentielle majeure face aux ordinateurs quantiques.

J'ai développé QURSA, une application qui met en lumière la puissance de calcul des ordinateurs quantiques et leur impact potentiel sur la sécurité du cryptosystème RSA (Visiter l'application). L'application est une implémentation avancée de l'algorithme de Shor, incluant la correction d'erreurs quantiques. Elle est actuellement disponible et hébergée pour des tests et des simulations, permettant aux utilisateurs de simuler l'attaque de Shor sur des clés RSA de différentes tailles.

Mes résultats confirment la vulnérabilité du cryptosystème RSA face à l'algorithme de Shor. Ils soulignent l'urgence de migrer vers des solutions de cryptage quantique-résistant pour garantir la sécurité des données sensibles.

Le mémoire propose une exploration approfondie de l'informatique quantique, une analyse critique du cryptosystème RSA et une démonstration concrète des faiblesses de sécurité à travers l'application QURSA. Face à l'émergence inévitable de l'informatique quantique, nos résultats incitent à une action rapide pour le développement et l'adoption de méthodes cryptographiques avancées, à l'épreuve des menaces quantiques.

ABSTRACT

This thesis delves into the escalating threat quantum computing poses to the security of classical cryptosystems, with a focus on the RSA (Rivest, Shamir, Adleman) system, the most widely used and recognized asymmetric cryptosystem known for its robustness. Despite its reputation for resistance against attacks by classical computing technologies, our research uncovers concerning vulnerabilities.

The RSA cryptosystem, along with other systems such as AES and SHA, have been meticulously analyzed. We have explored the subtleties of quantum computing, especially phenomena like entanglement, superposition, and quantum bits or *qubits*. This exploration has enabled an understanding of how these elements could be leveraged to threaten the security of current computer systems.

A central aspect of our study highlights a significant mathematical advancement facilitated by quantum computing : the ability to efficiently solve complex mathematical problems that are extremely difficult, or practically impossible, for classical computers. Specifically, we examined how quantum algorithms, such as Shor's algorithm, could theoretically factorize large RSA public keys. While the factorization of small keys is achievable with classical computers, the factorization of complex keys like RSA-2048 remains out of reach, underscoring a potential major vulnerability to quantum computers.

I developed QURSA, an application that illuminates the computational power of quantum computers and their potential impact on the security of the RSA cryptosystem (Visit the application). The application is an advanced implementation of Shor's algorithm, including quantum error correction. It is currently available and hosted for testing and simulations, allowing users to simulate Shor's attack on RSA keys of various sizes.

My findings confirm the vulnerability of the RSA cryptosystem to Shor's algorithm. They underscore the urgency of migrating to quantum-resistant encryption solutions to ensure the security of sensitive data.

The thesis offers an in-depth exploration of quantum computing, a critical analysis of the RSA cryptosystem, and a concrete demonstration of security weaknesses through the QURSA application. Given the inevitable emergence of quantum computing, our results prompt swift action for the development and adoption of advanced cryptographic methods, immune to quantum threats.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Intoduction

1.1 Objectifs du mémoire

L'informatique quantique, cette nouvelle frontière de la science computationnelle, soulève une question fascinante : comment cette technologie pourrait-elle redéfinir les limites de la sécurité informatique ? Prenons un exemple simple : si je vous dis que le produit de deux nombres premiers est 15, il est facile de déduire que ces nombres sont 3 et 5. Mais, que se passe-t-il si je vous présente un produit de deux nombres premiers, chacun ayant 300 chiffres ? La complexité devient vertigineuse, car les nombres premiers ne sont divisibles que par eux-mêmes et par 1. Cette tâche, pratiquement insurmontable pour les ordinateurs classiques, est précisément ce que les algorithmes quantiques, tels que celui de Shor, sont en train de rendre possible.

Imaginez le potentiel de cette technologie : en exploitant la puissance des ordinateurs quantiques, des problèmes mathématiques qui semblaient autrefois inaccessibles deviennent résolubles. Cette capacité a des implications majeures, notamment pour le cryptosystème RSA. Utilisé pour sécuriser les données dans le monde entier, RSA repose sur la difficulté de factoriser de grands nombres. Cependant, avec l'avènement des ordinateurs quantiques, la robustesse du RSA est remise en question. La recherche présentée ici explore cette vulnérabilité en détail, révélant comment, grâce aux progrès mathématiques et informatiques, RSA pourrait devenir non seulement faible, mais potentiellement cassable.

Au cœur de cette étude, nous nous concentrons principalement sur le cryptosystème RSA, un pilier de la sécurité informatique, désormais confronté à des défis sans précédent avec l'avènement de l'informatique quantique. Cette recherche vise à explorer en profondeur la manière dont l'informatique quantique, en particulier l'algorithme de Shor, pourrait théoriquement briser le RSA, remettant ainsi en cause sa capacité à sécuriser les données dans un avenir proche. Bien que notre étude intègre également une exploration des cryptosystèmes AES et SHA, ces derniers sont abordés principalement dans le contexte de leur complémentarité au RSA, notamment dans les systèmes hybrides et les applications liées à l'intégrité et aux signatures.

Pour illustrer de manière tangible les vulnérabilités du RSA face aux attaques quantiques, j'ai développé QURSA, une application innovante qui utilise l'algorithme de Shor pour démontrer la fragilité potentielle du RSA sous la pression des ordinateurs quantiques (Visiter l'application). À travers des simulations en temps réel et une gestion efficace du bruit quantique sur les plateformes d'IBM, QURSA a permis de décomposer des clés RSA de différentes longueurs, ouvrant la voie à des résultats prometteurs. Bien que ces expériences soient actuellement limitées par la puissance des ordinateurs quantiques existants, elles suggèrent la vulnérabilité potentielle de clés RSA bien plus complexes, y compris celles de la taille de RSA-2048.

Cette recherche apporte une contribution significative à la compréhension de l'impact potentiel de l'infor-

matique quantique sur le RSA et, par extension, sur la cryptographie moderne. QURSA, en tant qu'outil open source et de pointe, incarne cette révolution technologique et souligne l'importance d'élaborer des cryptosystèmes résistants aux attaques quantiques. En outre, cette étude vise à sensibiliser aux défis et aux risques associés à l'informatique quantique, encourageant un débat plus large sur la nécessité de développer des solutions cryptographiques plus sûres et efficaces pour l'ère quantique.

1.2 Conclusion

En conclusion, l'informatique quantique est une technologie passionnante qui offre des avantages considérables dans de nombreux domaines. Cependant, comme pour toute technologie, il y a des risques potentiels qui doivent être pris en compte. Cette recherche met en évidence la fragilité de la cryptographie classique face aux ordinateurs quantiques et souligne l'importance de développer des méthodes de cryptographie résistantes aux attaques quantiques pour protéger les données sensibles. L'application QURSA démontre de manière convaincante la vulnérabilité du cryptosystème RSA et montre la nécessité d'une nouvelle approche cryptographique pour assurer la sécurité des données dans l'avenir.

Chapitre 2

L'informatique quantique

2.1 Introduction

Ce chapitre propose une analyse approfondie de l'émergence de la révolution quantique et de son influence déterminante sur les sphères de la technologie avancée. Il vise à offrir une perspective historique et analytique sur l'étincelle initiale qui a déclenché le développement fulgurant du domaine quantique. Cette ascension remarquable s'est appuyée sur les contributions fondamentales de nombreux scientifiques éminents, dont les travaux pionniers ont jeté les bases de la conception et de la réalisation de l'ordinateur quantique.

Nous explorerons en détail les limitations inhérentes à l'informatique classique (voir 2.3), et examinerons l'observation de Moore (voir 2.3.2), en particulier comment elle se confronte à ses propres limites dans le contexte actuel. La problématique de la croissance exponentielle des besoins en puissance de calcul (voir 2.3.3) sera également abordée, ainsi que les mécanismes sous-jacents à l'ordinateur quantique et les principes primordiaux régissant les propriétés quantiques (voir 2.4).

Nous mettrons en lumière les différences fondamentales entre le *qubit*, élément de base de l'informatique quantique, et le *bit* classique (voir 2.4.2), puis nous analyserons les diverses portes logiques quantiques qui constituent le cœur de l'ordinateur quantique (voir 2.4.6). La question du bruit informatique, un défi majeur pour la fiabilité des systèmes quantiques, sera abordée, ainsi que les stratégies déployées par l'informatique quantique pour raccommoder cette problématique (voir 2.4.10).

Enfin, nous discuterons des implications révolutionnaires de cette technologie pour le monde de la recherche scientifique et de l'innovation technologique. Nous soulignerons notamment comment l'avènement de l'ordinateur quantique est susceptible d'impacter de manière significative la cryptographie moderne, en mettant en péril des systèmes cryptographiques établis comme le système RSA, un composant fondamental et indispensable dans l'architecture de la sécurité informatique courante, joue un rôle primordial dans la protection des données sur le World Wide Web (voir [47]). Cette analyse exhaustive permettra de mieux appréhender les enjeux et les défis posés par cette avancée technologique sans précédent.

2.2 La révolution quantique

La physique quantique est une branche de la physique, pareillement connue sous le nom de la mécanique quantique ou la théorie quantique. La mécanique est cette partie de la physique concernée par les choses qui bougent, des boulets de canon, des véhicules, des fusées, des planètes. La mécanique quantique est la partie de la physique qui décrit les mouvements d'objets aux niveaux moléculaire, atomique, subatomique, tels que les photons, les électrons. (Les molécules sont constituées d'atomes, qui sont eux même constitués d'électrons et de noyaux. Les noyaux sont constitués de protons et de neutrons. Les

protons et neutrons sont constitués de quarks et de gluons). *« I think I can safely say that nobody understands quantum mechanics »*, est la citation fameuse du grand physicien et lauréat du prix Nobel Richard Feynman [29]. La science a en effet parcouru un long chemin dans la compréhension de la mécanique quantique. Grâce à cette compréhension, les avancées technologiques révolutionnaires, comme l'invention de l'ordinateur quantique, les super lasers, les horloges ultra précises sont apparues. Sans une solide compréhension de la mécanique quantique, aurions-nous même pu rêver de créer quelque chose d'aussi puissante comme l'ordinateur quantique ? En fait, la physique quantique peut être perçue comme un ensemble de théories et d'expérimentations qui sortent de l'ordinaire. Ses origines ont commencé avec un ensemble d'explications mathématiques et de phénomènes physiques qui ne correspondent pas aux principes de la physique classique. La flamme de la révolution quantique a débuté en 1900 où, le physicien allemand Max Planck présente un article à la société allemande de la physique, dans lequel il dérive la théorie expérimentale du corps noir [7]. On peut décrire ce dernier comme un corps idéal qui absorbe toute l'énergie électromagnétique qu'il reçoit, sans en réfléchir ni en transmettre. Cependant, ce phénomène est pratiquement impossible, car un corps qui a ces propriétés verra sa température augmenter indéfiniment. En effet, à température constante, le corps noir réémet l'énergie qu'il absorbe sous forme de rayonnement électromagnétique. Son spectre d'émission dépend donc de sa température (Fig. 2.1).

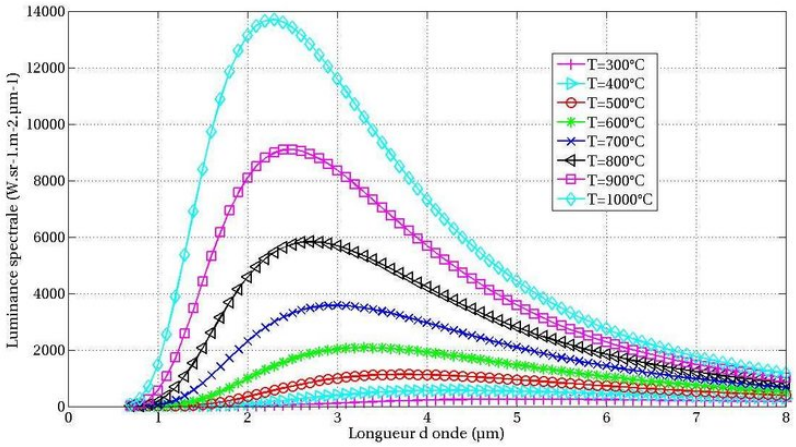


FIGURE 2.1 – Luminance du corps noir en fonction de la longueur d'onde pour diérentes températures

Dans sa quête pour réconcilier les résultats expérimentaux avec les théories préétablies, Max Planck s'est lancé dans une démarche audacieuse et novatrice. Confronté à des observations qui défiaient l'explication conventionnelle, il postula que la transmission d'énergie entre les atomes ne s'effectuait pas de manière continue, mais par des paquets énergétiques distincts, désormais connus sous le nom de *quanta*. Cette hypothèse révolutionnaire a mené Planck à formuler une relation fondamentale : $E = nhf$, où *« n »* représente un nombre entier allant de un à l'infini, *« f »* la fréquence du rayonnement électromagnétique et *« h »* une constante. En comparant cette formule avec les données expérimentales relatives au rayonnement du corps noir, Planck put déterminer la valeur de cette constante fondamentale :

$$h = 6,626 \times 10^{-34} joule seconde$$

Cette découverte marque un tournant crucial dans l'histoire de la physique, posant les fondations de la mécanique quantique.
En effet, la constante de Planck est une valeur fondamentale de la nature, aussi importante que la vitesse de la lumière ou la charge d'un électron. En 1905, Albert Einstein, publie un article dans lequel il décrit l'interaction entre la lumière et la matière en donnant une explication possible de l'effet photoélectrique

(Fig. 2.2 - A). Proposant ainsi que la lumière ne transporte pas l'énergie de manière continue, comme on peut s'y attendre, d'une onde classique, mais le transport se fait sous la forme de faisceaux, plus tard nommé photons. En fait, les électrons ne se détacheront du matériau que s'ils reçoivent une énergie suffisamment importante, et puisque l'énergie du photon dépend de la fréquence, seule la lumière au-dessus d'une certaine fréquence générera l'effet photoélectrique (Fig. 2.2 - B).

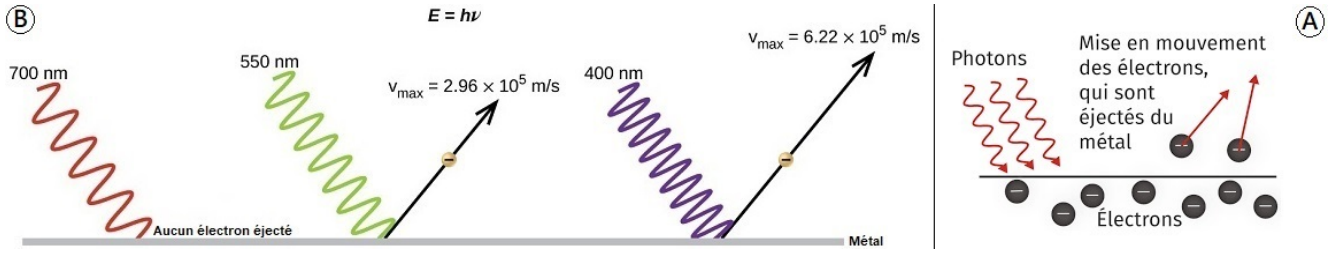


FIGURE 2.2 – Représentation des fréquences de la lumière et leur possibilité d'éjecter les électrons (B) et la représentation de l'expérience photoélectrique (A)

Dans la même année, Einstein publie la formule d'équivalence masse-énergie $E = mc^2$ depuis son article *« L'inertie d'un corps dépend-elle de son contenu énergétique ? »*, [28]. Une révolution mathématique distincte qui a produit un bouleversement dans le domaine de la physique. Contrairement à la théorie de la relativité, la naissance de la physique quantique ne peut pas être attribuée à une seule personne, on la décrit comme une contribution de plusieurs scientifiques telle que :

Louis de Broglie, la matière à l'échelle atomique a des propriétés ondulatoires, la longueur d'onde - 1924

$$\lambda = \frac{h}{p} = \frac{h}{m \times v} \tag{2.1}$$

λ : La longueur d'onde
h : La constante de Planck
p : L'élan
m : La masse
v : La vitesse

Erwin Schrödinger, la prédiction d'une manière analytique et précise de la probabilité d'événements et du comportement futur d'un système quantique, L'équation de Schrödinger - 1926

$$H\Psi = i\hbar\frac{\partial\Psi}{\partial t} \tag{2.2}$$

Ψ : La fonction d'onde
H : L'opérateur Hamiltonien
$\hbar$: La constante de Planck réduite, ($\hbar = \frac{h}{2\pi}$)
∂ : Le taux de changement
∂t : Le taux de changement par rapport au temps

Werner Heisenberg, La position et la fréquence d'une onde ne peuvent pas être mesurées exactement en même temps, Le principe d'incertitude de Heisenberg - 1927

$$\Delta x \times \Delta p \geq \frac{h}{4\pi} \tag{2.3}$$

Δx : L'incertitude de position

Δp : L'incertitude de l'élan

h : La constante de Planck

π : pi

Ainsi, ces trois scientifiques ont établi trois concepts fondamentaux de la mécanique quantique. Le premier concept est défini comme les propriétés quantifiées. La charge électrique, l'énergie, la lumière, le moment angulaire [54] et la matière sont tous quantifiés au niveau microscopique. Cela signifie que ces propriétés ne peuvent prendre que certaines valeurs spécifiques. L'énergie d'un photon ne peut alors avoir que certaines valeurs précises, contrairement à la notion de la mécanique classique, qui suppose que de telles propriétés sont seulement basées sur le modèle du spectre continu (Fig. 2.3).

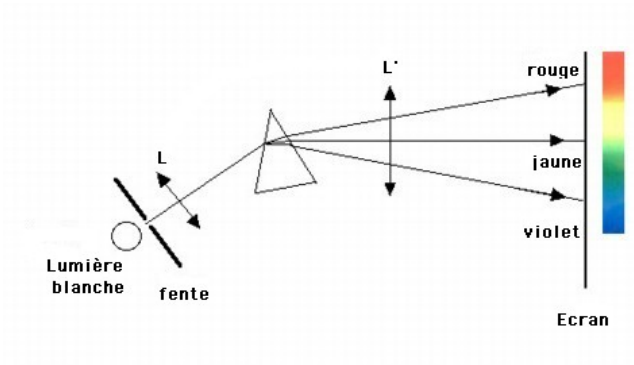


FIGURE 2.3 – Visualisation du spectre de la lumière blanche

Le concept suivant est la nature physique corpusculaire de la lumière. L'idée que la lumière peut être une particule a d'abord rencontré des critiques colossales de la part de la communauté scientifique en 1905, lorsque Einstein a introduit la dualité onde-corpuscule pour la lumière. Décrivant que la lumière à une double nature. Cette façon de voir ce phénomène était contradictoire à la ligne de pensée bien établien, selon laquelle la lumière se comporte que sous la forme d'ondes. La nature ondulatoire de la matière est la dernière notion conceptuelle sur laquelle la physique quantique fut construite. M. Max Born, la décrivant en 1928 comme une probabilité de trouver une molécule subatomique à un point donné dans l'espace [46]. La description mathématique de ce phénomène est connu par la densité de probabilité de la fonction d'onde « *Probability density of the wave function* » exprimé sous la forme algébrique $|\Psi(x)|^2$ (Fig. 2.4). Le plus étonnant est que la matière a également tendance à montrer des propriétés ondulatoires.

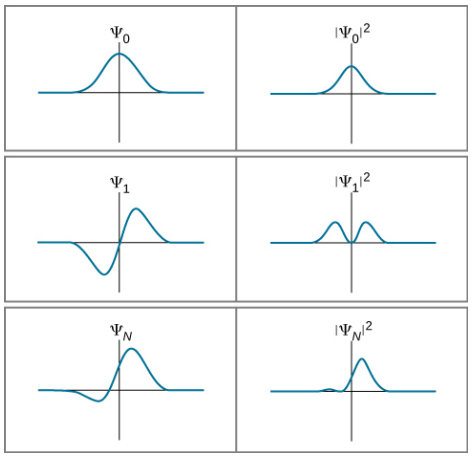


FIGURE 2.4 – Exemples des fonctions d'ondes et leurs probabilités de densité

2.3 Les limitations de l'ordinateur classique

L'humanité a connu une vague d'innovations technologiques grâce à l'influence de Alan Turing sur le développement de l'informatique théorique. Notamment en fournissant une formalisation des concepts d'algorithme et de calculs avec la Machine de Turing, *« Turing machine »* [31], connue aujourd'hui sous le nom de l'ordinateur classique. Ce concept est considéré comme un modèle d'ordinateur à usage général. Bien que cette technologie soit révolutionnaire, il existe cependant certains problèmes informatiques que la révolution numérique ne semble toujours pas pouvoir résoudre. Certains d'entre eux pourraient borner les avancées scientifiques et notre vision du monde, notamment les simulations des réactions subatomiques de la matière. Bien que les ordinateurs conventionnels doublent de puissance et de vitesse de traitement presque tous les deux ans selon la loi de *Moore*, (ce qui sera détailler davantage dans la partie 2.3.2).

Les ordinateurs ne paraissent toujours pas capables de résoudre efficacement des problèmes mathématiques complexes, comme la factorisation première, (cette problématique sera abordée avec plus de détail dans le chapitre III). On peut justifier cette impuissance par le fait que les ordinateurs numériques conventionnels d'aujourd'hui sont construits sur un modèle informatique classique limité. Ce bornage est dû au fait que la puce informatique ordinaire utilise des *bits*. Ce sont de minuscules interruptrices, qui peuvent être soit en position éteinte représentée par zéro (0), soit en position allumée représentée par un (1), (ce concept sera abordé plus en détails dans la partie 2.3.1). Chaque application, chaque site web, chaque photo est composée de millions de *bits*. Cela fonctionne très bien pour la plupart des taches numériques, mais ne représente vraiment pas le fonctionnement réel de l'univers. Cela s'explique du fait que dans la nature, les interactions microscopiques ne sont pas seulement allumées ou éteintes ; ils sont incertains. Et même les meilleurs supercalculateurs ne sont pas très bons pour faire face à l'incertitude.

Lorsqu'on traite les phénomènes au niveau des atomes, les lois de la physique classique deviennent inefficaces pour prédire précisément le trajet et la position, par exemple un électron. Imaginons que nous tentons d'expliquer la distinction fondamentale entre un ordinateur classique et un ordinateur quantique d'une manière métaphorique, sans nous contenter de la voir comme une simple amélioration des performances, à la manière dont on pourrait comparer un ancien modèle de téléphone intelligent à un modèle plus récent, cela serait réducteur et inexact. La comparaison doit être envisagée sur un plan bien plus imagé et profond, tel que la différence entre une bougie et une lampe électrique. La bougie éclaire grâce à la combustion, un processus simple et direct, mais limité en intensité et portée de lumière. C'est l'analogie de l'ordinateur classique, où les *bits*, telles des flammes, ne peuvent exister qu'en deux états : allumés ou éteints, (1) ou (0). La bougie, malgré son ancienneté, a son charme et ses utilisations, mais elle est limitée par sa nature physique.

Puis, il y a la lampe électrique, à première vue cet objet a le même but, qui est d'illuminer notre environnement. Cependant, la manière dont elle produit de la lumière est radicalement différente, exploitant les principes de l'électricité et de l'optique. Cette technologie permet une variation dans l'intensité, la couleur, et même de la direction de la lumière. L'ordinateur quantique, dans cette métaphore, est semblable à la lampe électrique, il n'est pas limité à des états binaires de (1) ou (0). Grâce au principe de la superposition, un *qubit* peut exister dans un état qui est à la fois (1) et (0), ce qui ouvre ainsi des possibilités immenses pour le traitement de l'information numérique.

La différence entre l'ordinateur classique et l'ordinateur quantique est donc bien plus qu'une question de vitesse ou d'efficacité. Il s'agit d'une différence de nature, comparable à celle séparant la bougie de la lampe électrique. Tandis que la bougie et la lampe visent toutes deux à repousser l'obscurité, la lampe utilise des principes et des possibilités qui étaient inimaginables pour les utilisateurs de la bougie.

2.3.1 Le mécanisme du calculateur conventionnel

Essentiellement, un ordinateur est composé d'un processeur, d'une mémoire vive et d'un disque dur. La mémoire vive ou la mémoire à accès aléatoire, *« RAM, Random Access Memory »* est un des éléments fondamentaux en architecture ordinateur. Il s'agit de l'espace de stockage rapide qui enregistre temporairement les programmes et instructions en cours. Cependant, la *RAM* fournit au processeur les données nécessaires pour les calculs en cours d'exécution. Un disque dur d'ordinateur *« HDD, Hard Disk Drive »* est un composant informatique qui a une capacité de stockage d'information digitale supérieure à la mémoire vive, mais plus long pour y accéder. Le disque dur stocke les données d'une manière continue contrairement à la mémoire vive. Le processeur ou l'unité centrale de traitement, *« CPU, Central Processing Unit »* est le composant autour duquel tout le reste est centré. Un processeur est un appareil électronique capable de manipuler des données d'une manière spécifiée par une séquence d'instructions. Ces derniers sont également appelés le code binaire. Cette séquence d'instructions peut être modifiée pour s'adapter à l'application et par conséquent, les ordinateurs ont une nature programmable.

Le code binaire est constitué de *bits*, signifie *binary digit*. Le *bit* est la plus petite unité d'information numérique pouvant être manipulée par un ordinateur. Afin de visualiser le fonctionnement de traitement de l'information numérique par ces derniers représentants un nombre naturel N sur un octet. Un octet se compose de huit *bits*, et nous pouvons représenter n'importe quel nombre entre 0 et 255 en utilisant un octet. Par exemple ; le nombre cent peut être représenté à l'aide d'un code binaire à huit *bits*, qui est (0110 0100). Dans un code binaire à huit *bits*, le *bit* le plus à gauche représente le signe du nombre et les sept autres *bits* représentent la grandeur du nombre. Si le *bit* le plus à gauche est 1, le nombre est négatif, et s'il est 0, le nombre est positif. Voici une représentation visuelle du concept (Fig. 2.5).

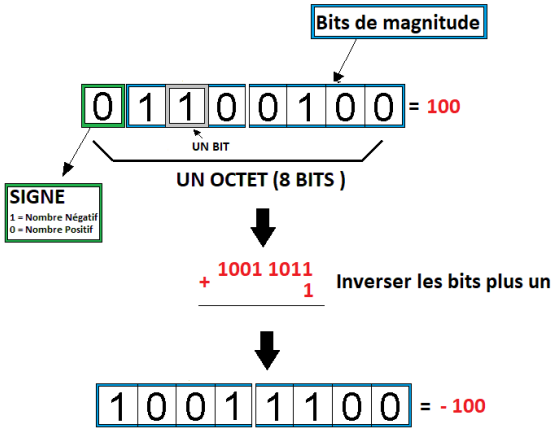


FIGURE 2.5 – Représentation des nombres binaires négatifs

Pour exprimer négativement le nombre cent dans notre représentation binaire, nous pouvons prendre le complément à deux de sa représentation binaire. Le complément à deux d'un nombre binaire est obtenu en inversant tous les *bits* de la représentation binaire, puis en ajoutant 1 au résultat. Dans ce cas, le complément à deux de (0110 0100) est (1001 1100).

Passons maintenant aux portes logiques qui sont un concept fondamental de l'informatique. Les portes logiques nous permettent de convertir différents courants ou entrées binaires en sorties, par exemple des opérations algébriques. Afin de bien comprendre ce concept, basons-nous sur les tables de vérité (Fig. 2.6). Chaque porte logique a un rôle précis. Par exemple ; la porte logique *AND* se base sur l'état des deux entrées binaires, parallèlement. Prenons $A = 0$ et $B = 1$ puis affectons la porte logique *AND* sur ces deux états. Le résultat de cette application est $A\ AND\ B = 0$, *ou* $A\ \&\&\ B = 0$. En conséquence, le résultat sera toujours nul, si l'une des entrées possède l'état (0). La porte logique *OR* est le contraire de *AND*.

Cependant, le résultat est toujours (1), si une des entrées a l'état (1), *A OR B = 1, ou A // B = 1.*

Les portes logiques

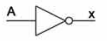
Nom	NOT	AND	NAND	OR	NOR	XOR	XNOR
Expr. Alg.	$\overline{A}$	AB	$\overline{AB}$	$A+B$	$\overline{A+B}$	$A \oplus B$	$\overline{A \oplus B}$
Symbole							

Table de vérité

A	X
0	1
1	0

B	A	X
0	0	0
0	1	0
1	0	0
1	1	1

B	A	X
0	0	1
0	1	1
1	0	1
1	1	0

B	A	X
0	0	0
0	1	1
1	0	1
1	1	1

B	A	X
0	0	1
0	1	0
1	0	0
1	1	0

B	A	X
0	0	0
0	1	1
1	0	1
1	1	0

B	A	X
0	0	1
0	1	0
1	0	0
1	1	1

FIGURE 2.6 – La table de vérité et les symboles des portes logiques

Les *transistors* sont les composants fondamentaux de toutes les micropuces, y compris l'unité centrale de traitement. Ces dernières nous permettent de construire les portes logiques et de former le code binaire utilisé par l'ordinateur afin de traiter la logique booléenne. Différentes configurations de *transistors* réalisent divers types de portes logiques, qui peuvent se combiner en réseaux appelés demi-additionneurs, qui se combinent également en additionneurs complets. Ces derniers relèvent tous les deux de la catégorie des circuits logiques combinatoires utilisés pour les opérations arithmétiques. La principale différence entre ces circuits logiques, est que le demi-additionneur fonctionne sur deux entrées et l'additionneur complet fonctionne sur trois entrées (Fig. 2.7).

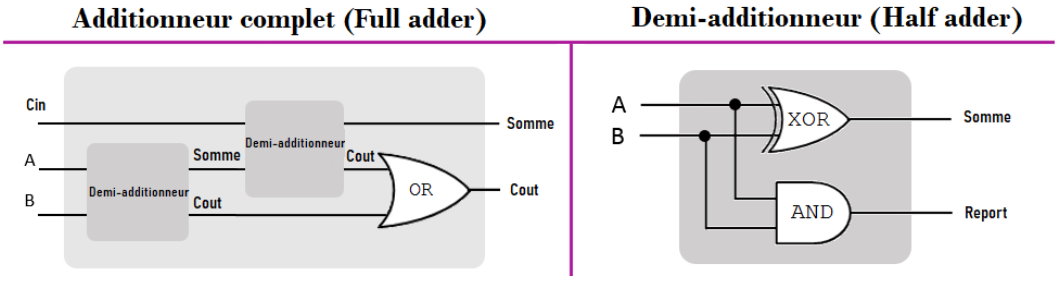


FIGURE 2.7 – Schéma représentatif du demi-additionneur et de l'additionneur complet

Les instructions dans un ordinateur sont une séquence binaire. Lors de l'exécution de ces derniers par un processeur, génèrent des actions numériques telles que les applications, les systèmes d'exploitations. Une bonne analogie est le mécanisme d'une boîte à musique. Une boîte à musique dispose d'un tambour rotatif avec de petites bosses et une rangée de dents. Au fur et à mesure que le tambour tourne, différentes broches sont activées à leur tour par les bosses. Tout ce processus génère des actions, dans ce cas l'action est la musique. D'une façon similaire, les motifs binaires des instructions alimentent l'unité d'exécution du processeur. Différents modèles de *bits* activent ou désactivent différentes parties du cœur de traitement. Ainsi, la configuration binaire d'une instruction donnée peut activer des opérations arithmétiques, tandis qu'une autre peut sauvegarder l'information digitale d'une manière définitive ou temporaire. Le processeur à lui seul est incapable d'effectuer avec succès les tâches numériques. Il nécessite de la mémoire pour le stockage des programmes et des données, de la mémoire vive pour faciliter la manipulation et l'utilisation de ces derniers ; puis d'au moins un périphérique *d'E/S périphérique d'entrée/sortie*, utilisé pour transférer des données entre l'ordinateur et le monde extérieur (Fig. 2.8).

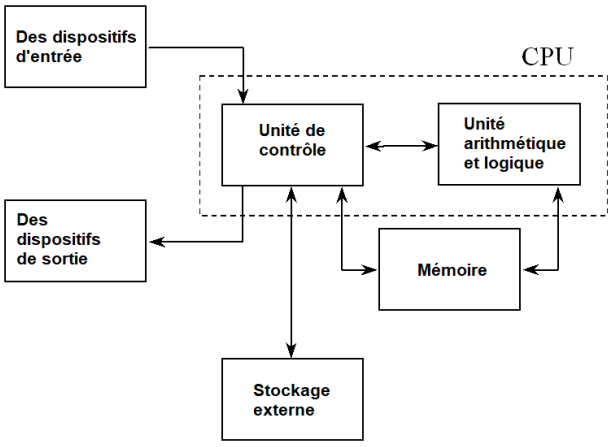


FIGURE 2.8 – Schéma représentant l'architecture générale des ordinateurs classiques

2.3.2 Les limites physiques des transistors et l'effondrement de la loi de Gordon Moore

L'observation de Gordon Moore, connue sous le nom de loi de Moore, stipule que le nombre de transistors sur une puce double environ tous les deux ans, ce qui a historiquement contribué à la croissance exponentielle de la puissance et de la vitesse de calcul (Fig. 2.9) [34]. Cependant, les limites de la technologie des transistors conventionnels sont approchées car la taille des transistors s'est réduite à une échelle de 10 nanomètres, et les rendre plus petits n'est plus pratique.

À mesure que la taille des transistors diminue, il y a deux défis principaux : minimiser la chaleur générée par le système et minimiser les effets tunnel quantiques. À mesure que la taille du transistor diminue, la quantité de chaleur générée par le système augmente, ce qui peut entraîner des fuites électriques et une instabilité. De plus, à de très petites tailles, les électrons peuvent traverser la grille du transistor, ce qui peut entraîner des erreurs dans les calculs. Ces défis rendent de plus en plus difficile le maintien de la précision et de la stabilité dans les systèmes informatiques conventionnels à base de transistors.

Selon le physicien théoricien Michio Kaku, la loi de Moore s'effondrera probablement d'ici la fin de la prochaine décennie et il ne sera plus possible de compter uniquement sur la technologie conventionnelle à base de transistors pour maintenir le progrès technologique [35]. Cela nécessitera une évolution vers des architectures informatiques alternatives, telles que l'informatique quantique, pour poursuivre le développement de la technologie informatique. La fin de la loi de Moore marquera donc le début d'une nouvelle ère dans les technologies de l'information, une ère qui nécessitera de nouvelles architectures informatiques plus complexes pour continuer à progresser [48].

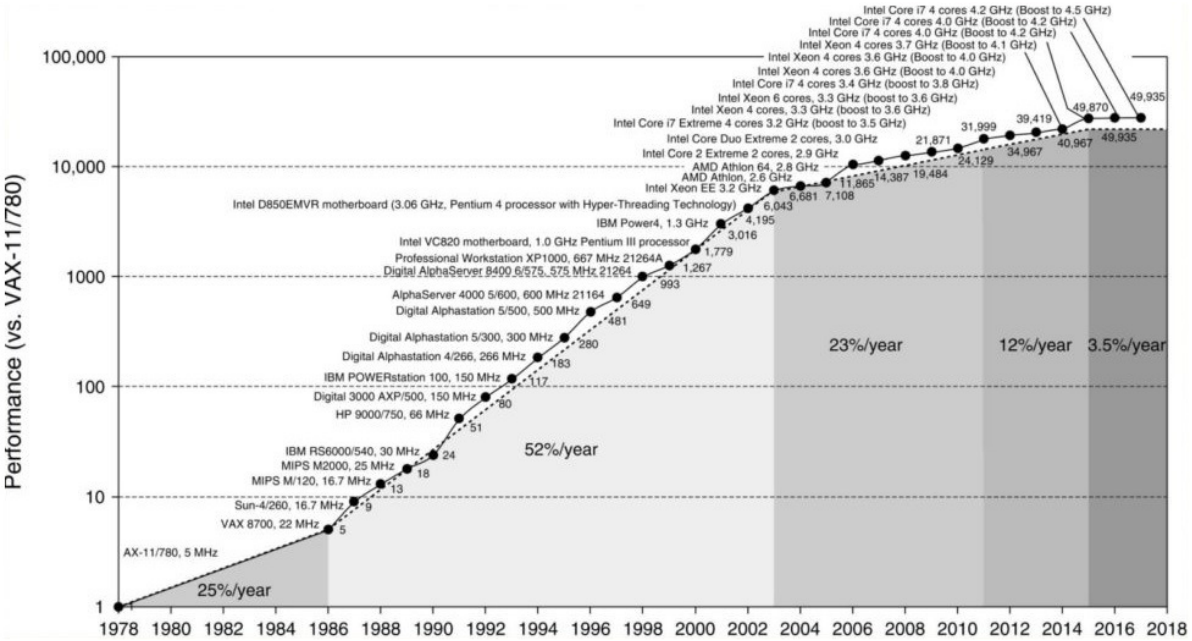


FIGURE 2.9 – Graphe représentant les performances des processeurs (1978 - 2018)

2.3.3 Le problème de la croissance exponentielle et les problèmes insolubles

La croissance exponentielle est un phénomène naturel qui se manifeste sous la forme algébrique $x_t = x_0(1+r)^t$, où x_0 représente l'état initial du système, r est le taux de croissance et t réfère au nombre d'intervalles de temps. Par exemple ; la croissance exponentielle des microorganismes, la réaction nucléaire en chaîne, la théorie de la complexité $E = DTIME(2^{0(n)})$ [43] sont tous des systèmes basés sur la croissance exponentielle. L'une des caractéristiques les plus importantes des systèmes exponentiels est que, bien que l'état initial de ces derniers progresse lentement, ils peuvent se traduire assez rapidement par des valeurs énormes ingérables. Afin d'expliquer ce concept, referons-nous au problème du blé et d'échiquier *Wheat and chessboard problem* [38]. Un échiquier se compose de 64 cases, imaginons que l'on veut mettre un seul grain de blé sur la première case de l'échiquier puis doubler le nombre sur les prochaines cases selon le nombre précédent (Fig. 2.10). Le problème peut être résolu par une simple addition. La somme des grains sur les 64 cases ($1 + 2 + 4 + 8 + 16...$). On conclut que le nombre total de grains est de $2^{64} - 1$, soit plus de 2000 fois la production mondiale annuelle de blé, ce qui est énorme. Cette référence nous permet d'illustrer la puissance de la croissance exponentielle et la progression simple, mais robuste du système.

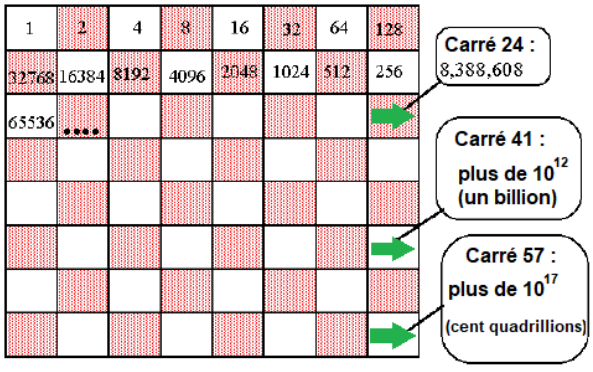


FIGURE 2.10 – Représentation de la croissance exponentielle du riz sur a jeux d'échiquier

Un algorithme qui s'exécute en temps exponentiel présente des difficultés majeures pour résoudre, dans un intervalle de temps raisonnable, des problèmes de grande taille. Ce type de complexité computationnelle est souvent considéré comme pratiquement insoluble, car le temps nécessaire pour résoudre le problème croît de manière exponentielle avec la taille de l'entrée. Pour une entrée de taille N, il faut 2^N microsecondes, ce qui signifie que le temps de calcul pour examiner toutes les possibilités augmente exponentiellement. Bien qu'une microseconde soit une unité de temps extrêmement brève, correspondant à quelques milliers d'instructions sur un ordinateur moderne, l'effet exponentiel devient rapidement prohibitif. Par exemple ; pour N=50, le temps nécessaire serait d'environ 35 ans de calcul ininterrompu avec une puissance de calcul constante. Si N accroit à 70, cela se traduirait théoriquement par approximativement 37 millions d'années de calcul continu, soulignant l'impraticabilité de tels algorithmes pour de grandes tailles d'entrée. Cependant, ces estimations sont théoriques et dépendent de la puissance de calcul disponible.

2.4 Principes et propriétés d'ordinateur quantique

L'informatique quantique est un domaine de l'informatique basée sur les principes de la théorie quantique. Les ordinateurs utilisés aujourd'hui sont limités, car ils utilisent le code binaire classique. L'informatique quantique utilise des *bits* quantiques ou des *qubits*. Grâce à la capacité unique des particules subatomiques, leur permettant d'exister dans plus d'un état à la fois, les ordinateurs quantiques expriment l'état (0) et (1) simultanément. L'informatique quantique a le potentiel d'arriver à la suprématie quantique qui a comme but de démontrer que les ordinateurs quantiques sont capables de résoudre un problème mathématique complexe qu'aucun ordinateur classique peut résoudre en un temps raisonnable. La simulation présentée (Fig. 2.11) montre une comparaison de la croissance de la puissance de calcul entre ordinateurs quantiques et classiques, utilisant une échelle logarithmique pour faciliter la visualisation et la comparaison des deux types de croissance. Cette échelle est particulièrement utile pour représenter la croissance exponentielle, qui est rendue sous forme de ligne droite, rendant les tendances plus perceptibles sur une large gamme de valeurs. Bien que cette visualisation ne capture pas toutes les nuances du développement technologique réel, elle illustre clairement l'avantage théorique majeur de la puissance de calcul quantique, qui augmente exponentiellement avec le nombre de *qubits*, par opposition à l'augmentation linéaire de la puissance de calcul classique en fonction du nombre de transistors. Il est important de noter que les représentations graphiques telles que celles-ci simplifient souvent la réalité pour mettre en lumière des principes sous-jacents, comme illustré dans l'annexe A, qui contient le code utilisé pour générer la simulation.

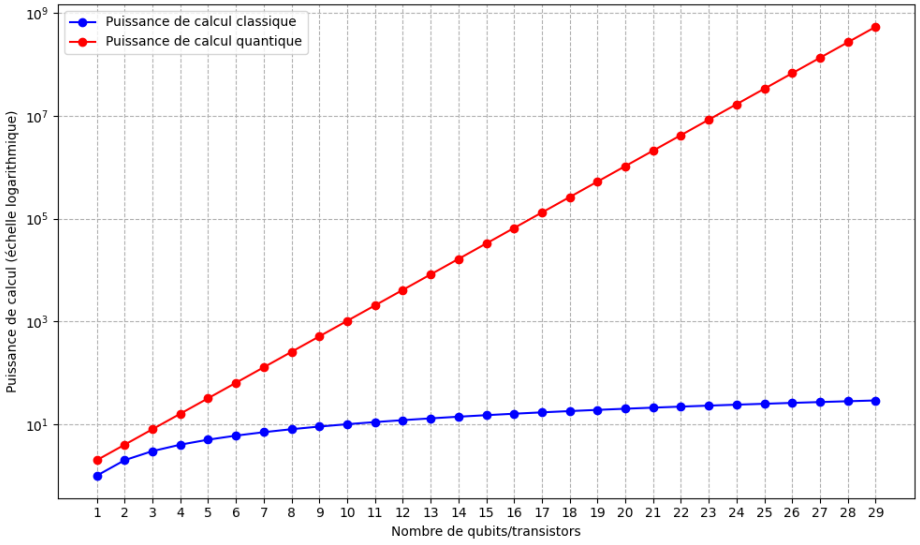


FIGURE 2.11 – Simulation : Croissance Calculatoire Classique vs Quantique

Dans cette partie, il sera abordé d'une manière approfondie les principes et les propriétés de l'ordinateur quantique, ce qui permettra à comprendre sa puissance, son potentiel et son impact sur la technologie. Une bonne compréhension du concept aidera également à résoudre l'une des problématiques de ce mémoire qui est la factorisation première. Cette résolution permettra la démonstration de la faiblesse du cryptosystème RSA face aux ordinateurs quantiques et l'instabilité que le monde de la cybersécurité va voir dans le futur proche, (l'explication détaillée de ces concepts est abordée dans le chapitre III).

2.4.1 Les fondements des nombres complexes et imaginaires dans l'espace de Hilbert pour un qubit

Avant de plonger dans l'univers fascinant de l'espace de Hilbert, il est essentiel de définir les nombres complexes et imaginaires. Ces derniers constituent la base sur lequel repose une grande partie de la physique quantique. Un nombre complexe z, où $z = a + bi$ combine une partie réelle a et une partie imaginaire bi, cette dernière étant un multiple de l'unité imaginaire i, où $i^2 = -1$ (Fig. 2.12). Cette structure permet l'élaboration de descriptions et des solutions à des problèmes qui seraient inaccessibles avec les nombres réels.

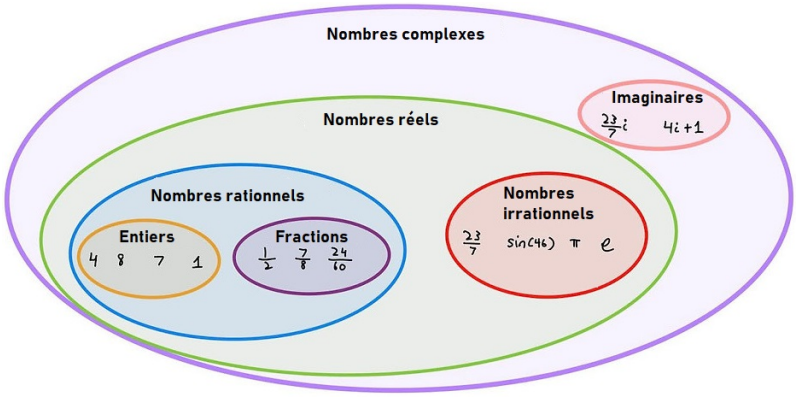


FIGURE 2.12 – Hiérarchie et classification des nombres en mathématiques

Les nombres complexes nous permettent d'explorer des dimensions supplémentaires en mathématiques, ce qui est particulièrement utile dans le contexte de l'espace de Hilbert. Ce dernier est un concept fondamental en mathématiques, cruciale notamment en physique quantique. On peut l'imaginer comme un cadre très général et flexible, idéal pour appréhender des systèmes et des phénomènes mathématiques complexes. Au cœur de l'espace de Hilbert se trouvent deux notions clés : l'espace vectoriel et le produit interne.

Un espace vectoriel, dans ce contexte, peut être vu comme un vaste terrain de jeu où l'on peut effectuer librement des opérations telles que l'addition ou la multiplication par un nombre complexe. Les vecteurs, éléments indispensables de cet espace, peuvent représenter une multitude de concepts, y compris des états quantiques.

Le produit interne, quant à lui, offre une méthode pour mesurer et comparer les « signatures » uniques de chaque vecteur dans l'espace de Hilbert, facilitant ainsi la compréhension des interactions ou des comparaisons entre vecteurs. Plus précisément, il nous permet de définir une « norme », qui mesure la longueur ou la taille d'un vecteur selon la formule $\|x\| = \sqrt{\langle x, x \rangle}$, où x est un vecteur de cet espace [52].

Ce cadre s'avère particulièrement pertinent lorsqu'on l'applique à la physique quantique, en particulier dans l'étude des *qubits*. Un *qubit*, l'unité de base de l'information quantique, tire avantage des propriétés des nombres complexes dans l'espace de Hilbert pour représenter des états qui vont bien au-delà du binaire traditionnel (0) et (1). Grâce à la superposition quantique, un *qubit* peut exister dans un état qui est une combinaison complexe de (0) et (1), offrant ainsi une capacité de traitement d'informations supérieure.

L'intégration des nombres complexes et imaginaires dans l'espace de Hilbert fournit un puissant langage mathématique pour décrire et manipuler les états quantiques des *qubits*. Cette synergie ouvre la porte à une compréhension plus profonde et à des avancées technologiques révolutionnaires en informatique quantique.

Considérons deux vecteurs dans $\mathbb{R}^2$, $\vec{a}$ et $\vec{b}$, avec :

$$\vec{a} = (a_1, a_2), \quad \vec{b} = (b_1, b_2).$$

Leur produit scalaire est donné par :

$$\langle \vec{a}, \vec{b} \rangle = a_1 b_1 + a_2 b_2.$$

Par exemple ; si $\vec{a} = (3, 4)$ et $\vec{b} = (1, 2)$, alors

$$\langle \vec{a}, \vec{b} \rangle = 3 \times 1 + 4 \times 2 = 11.$$

L'état d'un système quantique est spécifié par une fonction d'onde dans un espace de Hilbert complexe H. La fonction d'onde est notée en notation de Dirac par $|\Psi\rangle$, cette expression quantique est connue sous le nom de *ket*. La fonction d'onde $|\Psi\rangle(r, t)$, est liée à la probabilité d'état d'un système quantique à la position r , au temps t. Diverses fonctions d'onde de la mécanique quantique qui représentent l'état d'une particule quantique existent dans l'espace complexe H.

L'espace euclidien peut résoudre presque toutes les fonctions, mais il a des limites dimensionnelles. En d'autres termes, l'espace euclidien est considéré comme un espace de dimension finie. L'espace de Hilbert est une extension de l'espace euclidien qui est un espace de dimension infinie. L'application la plus connue de l'espace de Hilbert est la mécanique quantique. En utilisant des vecteurs propres, les espaces de Hilbert offrent la possibilité d'expliquer et de traiter le comportement des particules quantiques. L'espace de Hilbert pour un *qubit* est bidimensionnel, ce qui signifie que l'ensemble des vecteurs qui lui appartiennent peut être représenté par une combinaison linéaire de deux vecteurs $\langle 0|0\rangle = 1, \quad \langle 0|1\rangle = \langle 1|0\rangle = 0$ [41]. Le traitement de l'information quantique nécessite des transformations unitaires opérant sur des états à un et deux *qubits*, appelés les portes logiques quantiques (ce concept sera expliqué en détails dans la partie 2.4.7).

2.4.2 Les qubits, le produit et δ Kronecker

Un *qubit*, ou *bit* quantique, est l'unité fondamentale de l'informatique quantique [4]. Les ordinateurs conventionnels sont construits par des *transistors* qui sont soit actifs ou inactifs. Les ordinateurs quantiques sont construits en utilisant des particules subatomiques, telles que des électrons ou des ions, qui sont manipulées pour représenter des qubits. Ces particules peuvent être dans une variété d'états, tels que les niveaux d'énergie ou les états de *spin*, qui peuvent être utilisés pour coder de l'information. Cependant, il est important de noter qu'il existe également d'autres types d'architectures d'ordinateurs quantiques qui ne reposent pas sur des particules subatomiques. Par exemple, certains systèmes d'ordinateurs quantiques sont construits en utilisant des circuits supraconducteurs (atomes artificiels), qui peuvent également être utilisés pour créer et manipuler des qubits. De plus, il existe des ordinateurs quantiques qui utilisent d'autres systèmes physiques, tels que des qubits topologiques (anyons). Les *qubits* sont imprévisibles, cependant l'état du *qubits* est déterminé le moment de la mesure ou de l'observation. On peut représenter mathématiquement les deux états des deux vecteurs orthogonaux dans l'espace de Hilbert :

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}; \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{2.4}$$

Il est également possible de construire l'espace de Hilbert pour des systèmes contenant deux *qubits* ou plus. Pour un système à deux *qubits*, la dimension de l'espace de Hilbert est de 4×4, car il est composé de vecteurs et de matrices, calculé à l'aide du produit Kronecker de chaque vecteur et matrice pour le *qubit* individuel :

$$\begin{bmatrix} a \\ b \end{bmatrix} \otimes \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} a \begin{bmatrix} c \\ d \end{bmatrix} \\ b \begin{bmatrix} c \\ d \end{bmatrix} \end{bmatrix} = \begin{bmatrix} ac \\ ad \\ bc \\ bd \end{bmatrix}. \tag{2.5}$$

Voici la démonstration mathématique :

$$\{|0\rangle, |1\rangle\} \otimes \{|0\rangle, |1\rangle\} = \{|00\rangle, |01\rangle, |10\rangle, |11\rangle\} \tag{2.6}$$

$$|00\rangle \equiv \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}; \quad |01\rangle \equiv \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}; \quad |10\rangle \equiv \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}; \quad |11\rangle \equiv \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}. \tag{2.7}$$

Un *qubit* représente plus d'informations qu'un *bit* classique. Un nombre de n *qubits* représentent 2^n *bits*, ce qui est énorme (Fig. 2.13). Les *transistors* peuvent être enchaînés pour former des portes logiques qui effectuent les calculs classiques. Les électrons dans l'état d'alternance peuvent aussi être enchaînés pour former les portes quantiques qui effectuent des calculs quantiques. L'informatique quantique s'appuie sur deux propriétés des *qubits*, la superposition et l'intrication, ainsi que sur la stabilisation et la gestion de l'état des *qubits* avec la correction du bruit. Grâce à ces concepts, les ordinateurs quantiques sont exponentiellement plus puissants que les plus grands superordinateurs classiques.

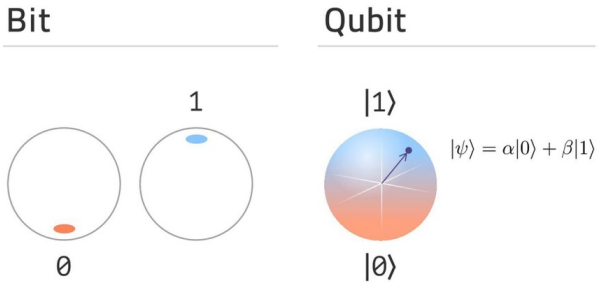


FIGURE 2.13 – Représentation d'un *qubit* et d'un *bit*

Le delta de Kronecker, noté δ_{ij}, est un symbole mathématique qui nous aide à déterminer si deux indices, i et j, sont égaux l'un à l'autre. Il a une définition générale donnée par :

$$\delta_{ij} = \begin{cases} 1 & \text{si } i = j \\ 0 & \text{si } i \neq j. \end{cases} \tag{2.8}$$

Prenons quelques exemples pour illustrer la définition générale du delta de Kronecker :

- δ_{23} : Ici, $i = 2$ et $j = 3$. Puisque i et j ne sont pas égaux ($2 \neq 3$), le delta de Kronecker δ_{23} prend la valeur 0. Donc, $\delta_{23} = 0$.
- δ_{22} : Dans ce cas, $i = 2$ et $j = 2$. Puisque i et j sont égaux ($2 = 2$), le delta de Kronecker δ_{22} prend la valeur 1. Donc, $\delta_{22} = 1$.

Le delta de Kronecker suit également certaines propriétés utiles, telles que :

— Propriété de multiplication : lorsque le delta de Kronecker est multiplié par lui-même, il conserve sa valeur si les indices sont égaux. Si i, j, et k sont des indices, alors la multiplication est donnée par :

$$\delta_{ij}\delta_{jk} = \delta_{ik} \quad \text{si } i = k \text{ et } j \text{ est égal à } i \text{ et à } k. \tag{2.9}$$

Dans tous les autres cas où i, j, et k ne sont pas tous égaux, le produit est zéro.

Le delta de Kronecker est utile lorsque nous travaillons avec des matrices ou des tenseurs et que nous devons effectuer des opérations qui impliquent la sommation sur des indices. Par exemple, considérons le produit de deux matrices A et B où B est une matrice diagonale avec des éléments b_{jj} sur la diagonale. Les éléments hors diagonale de B sont nuls, ce qui peut être exprimé en utilisant le delta de Kronecker : $B_{jk} = b_{jj}\delta_{jk}$. Le produit matriciel AB peut alors être écrit comme suit :

$$(AB)_{ik} = \sum_j A_{ij}B_{jk} = \sum_j A_{ij}b_{jj}\delta_{jk}. \tag{2.10}$$

Ici, la somme sur j élimine tous les termes où $j \neq k$ à cause de la présence de δ_{jk}, et nous sommes laissés avec :

$$(AB)_{ik} = A_{ik}b_{kk}. \tag{2.11}$$

Ceci illustre comment le delta de Kronecker sélectionne les éléments diagonaux de B lors du produit matriciel. Dans le contexte des *qubits* et du produit tensoriel (pas Kronecker), le delta de Kronecker n'est pas directement utilisé, mais le concept d'éléments diagonaux est analogue à l'interaction entre *qubits* dans l'espace de Hilbert des systèmes quantiques.

2.4.3 La superposition, l'intrication quantique et les états quantiques de Bell

Pour comprendre le concept de la superposition [32], considérons un système classique simple à deux états, en supposant un interrupteur qui peut être activé ou désactivé. Cependant, un système quantique à deux états est intégralement différent. À Chaque observation, l'état du système quantique est soit allumé ou éteint, tout comme un système classique. Mais avant chaque observation, le système quantique peut-être dans une superposition d'états actifs et inactifs à la fois (Fig. 2.14). Cela semble contre-intuitif. D'autre part, il est impossible d'échapper au fait que ce mystérieux phénomène est réel et naturel. Pourtant, notre monde lui-même est quantique et incertain. C'est pourquoi les systèmes quantiques ne peuvent pas être modélisés sur un ordinateur classique. Toutefois, ce concept permet la résolution des problèmes mathématiques complexes d'une nature qu'un ordinateur conventionnel n'est pas capable de résoudre efficacement.

En général, les physiciens considèrent qu'il est inutile de parler de l'état d'un système quantique, avant de le mesurer ou de l'observer. Cependant, il est important de noter que l'interaction entre deux systèmes quantiques ne conduit pas automatiquement à leur intrication. Bien que certaines interactions puissent induire une intrication, phénomène où les états de ces systèmes deviennent si intimement liés que l'état de l'un ne peut être pleinement déterminé sans connaître l'état de l'autre, cela n'est pas une conséquence systématique de toute interaction. Dans un état d'intrication quantique, la mesure de l'état quantique de l'un des systèmes révèle immédiatement des informations sur l'état de l'autre système, et ce, avec une précision absolue, quelle que soit la distance qui les sépare. Cette caractéristique défie les notions classiques de localité et d'indépendance des systèmes physiques. La localité, en physique classique, suggère que des objets distants ne peuvent pas avoir une influence instantanée l'un sur l'autre ; leurs interactions sont limitées par la vitesse de la lumière. L'indépendance, quant à elle, implique que l'état d'un système peut être décrit sans référence à l'état d'un autre système non lié. L'intrication quantique, en permettant une

corrélation instantanée entre des systèmes sur de grandes distances, remet donc en question ces principes, ouvrant la porte à une nouvelle compréhension de l'univers à la fois complexe et fascinante [2].

Supposons deux électrons, A et B qui sont dans un état d'intrication. Si le spin de A est vers le haut, le spin de B sera instantanément opposée à la rotation de l'Électron A d'une manière réciproque. Cela implique un degré de certitude que le spin de l'Électron B est à cent pourcents vers le bas. Bien qu'on sépare ces électrons des années-lumière, le résultat sera toujours le même. Changer l'état d'un *qubit* intriqué changera immédiatement l'état des *qubits* intriqués. Par conséquent, l'intrication améliore la vitesse de traitement des ordinateurs quantiques. Doubler le nombre de *qubits* ne doublera pas nécessairement le nombre de processus puisque le traitement d'un *qubit* exprime des informations sur plusieurs *qubits* intriqués. Toutefois, l'intrication quantique est nécessaire pour qu'un algorithme quantique offre une accélération exponentielle par rapport aux calculs classiques.

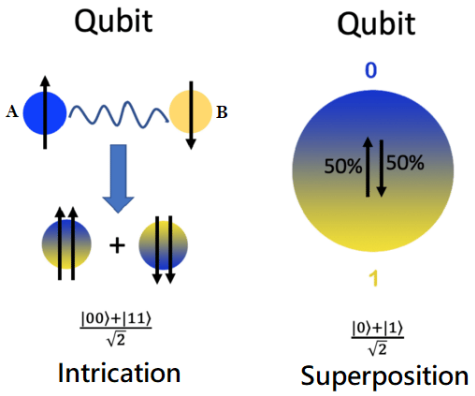


FIGURE 2.14 – Illustration de la superposition quantique et de l'intrication quantique

Le concept paire de Bell fait référence à l'un des quatre états quantiques intriqués à deux *qubits*, connus collectivement sous le nom de quatre états de Bell [3]. Deux des états de Bell entraînent la même superposition, de sorte que les deux *qubits* se retrouvent dans le même état lorsqu'ils sont mesurés. Les deux autres paires de Bell donnent une superposition égale, de sorte que les deux *qubits* se retrouvent dans des états opposés lorsqu'ils sont mesurés. Autrement dit, si le premier *qubit* est mesuré en $|0\rangle$, alors le deuxième *qubit* est mesuré en $|1\rangle$ ainsi de suite. Pour chaque paire de Bell, si un *qubit* est mesuré, nous pouvons déterminer l'état de l'autre *qubit* avec une grande certitude. Les deux paires de Bell $|\Psi^+\rangle$ et $|\Psi^-\rangle$ sont les deux cas où les deux *qubits* doivent se retrouver dans l'état opposé lorsqu'ils sont mesurés. D'un autre côté, les deux paires de Bell $|\Phi^+\rangle$ et $|\Phi^-\rangle$ sont les deux cas où les deux *qubits* doivent se retrouver dans la même état quand ils sont mesurés (Tab. 2.1).

Paires de Bell	**Représentation mathématique**
$\|\Phi^+\rangle$	$\frac{\|00\rangle+\|11\rangle}{\sqrt{2}}$
$\|\Phi^-\rangle$	$\frac{\|00\rangle-\|11\rangle}{\sqrt{2}}$
$\|\Psi^+\rangle$	$\frac{\|01\rangle+\|10\rangle}{\sqrt{2}}$
$\|\Psi^-\rangle$	$\frac{\|01\rangle-\|10\rangle}{\sqrt{2}}$

TABLE 2.1 – Tableau des représentations mathématiques des paires de Bell

2.4.4 L'effet tunnel et les fonctions d'ondes

En mécanique classique, une particule x, avec une énergie insuffisante, ne peut pas franchir une barrière de potentiel y, cette dernière sera capable de la franchir quand elle possédera une énergie supérieure à l'obstacle $E_x > E_y$. Cependant, dans le monde quantique, les particules sont associées à des fonctions d'ondes. En rencontrant une barrière, une fonction d'onde ne s'annule pas brusquement ; au contraire, son amplitude diminue d'une façon exponentielle (Fig. 2.15). Cette baisse d'amplitude correspond à une diminution de la probabilité de trouver une particule plus loin dans la barrière. Si cette dernière a une faible densité, l'amplitude peut être non nulle de l'autre côté. Cela implique qu'il est possible que certaines particules puissent traverser la barrière, toutefois, cette probabilité s'affaiblie tant que la densité de cette dernière augmente [8]. L'effet tunnel est défini comme le rapport de densité volumique de courant sortant de la barrière divisée par la densité volumique de courant incident sur la barrière. Si ce coefficient de transmission est une valeur non nulle, alors il y a une probabilité qu'une des particules la traverse. L'effet tunnel quantique est ce qui donne aux ordinateurs quantiques le potentiel d'accomplir des tâches d'une manière exponentielle ; mais également d'effectuer des fonctions, que les ordinateurs classiques ne peuvent gère accomplir à cause des limitations de la physique classique.

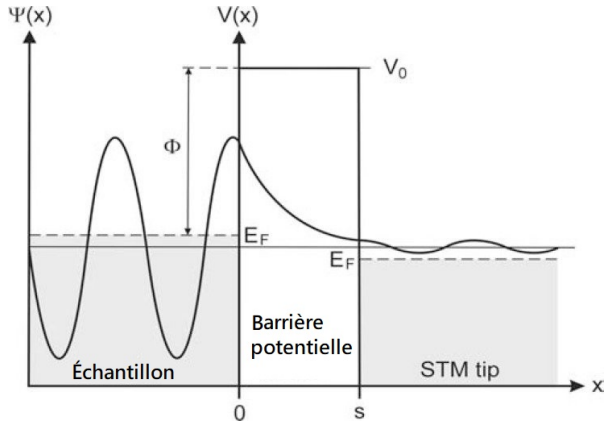


FIGURE 2.15 – Illustration de l'effet tunnel quantique

2.4.5 Les opérateurs unitaires, adjoints et autoadjoints

Une matrice A est unitaire si $UU^{\dagger} = I$, $U^{\dagger} = U^{-1}$ où $U^{\dagger}$ est la transposée conjuguée de U et I est la matrice d'identité.

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \tag{2.12}$$

Une porte quantique de n *qubits* est représentée par une matrice unitaire de taille $2^n \times 2^n$, qui applique une transformation spécifique à ces n *qubits*. Pour simplifier, imaginez une matrice comme un tableau de nombres qui, lorsqu'il est appliqué à un ensemble de *qubits*, change leur état d'une manière très précise.

Les matrices qui sont égales à leur propre conjugaison transposée, connues sous le nom de matrices hermitiennes [9], jouent un rôle crucial ici. Concrètement, si le complexe conjugué de tous les éléments d'une matrice (c'est-à-dire, changer le signe de la partie imaginaire de chaque élément) et puis transposer cette matrice (échanger les lignes et les colonnes), on obtient la matrice originale. Cette propriété est exprimée mathématiquement comme $A^{\dagger} = A$ où $A^{\dagger}$ est la conjugaison transposée de la matrice A. Parmi les portes quantiques de base, certaines comme la porte de Hadamard (H) et les portes de Pauli (X, Y, Z) sont des exemples de ces opérateurs hermitiens. Elles ont la particularité d'être leurs propres inverses ; appliquer deux fois de suite la même porte ne change pas l'état du *qubit*.

Les portes S et T sont unitaires, ce qui signifie qu'elles préservent la longueur des vecteurs d'état qu'elles transforment, un critère crucial pour les opérations quantiques. Cependant, contrairement aux opérateurs hermitiens, appliquer une porte S ou T puis son opération inverse (son adjoint) [21] ne ramène pas simplement au point de départ de la manière la plus directe comme le ferait un opérateur hermitien avec lui-même. En d'autres termes, même si les portes S et T ne reflètent pas cette symétrie miroir des opérateurs hermitiens (où l'opérateur est égal à son propre adjoint), elles jouent un rôle vital dans la manipulation des états quantiques.

2.4.6 Les portes logiques quantiques simples et la sphère de Bloch

Toutes les simulations des portes logiques quantiques présentées ont été méticuleusement élaborées avec l'outil *Q-CTRL*. Cette visualisation, améliore la compréhension des interactions subtiles et des fonctionnements internes des portes quantiques.

Les portes logiques quantiques nous permettent de contrôler l'état quantique d'un *qubit* ou d'un groupe de *qubits*. Les portes quantiques ressemblent aux portes informatiques classiques qui fonctionnent sur une base numérique. Différents circuits quantiques permettent d'utiliser différents algorithmes quantiques. Pourtant, Les portes quantiques sont les éléments fondamentaux de tous les calculs quantiques. L'état d'un *qubit* superposé est représenté avec deux nombres complexes qui invoquent la probabilité que ce dernier soit un (1) ou zéro (0) [23]. Cependant, la somme des probabilités doit toujours être égale à un.

$$|\Psi\rangle = \alpha|+1/2\rangle + \beta|-1/2\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}; \quad |+1/2\rangle = |0\rangle \quad |-1/2\rangle = |1\rangle; \quad \alpha, \beta \in \mathbb{C} \tag{2.13}$$

α et β représentent l'amplitude de la probabilité de densité :

$$|\alpha|^2 + |\beta|^2 = 1; \quad |\alpha\rangle = \cos\frac{\theta}{2} \quad |\beta\rangle = e^{i\phi}\sin\frac{\theta}{2}, \tag{2.14}$$

où θ est un angle polaire et ϕ est un angle azimutal. Après la mesure, l'état quantique du système s'effondre dans l'un des états suivants :

$$|\Psi_0\rangle = \frac{\alpha}{|\alpha|}|0\rangle \quad \text{ou} \quad |\Psi_1\rangle = \frac{\beta}{|\beta|}|1\rangle. \tag{2.15}$$

Nous pouvons représenter toutes les variations d'états du *qubit* données par une sphère tridimensionnelle unitaire [49]. Par définition, la sphère de Bloch qui est considéré comme un diagramme standard utilisé pour représenter la position et le changement d'état du *qubit* (Fig. 2.16). Néanmoins, tout point sur cette sphère représente une combinaison linéaire des états zéro (0) et un (1) avec des coefficients complexes. Une impulsion ($\frac{\pi}{2}$) permet d'effectuer la rotation d'un *qubit* de l'état nul à l'état de superposition. Les rotations d'un *qubit* sont mesurées en *radians*.

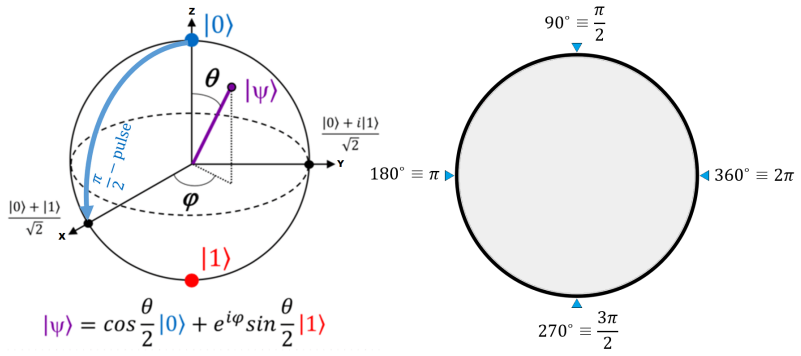


FIGURE 2.16 – Représentation de la sphère de Bloch et le diagramme des rotations d'un *qubit* en radians

2.4.6.1 Les portes logiques quantiques Pauli X,Y et Z

En mécanique quantique, les trois portes de Pauli sont représentées par la lettre grecque σ et définissent les états observables du spin en mécanique quantique. Seuls les états propres de ces matrices sont considérées comme des valeurs de mesure identifiables lors de l'interaction avec des objets quantiques. Les trois matrices correspondent à des mesures effectuées dans trois directions perpendiculaires [16]. La porte logique quantique X « σ_x » (Fig. 2.17) suit la même logique que l'opérateur *NOT* classique (Fig. 2.6). L'application de la porte logique *NOT* sur un *bit* nul (0) donne un *bit* non nul (1) réciproquement. Voici une représentation de la porte logique X sur les deux états d'un *qubit* :

$$X|0\rangle = |1\rangle\,; \quad X|1\rangle = |0\rangle. \tag{2.16}$$

Il est important de comprendre d'une manière mathématique la logique de la porte X, puis prouver l'égalité de cette dernière. On peut donc la représenter comme une matrice logique de dimensions 2×2 :

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{2.17}$$

La transformation qu'une porte logique émet sur l'état du *qubit* peut-être décrit comme la multiplication de la matrice de la porte logique par le vecteur d'état. Voici la démonstration mathématique qui décrire le changement :

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle \tag{2.18}$$

$$X|1\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle. \tag{2.19}$$

On peut visuellement décrire le circuit et la rotation du *qubit* après l'affectation de la porte logique X sur la sphère de Bloc .

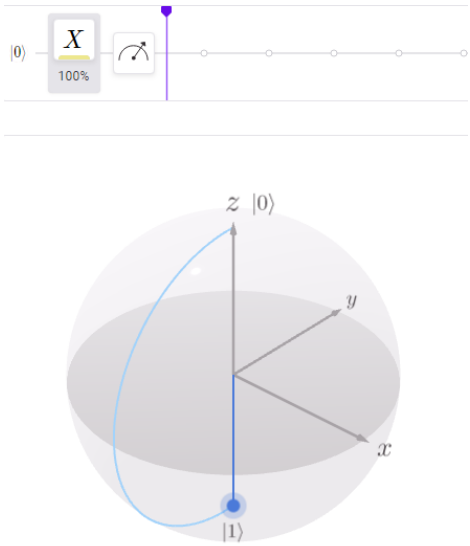


FIGURE 2.17 – Simulation de la porte X sur la sphère de Bloch

Le résultat de la porte Pauli Y « σ_y » (Fig. 2.18) est le même que celui de la porte Pauli X. Cependant la porte X se déplace dans l'espace réel, tandis que la porte Y se déplace dans l'espace imaginaire « i ».

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}. \tag{2.20}$$

La représentation mathématiquement de l'état du *qubit* après une application de la porte logique Y, est démontré ci-dessous :

$$Y|0\rangle = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ i \end{bmatrix} = +i|1\rangle \tag{2.21}$$

$$Y|1\rangle = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -i \\ 0 \end{bmatrix} = -i|0\rangle. \tag{2.22}$$

Le circuit et la rotation du *qubit* après l'affectation de la porte logique Y est représenté :

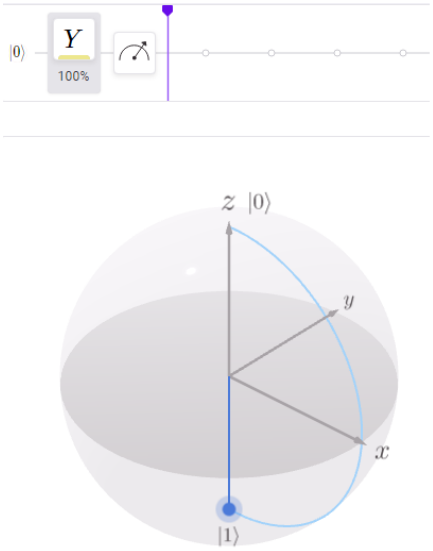


FIGURE 2.18 – Simulation de la porte Y sur la sphère de Bloch

La porte Z « σ_z » (Fig. 2.19) fait tourner le vecteur d'état autour de l'axe de « z ». Pour réaliser cela, la porte Z fait une rotation de π radians équivalent à 180 degrés. Toutefois, cette manipulation inverse la phase du qubit. On peut définir la porte Z sous la forme matricielle :

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = P(\pi), \tag{2.23}$$

où P représente la porte logique de déphasage qui est expliqué d'avantage dans la partie 2.4.6.3. Voici la représentation mathématiquement de l'état du qubit, après une application de la porte logique Z :

$$Z|0\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle \tag{2.24}$$

$$Z|1\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \end{bmatrix} = -|1\rangle \tag{2.25}$$

La représentation de la porte Z sur la sphère de Bloch :

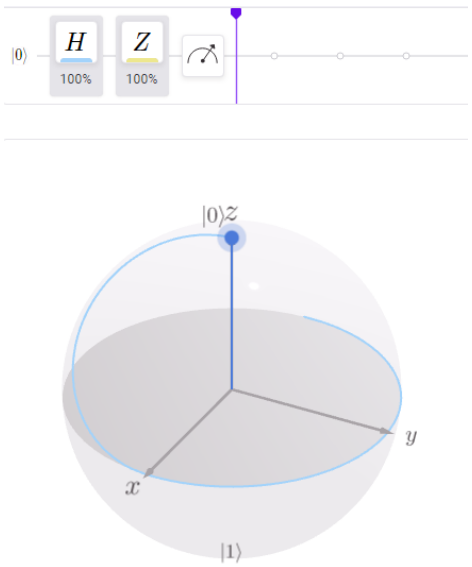


FIGURE 2.19 – Simulation de la porte Z sur la sphère de Bloch

Lorsqu'une matrice de Pauli est élevée à la puissance deux cette dernière devient égales à la matrice d'identité. Voici la représentation mathématique de ce concept :

$$X^2 = Y^2 = Z^2 = I. \tag{2.26}$$

2.4.6.2 La porte logique quantique H et le transformé de Walsh

La porte logique quantique *Hadamard* H (Fig. 2.20) est une matrice carrée complexe auto-adjointe $H = H^{\dagger}$ [17]. Cette porte a le rôle d'amener un vecteur à l'état superposé, elle permet de transformer les *qubits* d'un système quantique vers la superposition d'états. Il est possible de la représenter sous la forme matricielle :

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \frac{X+Z}{\sqrt{2}} = H^{\dagger}. \tag{2.27}$$

Cela invoque que la porte d'*Hadamard* est hermitienne et unitaire comme suit :

$$HH^{\dagger} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I. \tag{2.28}$$

Toutefois, cette dernière est un outil très puissant, utilisé pour implémenter la plupart des algorithmes quantiques.

$$H|1\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle \tag{2.29}$$

$$H|0\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle. \tag{2.30}$$

L'application de la porte d'*Hadamard* aux vecteurs de base standard d'un espace de Hilbert unidimensionnel résulte à la notation exprimée par l'ensemble $|+\rangle, |-\rangle$ est appelé la base polaire :

$$H|+\rangle = |0\rangle; \quad H|-\rangle = |1\rangle. \tag{2.31}$$

Voici la démonstration mathématique qui explique ces principes :

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = |+\rangle; \quad H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = |-\rangle \tag{2.32}$$

$$H|+\rangle = H\left(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right) = \frac{1}{2}\left((|0\rangle + |1\rangle) + (|0\rangle - |1\rangle)\right) = |0\rangle \tag{2.33}$$

$$H|-\rangle = H\left(\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right) = \frac{1}{2}\left((|0\rangle + |1\rangle) - (|0\rangle - |1\rangle)\right) = |1\rangle. \tag{2.34}$$

À partir de la représentation mathématique, il peut être constaté que le résultat après l'application de la porte H aux deux états du *qubits* produit une combinaison linéaire de ces derniers. De ce fait, la probabilité que le *qubit* soit $|-\rangle$ ou $|+\rangle$ dépend systématiquement du résultat combinatoire de $|1\rangle$ et $|0\rangle$. À travers cette démonstration mathématique, on peut prendre conscience de la puissance de la superposition d'états d'un système quantique.

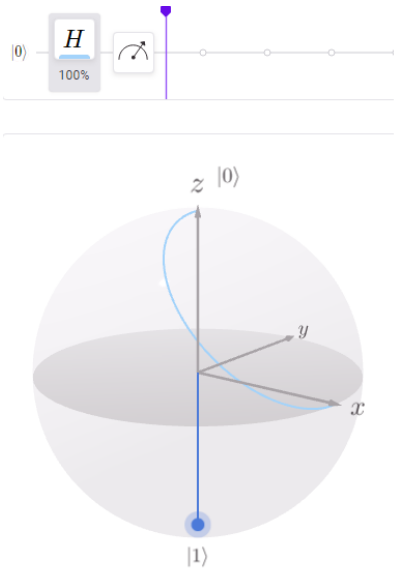


FIGURE 2.20 – Simulation de la porte *Hadamard* sur la sphère de Bloch

La matrice de Walsh est une matrice carrée de dimension 2^n, où n est un nombre naturel [10]. Les matrices d'*Hadamard* de dimension 2^x pour $x \in N$sont données par la formule récursive :

$$H(2^x) = \begin{bmatrix} H(2^{x-1}) & H(2^{x-1}) \\ H(2^{x-1}) & -H(2^{x-1}) \end{bmatrix} \tag{2.35}$$

$$H(2^0) = \begin{bmatrix} 1 \end{bmatrix}; \quad H(2^1) = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}; \quad H(2^2) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}. \tag{2.36}$$

Seuls $(1, -1)$ apparaissent dans les matrices de Walsh utilisées pour calculer la transformée de Walsh ou d'Hadamard qui ont la forme matricielle exprimés comme de suite :

$$H_m = \frac{1}{\sqrt{2}} \begin{bmatrix} H_{m-1} & H_{m-1} \\ H_{m-1} & -H_{m-1} \end{bmatrix}. \tag{2.37}$$

$$H_0 = \begin{bmatrix} 1 \end{bmatrix}; \quad H_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}; \quad H_2 = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}. \tag{2.38}$$

2.4.6.3 Les portes logiques quantiques S, $S^\dagger$, T, $T^\dagger$ et la porte de déphasage P

La probabilité relative à l'état $|1\rangle$ ou $|0\rangle$ est inaltérable après l'application de la porte logique de déphasage P qui a le rôle de modifier la phase de l'état du système quantique en représentant les états de base $|1\rangle \rightarrow e^{i\varphi}|1\rangle$ et $|0\rangle \rightarrow |0\rangle$. Cette dernière appartient aux types de portes quantiques à un seul qubit. Cette porte effectue une rotation le long de l'axe z sur la sphère de Bloch de φ radians. La porte de déphasage est représentée par la matrice :

$$P(\varphi) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\varphi} \end{bmatrix}. \tag{2.39}$$

La porte *Phase* est souvent appelée porte Z90 ou porte S (Fig. 2.21). Cette dernière est évaluée comme la moitié de la porte σ_z , au lieu d'aller à mi-chemin autour de l'axe de z de la sphère de Bloch, la porte logique S n'en fait que $\frac{1}{4}$: $S = \sqrt{Z}$.

$$S = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{2}} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} = P(\frac{\pi}{2}) = \sqrt{Z}. \tag{2.40}$$

La représentation de la rotation du *qubit* après l'affectation de la porte logique S.

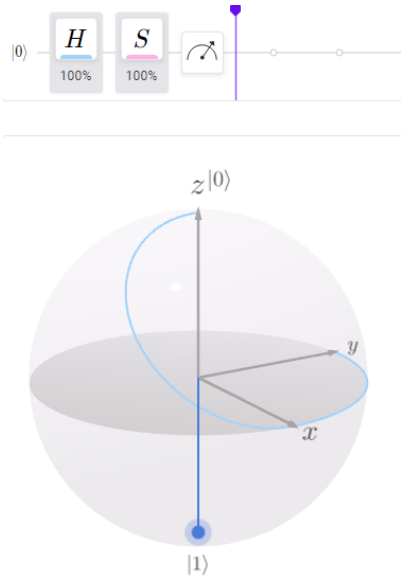


FIGURE 2.21 – Simulation de la porte *Phase* sur la sphère de Bloch

La porte logique S peut egalement faire le sens contraire de la même structure de rotation, appelé $S^\dagger$ «*S dagger*».

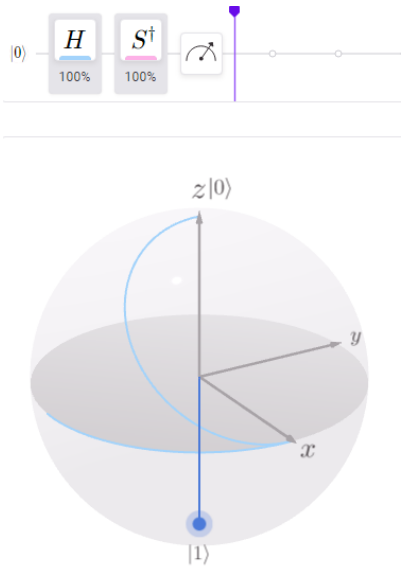


FIGURE 2.22 – Simulation de la porte *Phase-dagger* sur la sphère de Bloch

La porte logique T (Fig. 2.23) est évaluée comme la moitié de la porte S. Cette dernière fait un trajet de $\frac{1}{8}$ autour de l'axe de z de la sphère de Bloch qui peut être traduit sous la forme mathématique : $T = \sqrt[4]{Z}$.

$$T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{bmatrix} = P(\frac{\pi}{4}) = \sqrt{S} = \sqrt[4]{Z}. \tag{2.41}$$

La représentation de la rotation du *qubit* après l'affectation de la porte logique T.

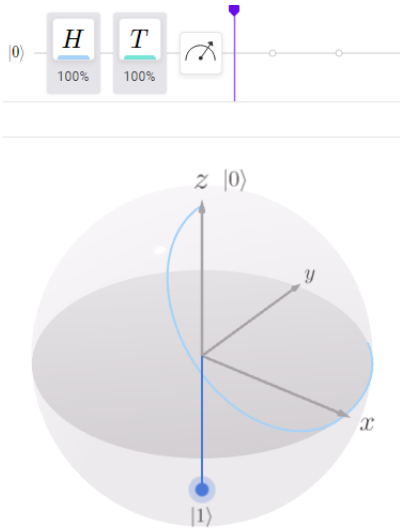


FIGURE 2.23 – Simulation de la porte T sur la sphère de Bloch

Similairement à la porte S, la porte logique T peut également faire le sens contraire de la même structure de rotation, appelée $T^{\dagger}$ «*T dagger*».

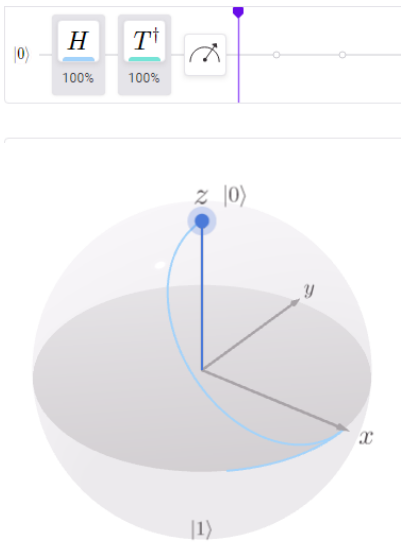


FIGURE 2.24 – Simulation de la porte *T-dagger* sur la sphère de Bloch

2.4.6.4 L'identité des portes quantiques et la base Hadamard

Sur le plan mathématique, la multiplication d'un ensemble de matrices simples donne une matrice complexe. En informatique quantique, le but est l'inverse : décomposer une porte ou un algorithme complexe en portes logiques quantiques plus simples. Ces portes peuvent être réalisées dans la base Hadamard. En utilisant la porte H , qui est très utile pour effectuer des manipulations de circuits flexibles et schématiques. Voici quelques exemples avec leurs preuves mathématiques [40] :

$$X = HZH \tag{2.42}$$

$$\begin{aligned} HZH &= \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \\ &= X. \end{aligned} \tag{2.43}$$

$$Z = HXH \tag{2.44}$$

$$\begin{aligned} HXH &= \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \\ &= Z. \end{aligned} \tag{2.45}$$

2.4.7 Les portes logiques multi-qubits *CNOT* et *SWAP*

Les portes *multi-qubits* sont plus complexes que les portes qui ne comportent qu'un *qubit* et ne peuvent être illustrées à l'aide de la sphère de Bloch. De ce fait, la représentation de ces concepts est seulement décrite par leurs matrices de comportement. Il est important de noter que toutes les portes doivent avoir le même nombre d'entrées et de sorties, car aucune information ne peut être perdue dans les systèmes quantiques. La porte logique *CNOT* (Fig. 2.25) appelée aussi porte CX ou porte X contrôlée est essentielle pour effectuer les traitements de l'information quantique, cependant elle peut être utilisée pour effectuer une intrication ou une désintrication du système quantique. *CNOT* effectue la même opération que la porte Pauli X, mais contrôlée par un autre *qubit*. Cette dernière agit sur le *qubit* cible du système et change son état, si le *qubit* de contrôle est dans l'état $|1\rangle$. Si le *qubit* de contrôle est dans l'état $|0\rangle$, rien n'arrive au *qubit* cible. Voici la matrice de représentation de la porte *CNOT* [22].

$$CNOT = \begin{bmatrix} 1\,0\,0\,0 \\ 0\,1\,0\,0 \\ 0\,0\,0\,1 \\ 0\,0\,1\,0 \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & X \end{bmatrix}. \tag{2.46}$$

Le circuit de la porte *Controlled-NOT* peut être représenté par le schéma suivant :

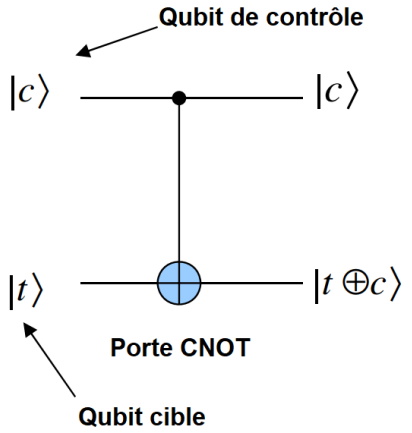


FIGURE 2.25 – Représentation du circuit de la porte logique quantique CNOT

Le signal supérieur contrôle le signal inférieur. L'entrée $|t\rangle$ ne serait pas changée si le signal supérieur $|c\rangle$ est $|0\rangle$, en revanche ; si le signal supérieur est $|1\rangle$, donc la porte inverse le signal du bas. Il est possible de représenter la porte *CNOT* dans la base d'*Hadamard* (Fig. 2.26). Pour réaliser cela d'abord, le produit tensoriel de deux transformées d'*Hadamard* opérant séparément sur deux *qubits* est représenté par H_2. Voici la représentation mathématique :

$$H_2 CNOT H_2 = C\bar{N}OT \tag{2.47}$$

alors :

$$C\bar{N}OT = \begin{bmatrix} 1\,0\,0\,0 \\ 0\,0\,0\,1 \\ 0\,0\,1\,0 \\ 0\,1\,0\,0 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}\begin{bmatrix} 1\,0\,0\,0 \\ 0\,1\,0\,0 \\ 0\,0\,0\,1 \\ 0\,0\,1\,0 \end{bmatrix}\frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \tag{2.48}$$

$$CNOT|00\rangle = |00\rangle\,; \quad CNOT|01\rangle = |01\rangle\,; \quad CNOT|10\rangle = |11\rangle\,; \quad CNOT|11\rangle = |10\rangle \tag{2.49}$$

$$C\bar{N}OT|00\rangle = |00\rangle\,; \quad C\bar{N}OT|01\rangle = |11\rangle\,; \quad C\bar{N}OT|10\rangle = |10\rangle\,; \quad C\bar{N}OT|11\rangle = |01\rangle. \tag{2.50}$$

Cette représentation matricielle échange les termes $|01\rangle$ et $|11\rangle$, tout en laissant les termes $|00\rangle$ et $|10\rangle$, ce qui signifie que le deuxième *qubit* devient le *qubit* de contrôle et le premier prend le rôle de la cible.

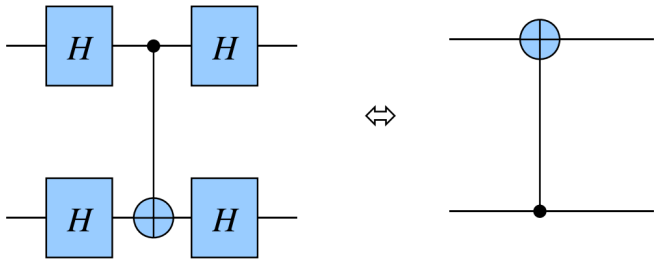


FIGURE 2.26 – Représentation du circuit de la porte logique quantique CNOT dans la base Hadamard

Le but de la porte $SWAP$ (Fig. 2.27) est d'échanger les positions de deux *qubits* qui peut être implémentée en utilisant trois portes $CNOT$

$$SWAP|\psi_1\rangle|\psi_2\rangle = |\psi_2\rangle|\psi_1\rangle$$

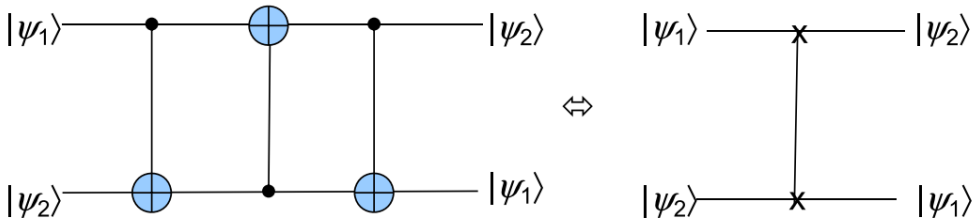


FIGURE 2.27 – Représentation du circuit de la porte logique quantique $SWAP$

Voici la matrice représentative de la porte logique $SWAP$:

$$SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \frac{I \otimes I + X \otimes X + Y \otimes Y + Z \otimes Z}{2}. \tag{2.51}$$

2.4.8 Les fonctions logiques universelles à multi-entrée, multi-sortie et le calcule réversible

En informatique, une porte logique est dite universelle lorsqu'elle peut être utilisée pour exprimer toutes les fonctions booléennes sur un nombre quelconque de *bits*, sans avoir besoin d'utiliser d'autres types de portes logiques [42]. La mécanique classique et la mécanique quantique dans la formulation hamiltonienne ne décrivent que des processus réversibles. Les portes logiques classiques AND et OR ne sont pas universels, car ils ne permettent pas d'exprimer l'ensemble des fonctions booléennes. Néanmoins, ils sont capables de représenter certaines fonctions booléennes non monotones, comme la fonction XOR (exclusive OR). Les portes logiques AND et OR sont irréversibles, c'est-à-dire qu'il est impossible de

reconstituer les entrées à partir de la sortie. Cependant, il existe des portes logiques classiques qui sont réversibles, tels que la porte *NOT* (ou porte *NAND* avec une seule entrée). La matrice de la porte *NOT* est la suivante :

$$NOT = \begin{bmatrix} 0 & 1 \end{bmatrix}. \tag{2.52}$$

Les portes logiques quantiques peuvent être inversés afin de retrouver l'état d'origine de la sortie, ce qui n'est pas le cas des portes logiques classiques. Cela permet de réaliser des calculs quantiques de manière réversible. La porte *Toffoli* (*CCNOT*) [30] est universelle pour les calculs classiques (Fig. 2.28), c'est-à-dire qu'elle peut être utilisée pour exprimer toutes les fonctions booléennes sur trois *bits*. Elle peut également être utilisée comme porte universelle pour les calculs quantiques en combinaison avec d'autres portes logiques quantiques, comme la porte Hadamard à un seul *qubit*. La matrice de la porte *Toffoli* est la suivante :

$$CCNOT = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}. \tag{2.53}$$

Voici la représentation du circuit *CCNOT* :

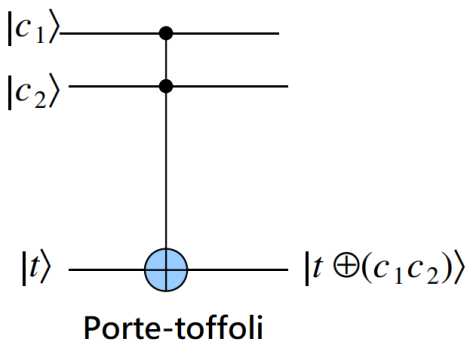


FIGURE 2.28 – Représentation du circuit de la porte logique quantique *CCNOT*

La porte *Toffoli* a trois entrées (x, y, z) et trois sorties (x', y', z'). La valeur de la sortie z' est déterminée par les valeurs des entrées x et y, comme indiqué dans la matrice ci-dessus. Par exemple ; si x et y sont tous deux égaux à 1, alors z' sera inversé (passera de 0 à 1 ou de 1 à 0). La porte *CNOT* est réversible, ce qui signifie qu'on peut reconstituer les entrées à partir de la sortie. Par exemple ; si y' est égal à 1, alors x peut-être soit 0, soit 1. Il existe également des portes logiques multi-entrée et multi-sortie, qui peuvent être utilisés pour représenter des fonctions logiques plus complexes. Par exemple ; une porte logique à 3-entrée *AND* peut être utilisé pour exprimer une fonction *AND* à trois entrées :

$$ANDD_3 = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \end{bmatrix}. \tag{2.54}$$

Les portes logiques multi-entrée et multi-sortie sont particulièrement utiles pour la conception de circuits logiques complexes. En effet, ils permettent de représenter des fonctions booléennes sur un nombre plus important de *bits*, ce qui est nécessaire pour réaliser des calculs de manière plus efficace. Il est important de noter que la plupart des portes logiques quantiques sont réversibles, contrairement aux portes logiques classiques. Cela signifie qu'on peut reconstituer les entrées à partir de la sortie de manière réciproque, ce qui est crucial pour la réalisation de calculs quantiques.

2.4.9 L'interférence, la décohérence et l'imprécision dans la réalisation des portes quantiques

La décohérence étudie les interactions entre un système quantique et son environnement macroscopique, généralement traité de manière classique. En effet, aucun système n'est complètement isolable de son environnement. La décohérence des effets quantiques macroscopiques est expérimentalement corrélée à la perte d'isolement. Une caractéristique principale des systèmes quantiques est l'apparition d'effets d'interférence ondulatoires. Ces interférences ne se manifestent que dans un grand nombre d'expériences identiques répétées sur des particules uniques. L'interférence ne peut pas être directement observée dans une seule expérience, car elle nécessite la superposition d'états quantiques multiples.

Un système est dit cohérent lorsqu'il présente une interférence. La décohérence correspond à la perte ou à la suppression de cette interférence due aux interactions avec l'environnement. Les expériences d'interférence exigent une isolation extrême du système d'intérêt, à l'exception du mécanisme de mesure ou d'observation. En effet, l'environnement peut facilement perturber les interférences quantiques. Les interférences quantiques jouent un rôle crucial dans le calcul quantique. Elles permettent de réduire l'amplitude des informations indésirables, éliminant ainsi le hasard quantique et améliorant la précision des calculs [11]. Cependant, les interférences sont difficiles à contrôler car elles peuvent interagir avec le système quantique lors de l'observation ou de la mesure. Cette interaction peut provoquer l'effondrement de la fonction d'onde $\hat{Q}$ (Fig. 2.29).

L'opérateur $\hat{Q}$ [12] représente une propriété quantique mesurable du système, telle que la position, la quantité de mouvement ou le spin. Cet opérateur agit sur l'état quantique du système, la fonction d'onde (ψ) pour en extraire une valeur spécifique de la propriété mesurée. L'effondrement de la fonction d'onde lors de la mesure correspond à la sélection d'un état propre particulier de cet opérateur, c'est-à-dire un état dans lequel la propriété mesurée par $\hat{Q}$ a une valeur bien définie. Lorsqu'un système quantique est dans un état initial représenté par la fonction d'onde ψ, on peut le décomposer mathématiquement sur la base des états propres de l'opérateur $\hat{Q}$ que l'on va noter ϕ_i : $|\psi\rangle = \sum_i c_i|\phi_i\rangle$.

Dans cette somme, chaque ϕ_i représente un état propre possible du système, et c_x est le coefficient associé à l'état propre ϕ_x. Ce coefficient c_x quantifie l'amplitude de la contribution de l'état propre ϕ_x à l'état global du système décrit par la fonction d'onde ψ. Lors de la mesure, l'effondrement de la fonction d'onde fait que le système bascule dans un état propre particulier de l'opérateur $\hat{Q}$. La probabilité P_x que le système s'effondre dans l'état propre ϕ_x est donnée par la probabilité de Born, calculée comme le module au carré du coefficient associé c_x :$P_x = |c_x|^2$.

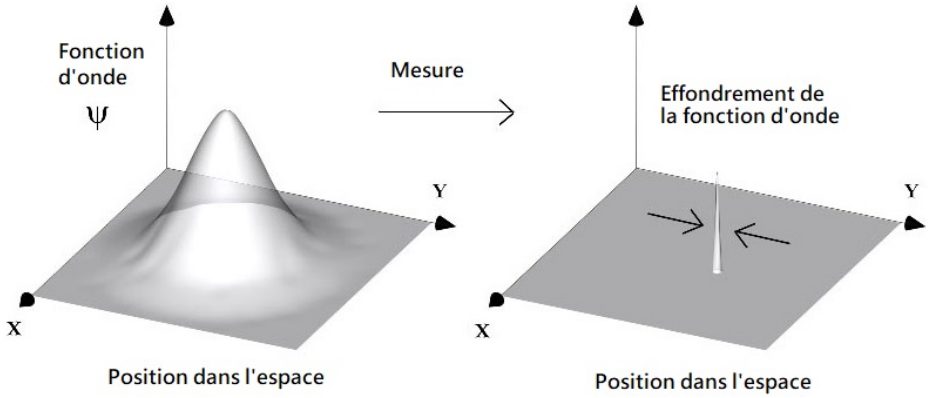


FIGURE 2.29 – Simulation de l'effondrement de la fonction d'onde

L'état d'un système quantique est modélisé par des opérateurs de densité sur un espace de Hilbert de dimension finie. Les opérateurs de densité peuvent être considérés comme des matrices semi-définies positives hermitiennes avec une trace égale à un. L'évolution d'un système quantique est décrite par des canaux quantiques, notés Φ, qui sont un type spécial de communication utilisé pour transférer des informations quantiques entre deux ou plusieurs *qubits*. Ces canaux fonctionnent linéairement sur des opérateurs préservant la trace complètement positifs (CPTP) $\Phi : L(H_A) \to L(H_B)$. L'imprécision dans la réalisation des portes quantiques et la décohérence des systèmes quantiques sont des concepts liés. Supposons que nous voulions appliquer la porte X (une rotation autour de l'axe X) à un *qubit* en utilisant un opérateur de rotation $\hat{R}_X(\theta)$. Nous pourrions le faire en appliquant un champ magnétique dans la direction X avec une force spécifique pendant une durée spécifique. Cependant, ni l'intensité du champ ni la durée ne seront parfaitement précises. Cela implique que l'angle de rotation réel θ' sera une variable aléatoire qui n'est pas égale à la valeur prévue θ [14]. Supposons que $\delta\theta$ est une variable aléatoire gaussienne : $\theta' = \theta + \delta\theta$, alors

$$\hat{R}_X(\theta + \delta\theta) = \hat{R}_X(\theta)\hat{R}_X(\delta\theta). \tag{2.55}$$

Dans ce cas, la valeur de $\delta\theta$ est incertaine, donc la meilleure approche pour décrire l'état du qubit bruité après la rotation est d'utiliser un opérateur de densité, qui est un ensemble infini :

$$\int d(\delta\theta)p(\delta\theta)\hat{R}_X(\theta)\hat{R}_X(\delta\theta)|\psi\rangle\langle\psi|\hat{R}_X^\dagger(\theta)\hat{R}_X^\dagger(\delta\theta). \tag{2.56}$$

L'intégrale $\int d(\delta\theta)p(\delta\theta)$ est sur toutes les valeurs possibles de $(\delta\theta)$ pondérées par la distribution de probabilité $p(\delta\theta)$. Cela représente le fait que l'erreur dans le fonctionnement de la porte peut prendre n'importe quelle valeur avec une certaine probabilité.

Les termes $\hat{R}_X(\theta)\hat{R}_X(\delta\theta)$ représentent l'action de la porte bruitée. C'est le produit de l'opération de porte idéale, $\hat{R}_X(\theta)$, et du terme d'erreur ou de bruit, $\hat{R}_X(\delta\theta)$. Cette opération de porte bruitée est appliquée à l'état quantique.

Les termes $|\psi\rangle\langle\psi|$ représentent l'état initial du système quantique, qui est un état pur. Il s'agit de l'état avant l'application de l'opération de porte.

Finalement les termes $\hat{R}_X^\dagger(\theta)\hat{R}_X^\dagger(\delta\theta)$ représentent le conjugué hermitien de l'opération de porte bruitée. Ils sont appliqués à l'état quantique après l'opération de porte bruitée.

En combinant tous ces éléments, la formule représente l'état du système quantique après une opération de porte bruyante. L'intégrale sur toutes les valeurs possibles de $\delta\theta$ tient compte du fait que l'erreur peut prendre n'importe quelle valeur avec une certaine probabilité. L'intégrale utilise également le fait que l'opérateur de densité, qui décrit l'état d'un système quantique, peut être écrit comme une intégrale

sur tous les états purs possibles pondérés par leurs probabilités. Cependant, cette intégrale peut être approchée par une opération quantique finie

$$|\psi\rangle\langle\psi| \to (1-\epsilon)\hat{R}_X(\theta)|\psi\rangle\langle\psi|\hat{R}_X^\dagger(\theta) + \epsilon X\hat{R}_X(\theta)|\psi\rangle\langle\psi|\hat{R}_X^\dagger(\theta)X. \tag{2.57}$$

Cette équation décrit une technique pour atténuer l'effet du bruit quantique sur un *qubit* bruyant. Le premier terme, $(1-\epsilon)\hat{R}_X(\theta)|\psi\rangle\langle\psi|\hat{R}_X^\dagger(\theta)$, représente l'état idéal du *qubit* avec le bruit supprimé. Ici, $\hat{R}_X(\theta)$ est un opérateur de rotation qui fait tourner le *qubit* autour de l'axe X d'un angle θ, et $\hat{R}_X^\dagger(\theta)$ est le conjugué hermitien de cet opérateur, qui fait pivoter le *qubit* du même angle.

Le second terme de l'équation, $\epsilon X\hat{R}_X(\theta)|\psi\rangle\langle\psi|\hat{R}_X^\dagger(\theta)X$, représente la contribution du bruit à l'état du *qubit*. Le facteur ϵ représente la force du bruit. L'opérateur X représente une erreur de retournement de *bit*, qui peut faire basculer le qubit de l'état $|0\rangle$ à l'état $|1\rangle$ ou vice versa. Ce terme représente la possibilité que le qubit ait subi une erreur de retournement de bit lors de l'opération de rotation.

En combinant ces deux termes, l'équation décrit un état modifié qui prend en compte le bruit dans le système. Le but de cette technique est de retrouver l'état idéal du qubit en appliquant une correction proportionnelle à la force du bruit. En pratique, la valeur de ϵ est généralement déterminée empiriquement en mesurant le bruit dans le système et en optimisant la correction en conséquence. Cela surmonte le bruit quantique au niveau du *qubit* bruyant, ainsi améliore la précision de la valeur de rotation θ.

2.4.10 La correction du bruit quantique et le théorème du non-clonage

La correction du bruit quantique QEC est un processus qui identifie et corrige les erreurs de calculs des ordinateurs quantiques. Les informations quantiques sont schématisées par des *qubits* logiques qui sont eux-mêmes composés de plusieurs *qubits* physiques, puis représentés par des *qubits* ancilla (Fig. 2.30). Ce mécanisme protège l'intégrité des informations quantiques. Cependant, cela a un coût en termes de nombre de *qubits* requis. Dans l'ensemble, plus le calcul est bruyant, plus il est nécessaire d'utiliser des *qubits* physiques afin de composer les *qubits* logiques nécessaires pour remédier à l'ensemble du bruit quantique du système. Selon le type d'algorithme exécuté, le rapport entre le nombre de *qubits* physiques et les *qubits* logiques varie.

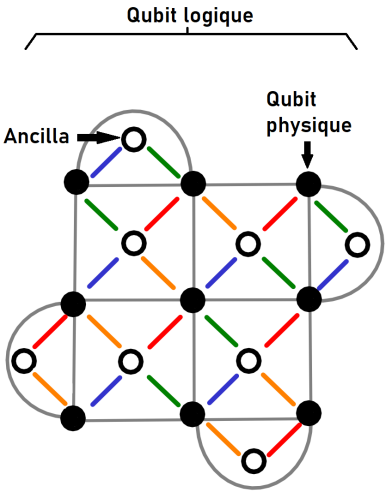


FIGURE 2.30 – Représentation d'un *qubit* logique

Pour comprendre comment il est possible de gérer le bruit dans un système classique, lors de la transmission d'un seul *bit* d'une source (X) à une destination (Y) sur un canal de communication, il existe une probabilité d'erreur, notée p. La probabilité que le *bit* demeure inchangé pendant la transmission est (1 -

p) (Fig. 2.31). Dans le canal symétrique binaire, la probabilité d'erreur est la même pour basculer un '0' vers un '1' ou un '1' vers un '0'.

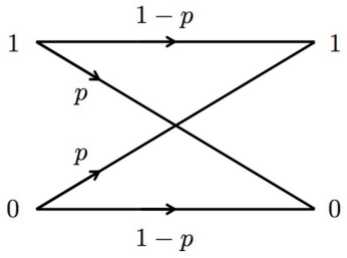


FIGURE 2.31 – Représentation d'un canal symétrique binaire

Pour corriger les erreurs de transmission, la redondance est employée. Par exemple, si nous voulons envoyer la séquence '1' suivi de '0', nous pouvons utiliser un schéma de codage où '1' est représenté par '111' et '0' est représenté par '000'. Ces *bits* codés sont ensuite transmis via le canal.

Les probabilités de différents scénarios d'erreur peuvent être calculées comme suit :

— La probabilité d'absence d'erreurs (tous les *bits* restent inchangés) : $(1-p)^3$.

— La probabilité d'une seule erreur (un *bit* inversé, deux *bits* restent inchangés) :

  — Il existe trois positions possibles pour que l'erreur se produise (premier, deuxième ou troisième *bit*). La probabilité d'une seule erreur dans l'une de ces positions est $(1-p)\cdot p\cdot(1-p) = (1-p)^2\cdot p$. La probabilité totale d'une seule erreur est donc $3\cdot(1-p)^2\cdot p = 3(1-p)^2 p$.

— La probabilité de deux erreurs (deux *bits* inversés, un *bit* reste inchangé) :

  — Il existe trois combinaisons possibles pour deux erreurs : '01x', '10x', '0x1' (où 'x' représente le *bit* inchangé). La probabilité que deux erreurs se produisent dans l'une de ces combinaisons est $p\cdot(1-p)\cdot p = p^2\cdot(1-p)$. Puisqu'il y a trois combinaisons, la probabilité totale de deux erreurs est $3\times p^2\times(1-p) = 3(1-p)p^2$.

— La probabilité de trois erreurs (tous les *bits* inversés) :

  — La probabilité que les trois *bits* soient inversés est $p\cdot p\cdot p = p^3$.

Pour illustrer la détection d'erreur, considérons les exemples :

— Les séquences reçues '000', '010', '001' et '100' peuvent être interprétées comme '0'.
— Les séquences reçues '111', '110', '101' et '011' peuvent être interprétées comme '1'.
— Pour évaluer la probabilité globale d'échec, qui représente la probabilité d'erreurs dans la transmission de la séquence de *bits*, on peut utiliser l'expression algébrique [13] :

$$3(1-p)p^2 + p^3 = 3p^2 - 2p^3. \tag{2.58}$$

En comparant la probabilité d'échec à la probabilité d'erreur initiale p, nous pouvons évaluer l'efficacité de la correction d'erreur. Si la probabilité d'échec est inférieure à p, cela implique que la redondance introduite a réduit les risques d'erreurs lors de la transmission. Cette approche de codage des informations avec redondance est couramment utilisée pour la détection et la correction d'erreurs dans les systèmes de communication. Le théorème du non-clonage quantique démontre l'impossibilité de faire des copies identiques d'états quantiques inconnus [1].

Bien qu'il soit difficile de formuler une théorie de correction d'erreur quantique avec l'approche simple, il est possible de répandre les informations logiques d'un *qubit* sur un état hautement intriqué de plusieurs *qubits* physiques. La redondance du code répéteur consiste à faire des copies de chaque *bit* d'information, puis de vérifier périodiquement ces copies les unes par rapport aux autres. Si l'un des *bits* est différent des autres, le calculateur peut corriger l'erreur et poursuivre le calcul. Cependant, l'équivalent quantique de cette logique peut-être représenté pas trois *qubits* physiques qui encodent l'état d'un seul *qubit* : $|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$ à un *qubit* logique : $|\Psi\rangle = (\alpha|000\rangle + \beta|111\rangle)^{\otimes 3} + (\alpha|000\rangle - \beta|111\rangle)^{\otimes 3}$ [44]. Le code du répéteur quantique ne peut pas être exactement le même que la version classique.

La puissance essentielle du calcul quantique vient du fait que les *qubits* peuvent exister dans une superposition d'états. Étant donné que la mesure d'un état quantique détruit la superposition, donc il n'y a pas un moyen évident de vérifier si une erreur s'est produite ou pas. Le seul moyen de savoir si les trois *qubits* physiques intriqués sont dans le même état est de vérifier que si l'un des *qubits* est différent, de ce fait cela indique qu'une erreur s'est produite. Pour arriver à ce but, on représente la série des *qubits* par deux *qubits* appelés *ancilla*. Le premier *ancilla* Q_x compare les premières Q_1 et deuxièmes Q_2 *qubits* physiques, le deuxième *ancilla* Q_y compare les deuxièmes Q_2 et les troisièmes Q_3. En mesurant les états de ces *qubits* auxiliaires, on peut ainsi considérer si les trois *qubits* contenant des informations sont dans des états identiques sans ennuyer l'état quantique de l'un d'entre eux (Tab. 2.2).

$Q_x \rightarrow (Q_1 \quad et \quad Q_2)$	$Q_y \rightarrow (Q_2 \quad et \quad Q_3)$	*Conclusion*
$Q_1(1)Q_2(1) \rightarrow$ *correspondance*	$Q_2(1)Q_3(1) \rightarrow$ *correspondance*	Aucune erreur
$Q_1(0)Q_2(1) \rightarrow$ *discordance*	$Q_2(1)Q_3(1) \rightarrow$ *correspondance*	Erreur produite en Q_1
$Q_1(1)Q_2(0) \rightarrow$ *discordance*	$Q_2(0)Q_3(1) \rightarrow$ *discordance*	Erreur produite en Q_2
$Q_1(1)Q_2(1) \rightarrow$ *correspondance*	$Q_2(1)Q_3(0) \rightarrow$ *discordance*	Erreur produite en Q_3

TABLE 2.2 – Table logique des relations entre la série des *qubits* physiques et les *ancilla*.

La phase relative entre les *qubits* peut aussi subir des erreurs causées par le bruit. On peut considérer cette phase comme une onde quantique, elle nous indique l'emplacement des crêtes *crest* et des creux *trough* de cette dernière (Fig. 2.32). Lorsque deux ondes sont en phase, leurs ondulations sont synchronisées. Si elles entrent en collision, elles interfèrent d'une façon constructive, en se fusionnant en une seule onde, doublant de taille de cette dernière. Si les ondes sont déphasées, lorsqu'une onde est à son sommet, ainsi l'autre est à son point le plus bas, cela produit une annulation ondulatoire. Ce phénomène crée une situation dans laquelle la bonne réponse à un calcul quelconque interfère d'une manière constructive, qui est ensuite amplifiée, tandis que la mauvaise réponse est supprimée par une interférence destructive. Si une erreur provoque une inversion de phase, alors l'interférence destructive peut se transformer en interférence constructive. Cependant, l'ordinateur quantique amplifie la mauvaise réponse, ce qui cause l'inexactitude des résultats du calcul. Pour remédier à ce problème, chaque *qubit* logique est codé en trois *qubits*, puis les *qubits* auxiliaires vérifient si l'une des phases s'est inversée. La combinaison de ces deux méthodes a rendu la correction d'erreur quantique possible.

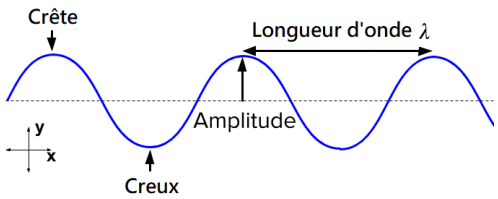


FIGURE 2.32 – Représentation d'une onde quantique

2.5 Conclusion

L'informatique quantique est un domaine en pleine expansion, doté d'un potentiel considérable pour des percées scientifiques dans de nombreux domaines, notamment la cryptographie et les télécommunications. Dans ce chapitre, nous avons présenté l'histoire de la révolution quantique, initiée par les travaux remarquables de scientifiques éminents tels que Max Planck, qui a introduit la constante de Planck 2.2, et Erwin Schrödinger, qui a formulé l'équation de Schrödinger 2.2. Nous avons également souligné les limites physiques de l'informatique classique, notamment la fin imminente de la loi de Moore 2.3.2 et la contrainte physique sur la taille des transistors. Nous avons ensuite exploré les concepts quantiques qui sont fondamentaux pour surmonter ces limites et propulser la science dans une nouvelle ère de technologie et d'innovation. Les phénomènes quantiques tels que la superposition et l'intrication jouent un rôle crucial dans l'accélération exponentielle des calculateurs quantiques par rapport aux calculateurs classiques. Les notions de *qubit* 2.4.2, de portes logiques quantiques 2.4.6, et de correction des erreurs quantiques 2.4.10 sont des mécanismes complexes, mais importants pour stabiliser et corriger les erreurs des calculs quantiques, permettant ainsi la réalisation des ordinateurs quantiques. Ces concepts et processus quantiques permettent la réalisation de certaines tâches telles que la factorisation en temps polynomial, un élément clé de ce mémoire développé en détail dans le chapitre III.

La cryptographie moderne, telle que le cryptosystème RSA, utilise la complexité de la factorisation comme un outil de défense de première ligne contre les ordinateurs classiques. Cependant, les ordinateurs classiques sont incapables d'exécuter des algorithmes quantiques en raison des limitations naturelles de la physique classique et de la représentation binaire traditionnelle qui est la base de l'information et la communication digitale pour les ordinateurs conventionnels.

Chapitre 3

Le cryptosystème RSA et l'algorithme quantique de *Peter Shor*

3.1 Les méthodes élémentaires et les objectifs fondamentaux de la cryptographie moderne

La cryptographie est l'ensemble des techniques utilisées pour assurer la confidentialité, l'intégrité et l'authenticité des informations et des communications. Le terme « crypto » dérive du mot grec *kryptós*, qui signifie « caché ». Elle consiste en la transformation des informations en question en une forme complexe, de telle sorte qu'elles soient très difficiles à déchiffrer par un tier, mais qui permet à l'expéditeur et au destinataire de les lire et de les traiter. Les algorithmes cryptographiques sont utilisés pour générer des signatures numériques, protéger les données lors de la navigation sur Internet, les communications confidentielles telles que les transactions bancaires et les informations gouvernementales. Lors de la transmission des données numériques, la méthode de chiffrement courante consiste à utiliser une clé secrète partagée entre les communications, connue sous le nom de système cryptographique symétrique. Les données sont cryptées à l'aide d'une seule clé secrète, puis le message chiffré ainsi que la clé secrète sont envoyés au destinataire pour dévoiler l'information en question. Cependant, un problème survient si un tiers malveillant intercepte le message. Dans ce cas, il peut décrypter et lire le message secret car il dispose de la clé secrète. Afin de résoudre ce problème, il est essentiel d'utiliser un système à clé publique, également connu sous le nom de système cryptographique asymétrique (Fig. 3.1). Dans ce système, chaque utilisateur dispose de deux clés, une publique et une privée. Les expéditeurs récupèrent la clé publique de leurs destinataires qui est ouvertement disponible sur l'ensemble du réseau, puis codent le message à l'aide de cette dernière avant de l'envoyer à la destination désirée. Lorsque le message atteint sa destination, seule la clé privée du récepteur peut décoder l'information, ce qui signifie qu'il est impossible de lire le message sans la clé privée correspondante.

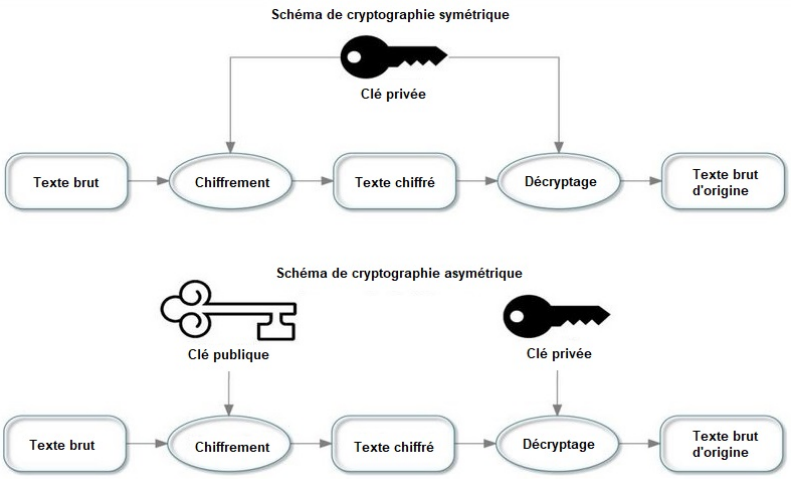


FIGURE 3.1 – Schéma explicatif du cryptosystème symétrique et asymétrique

Les objectifs fondamentaux de la cryptographie moderne peuvent être définis en quatre termes clés : la confidentialité, l'intégrité, la non-répudiation et l'authentification [33]. Le concept de confidentialité implique que l'information ne peut être interprétée que par le destinataire autorisé. L'intégrité des données signifie que les informations ne peuvent être altérées pendant le stockage ou le transfert entre l'expéditeur et le destinataire. Le principe de non-répudiation évoque que le créateur de l'information ne peut nier son implication dans la création ou la transmission de l'information. Enfin, l'authentification concerne la validation de l'identité de l'expéditeur et du destinataire, ainsi que de l'origine de l'information échangée. Les protocoles qui répondent à ces quatre critères sont appelés cryptosystèmes. Ces systèmes reposent sur des procédures mathématiques et informatiques complexes qui permettent la réalisation de la cryptographie à petite ou grande échelle. Un cryptosystème est construit sur trois mécanismes clés : la génération de clés, le chiffrement et le déchiffrement.

3.2 Analyse comparative des algorithmes cryptographiques symétriques, asymétriques et hybrides

Le cryptosystème *Rivest-Shamir-Adleman* (RSA) est largement utilisé dans de nombreuses infrastructures informatiques en tant qu'algorithme de chiffrement asymétrique. Toutefois, il est important de noter que cette affirmation est vraie à l'heure actuelle et pourrait changer dans le futur. Cette méthode utilise une paire de clés mathématiquement liées pour chiffrer et déchiffrer des informations. La fonctionnalité de ce cryptosystème est souvent employée pour générer et authentifier les signatures numériques qui peuvent attester de la légitimité et de l'intégrité des informations numériques. Cependant, les cryptosystèmes asymétriques présentent certaines limitations, notamment leur inefficacité à chiffrer des données de taille énorme. Ces méthodes de chiffrement requièrent plus de ressources de calcul que les algorithmes symétriques, ce qui les rend plus coûteuses en termes de temps de calcul. Afin d'optimiser les systèmes cryptographiques, il est souvent nécessaire d'associer l'algorithme RSA à des méthodes de chiffrement symétrique tels que *Advanced Encryption Standard* (AES) [5]. Cette fusion algorithmique permet de sécuriser les données de manière plus efficace, en utilisant un algorithme à clé symétrique pour chiffrer les informations, puis en encodant la clé symétrique avec le chiffrement RSA. Seule l'entité disposant de la clé privée RSA peut déchiffrer la clé symétrique, qui permettra alors de révéler les données dans leur état initial. Sans la clé symétrique, les informations demeurent pratiquement indéchiffrables. Ce processus de chiffrement hybride est donc couramment utilisé pour sécuriser les informations numériques de manière optimale, sans consommer trop de temps ni de ressources de calcul.

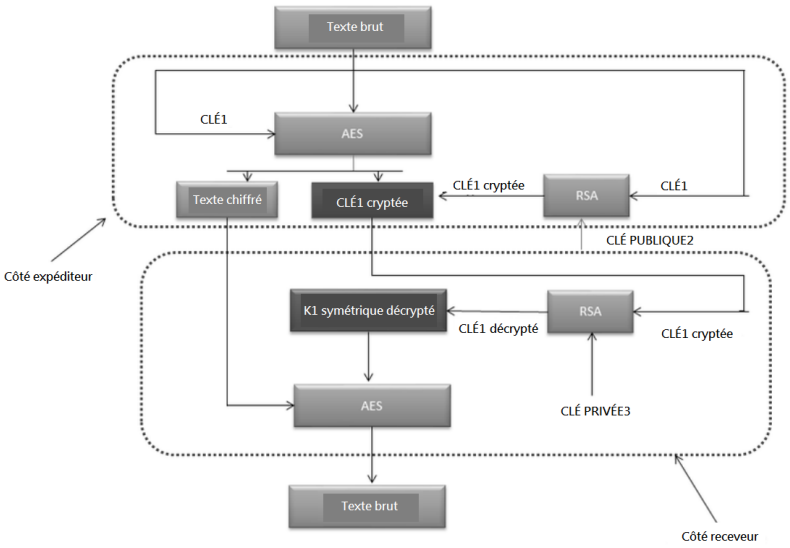


FIGURE 3.2 – L'architecture du système de chiffrement hybride RSA et AES

La puissance de la cryptographie moderne repose sur la synergie de deux types d'algorithmes, conférant ainsi à ce mécanisme une robustesse, une stabilité et une rapidité exemplaires. Dans cette section, nous présentons une expérience démontrant les performances des systèmes cryptographiques purs et hybrides, en nous appuyant sur un langage de programmation scientifique hautement performant, à savoir *Python*, (voir l'annexe B). L'expérience consiste à mesurer les différentes vitesses des algorithmes cryptographiques, en se basant sur trois critères de mesure, à savoir la génération de la paire de clés, le chiffrement et le déchiffrement des données dans notre cas, on utilise les tailles de fichier suivantes : [64, 128, 256, 512, 1024, 2048]. Nous débutons par l'évaluation du cryptosystème RSA, avant de passer aux systèmes AES et hybride (RSA + AES). Afin de garantir la reproductibilité des résultats, les algorithmes ont été testés sur les serveurs de *Google Colaboratory*.

3.2.1 Observations et résultats

À l'examen minutieux du graphique présenté en (Fig. 3.3), les variations entre les différentes approches cryptographiques se manifestent avec éloquence. Le cryptosystème RSA est l'un des algorithmes asymétriques les plus répandus, utilisé tant pour le chiffrement que pour la signature numérique. Il fonctionne sur le principe d'une paire de clés : une clé publique pour le chiffrement et une clé privée pour le déchiffrement. Ces clés sont générées à partir d'une procédure mathématique complexe. Pour notre expérimentation, nous avons opté pour une taille de clé standard de *2048 bits*, équivalente à une longueur de 617 chiffres décimaux. Cette taille est généralement reconnue comme offrant un bon équilibre entre sécurité et performance. Toutefois, d'après les résultats de notre étude, il est clair que le temps de chiffrement RSA augmente significativement avec la taille du fichier. Cette croissance rapide du temps de chiffrement peut devenir problématique lorsqu'on travaille avec des fichiers de grande taille, rendant RSA moins adapté à de telles situations.

Par rapport au RSA, l'algorithme AES [19] démontre une évolution du temps de chiffrement quasiment constante, suggérant une complexité algorithmique approximativement linéaire, soit O(n). Cette caractéristique souligne la capacité de l'AES à traiter efficacement et rapidement des blocs de données de tailles variées, faisant en sorte que son temps de chiffrement soit essentiellement proportionnel à la taille des données en entrée.

Le système hybride RSA+AES [19], quant à lui, bénéficie des avantages de chacun. Tout en offrant la sécurité robuste du RSA, il tire parti de la rapidité de l'AES, se situant donc entre les deux en termes de

performance.

Le choix d'un algorithme de cryptographie doit tenir compte de plusieurs facteurs, dont le type et la taille des données à protéger, ainsi que les contraintes de performance. Par exemple, si la sécurité est une priorité absolue et que les données à chiffrer sont relativement petites, le RSA peut être préférable. En revanche, pour le chiffrement en temps réel de gros volumes de données, l'AES est nettement plus avantageux.

Le cryptosystème hybride, combinant RSA et AES, offre un équilibre entre la sécurité renforcée du RSA et la rapidité de l'AES. Il pourrait donc être envisagé pour des applications nécessitant à la fois une haute sécurité et une performance acceptable.

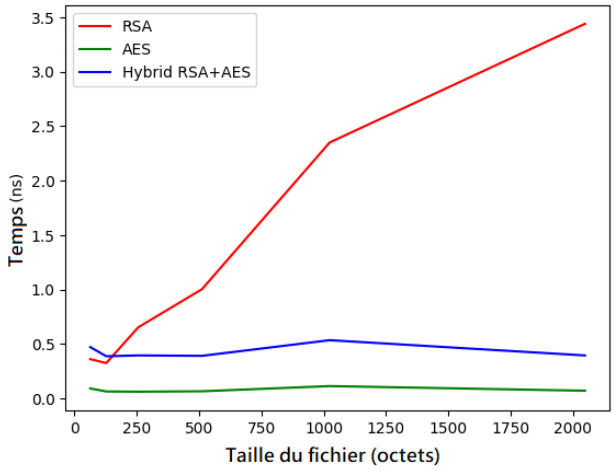


FIGURE 3.3 – Comparaison des Temps d'Encodage : RSA, AES et le Système Hybride en Fonction de la Taille du Fichier

3.3 La structure mathématique et le fonctionnement du schéma cryptographique asymétrique RSA

Le cryptosystème RSA joue un rôle essentiel dans de nombreux systèmes cryptographiques hybrides modernes. Bien que d'autres méthodes, comme l'ECC, « Elliptic Curve Cryptography » [20], soient également utilisées, RSA reste une référence en matière de chiffrement asymétrique. En effet, les systèmes asymétriques sont particulièrement prisés pour la distribution de clés et la signature numérique.

Cependant, comme tout système cryptographique, RSA a ses limites. Bien qu'il soit actuellement très fiable contre les méthodes d'attaque conventionnelles, l'émergence d'ordinateurs quantiques pose une menace potentielle à sa sécurité. Par conséquent, les informations confidentielles peuvent être exposées à un risque considérable (ce concept sera examiné de manière plus approfondie dans la section 3.5). Pour mettre cela en perspective, déchiffrer des informations codées avec le modèle cryptographique RSA-2048 sans la clé privée à l'aide des techniques classiques actuelles prendrait une durée astronomique, évaluée à « six virgule quatre quadrillions d'années ». Cependant, il est théoriquement envisagé qu'un ordinateur quantique idéalement performant avec « quatre mille quatre-vingt-dix-neuf qubits » pourrait compromettre le cryptosystème RSA-2048 en une fraction de ce temps [18]. Néanmoins, il convient de noter que la réalisation d'un tel ordinateur quantique reste, pour le moment, hors de notre portée. De plus, la communauté cryptographique est déjà à la recherche de solutions post-quantiques, préparant le terrain pour l'avenir.

Cela souligne le potentiel impressionnant de l'informatique quantique, mais également la nécessité de tenir

compte des avancées technologiques futures dans l'évaluation de la sécurité des systèmes cryptographiques. Cela témoigne de la puissance et des capacités impressionnantes de l'informatique quantique. Ainsi, une bonne compréhension du fonctionnement du cryptosystème RSA est nécessaire pour comprendre les faiblesses de ce dernier face aux algorithmes quantiques, tels que l'algorithme de Peter Shor [24]. Tout d'abord, la clé publique est créée en multipliant deux grands nombres premiers. Quant à la création de la clé privée, le processus est plus complexe et implique également ces derniers.

Les ordinateurs classiques sont incapables de factoriser le produit de deux nombres premiers extrêmement grands dans un temps raisonnable, car cela nécessite d'essayer toutes les combinaisons possibles, ce qui représente un nombre astronomique de cas. La multiplication de deux grands nombres est une opération fondamentalement moins complexe que la factorisation de leur produit. Les algorithmes de multiplication démontrent une complexité temporelle logarithmique $O(log\, n)$, ce qui signifie que le temps nécessaire pour effectuer le calcul augmente proportionnellement au logarithme de la taille des nombres. En revanche, l'algorithme de factorisation d'entiers à usage général le plus efficace, « le crible spécial de corps de nombres (SNFS) », présente une complexité temporelle sous-exponentielle $O\left(\exp\left(\left(\frac{64}{9}\right)^{1/3} \cdot (\log N)^{1/3} \cdot (\log\log N)^{2/3}\right)\right)$ [15]. Cependant, la complexité du SNFS est heuristique et basée sur des analyses et des observations, non sur une preuve mathématique stricte.

Cette forte disparité de complexité signifie que la factorisation d'un grand nombre est exponentiellement plus difficile que la multiplication de deux nombres comparables. Ce principe a des ramifications importantes pour la cryptographie. De nombreux systèmes de chiffrement modernes reposent sur la difficulté de prendre en compte de grands nombres entiers pour assurer la sécurité. La découverte d'un algorithme de factorisation rapide en temps polynomial constituerait une menace majeure pour l'intégrité de ces systèmes, notamment l'algorithme de Shor quantique.

L'implémentation de RSA fait un usage intensif de l'arithmétique modulaire et du théorème d'Euler. Pour réaliser l'algorithme RSA d'abord, il faut choisir deux grands nombres premiers p et q. Leur produit, N, représente la moitié de la clé publique. L'autre moitié est calculée par $\phi(N) = (p-1)(q-1)$, par la suite, on choisit un nombre e relativement premier à $\phi(N)$ et $1 < e < \phi(N)$. (Voir la représentation logique de ce concept avec le langage de programmation *Python* depuis l'annexe C). En pratique, e est souvent égal à $F_4 = 2^{16} + 1$. Les choix courants sont $F_0 = 3, F_1 = 5, F_2 = 17, F_3 = 257\, et\, F_4 = 65537$ qui sont considérés comme des nombres premiers de *Fermat* réalisables avec la forme algébrique $2^{2^n} + 1$. Cependant, e et N décrivent la clé publique. Pour arriver à générer la clé privée, on doit calculer l'inverse modulaire d de e modulo $\phi(N)$. Autrement dit, $de \equiv 1(mod\, \phi(N))$. Il est possible de calculer la clé privée avec la fonction *Modular_ multiplicative_ inverse* depuis l'annexe C. d dans ce cas représente la clé privée qui n'est pas divulguée publiquement.

Pour transmettre les informations numériques, tout d'abord, l'expéditeur doit convertir le message en question en un nombre m. Le processus de conversion est basé sur le code *ASCII*, (Tab. 3.1). Alors pour le chiffrer, on calcule $c = m^e(mod\, N)$, c représente le message chiffré. La fonction *Crypter* depuis l'annexe C décrit le concept. Pour déchiffrer ce dernier, on calcule $m = c^d(mod\, N)$ ainsi, on peut récupérer le nombre original m qui va être traduit par la suite en texte compréhensible par l'humain à l'aide du code *ASCII*. La réalisation en code de ce concept est démontré grâce à la fonction *Decrypter* depuis l'annexe C.

Supposons que nous voulions calculer manuellement l'algorithme RSA. On choisit les nombres premiers $p = 383$ et $q = 389$. D'abord, on calcule la moitié de la clé publique $N = pq = 148987$. Puis, on calcule $\phi(N) = (p-1)(q-1) = 148216$. Ainsi, on choisit $e = F_4 = 65537$, car ce nombre est relativement premier à $e = F_4 = 65537$ et $1<65537<\phi(N)$. Donc, on obtient la clé publique qui a la forme *(N,e)=(148987,65537)*. On calcule par la suite la clé privé d voici l'expression mathématique pour trouver la valeur de d :

$$\begin{aligned} ed, (\text{mod}, \phi(N)) &= 1 \\ 65537 \cdot d, (\text{mod}, 148216) &= 1 \\ 65537 \cdot 4401, (\text{mod}, 148216) &= 1 \end{aligned} \tag{3.1}$$

alors :

$$d = 4401.$$

Supposons maintenant que l'on veut chiffrer le message *UQTR*. En se basant sur le code *ASCII*, cependant [*U = 85, Q = 81, T = 84 et R = 82*], ainsi la totalité du message est *m = 85 81 84 82*. Ensuite, on évalue chaque lettre par la formule suivante : $c = m^e (mod\, N)$, par exemple $U_c = 85^{65537}\,(mod\, 148987) = 8165$, ainsi de suite. On obtient finalement le message chiffré : *[8165, 88697, 45669, 68485]*. Pour déchiffrer ce dernier, on utilise la formule suivante : $m = c^d (mod\, N)$ pour chaque caractère chiffré. Par exemple $8165^{4401}\,(mod\, 148987) = 85 = U$, Donc voilà le fonctionnement mathématique de l'algorithme RSA.

Decimal	**Binaire**	**Évaluation**
81	0101 0001	Q
82	0101 0010	R
84	0101 0100	T
85	0101 0101	U

TABLE 3.1 – Codes de caractères alphabétiques ASCII

3.4 L'Authentification et l'intégrité des données à l'aide du hachage et du cryptosystème RSA

Le cryptosystème RSA est efficace pour confirmer qu'un message a été envoyé par une entité quelconque qui affirme l'avoir acheminé, ainsi que pour prouver qu'un message n'a pas été altéré ou falsifié. Pour prouver l'authenticité du message, il est possible de calculer un hachage. Ce concept peu être décrit comme une fonction qui prend des informations d'une taille arbitraire et les transformer en une valeur de longueur fixe, puis signer l'information avec la clé privée génère (Fig. 3.4). Une fois le message signé, la signature numérique est ensuite envoyée au destinataire avec le message crypté. Pour vérifier la signature numérique, le destinataire utilise d'abord la même fonction de hachage pour trouver la valeur de hachage du message qu'il a reçu. Le destinataire applique ensuite la clé publique de l'expéditeur à la signature numérique, en utilisant la formule de chiffrement RSA discuté dans la partie 3.3, afin d'obtenir le hachage de la signature numérique. En comparant le hachage du message reçu avec le hachage de la signature numérique cryptée, le destinataire peut déterminer si le message est authentique. Si les deux valeurs sont identiques, le message n'a pas été modifié alors ce dernier a été signé par l'expéditeur d'origine. Si le message est modifié, la valeur de hachage serait complètement différente.

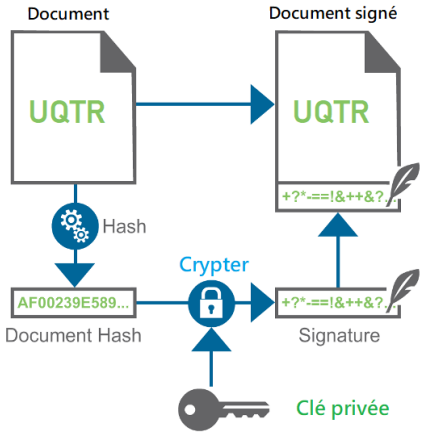


FIGURE 3.4 – Schéma descriptif du processus de création de la signature électronique

Les fonctions de hachage cryptographique sont utilisées afin de sécuriser les données en fournissant trois caractéristiques de sécurité fondamentales : la résistance à la pré-image, la résistance à la seconde pré-image et la résistance aux collisions (Fig. 3.5). Le concept indispensable et primaire du hachage cryptographique réside dans la résistance à la pré-image, ce qui rend le processus de trouver un message x depuis la valeur du hachage respective $h(x)$ une procédure ardue. Cette sécurité est garantie par la nature des fonctions unidirectionnelles. La résistance pré-image est nécessaire pour résister aux attaques *Brute-force*. La deuxième résistance de pré-image est définie comme étant donné une information x_1 , il est difficile de trouver une autre information x_2 telle que $x_1 \neq x_2$ et $h(x_1) = h(x_2)$. Les fonctions dépourvues de cette propriété sont vulnérables aux secondes attaques de pré-image. La dernière caractéristique est la résistance aux collisions, qui a le but de rendre extrêmement difficile de trouver deux messages complètement différents qui hachent la même valeur : $h(x_1) = h(x_2)$. Afin de réussir cette caractéristique, referont nous au principe du pigeonnier. Il doit y avoir un nombre similaire d'entrées possibles N_{in} à des sorties possibles N_{out} , car s'il existe plus d'entrées que de sorties $N_{in} > N_{out}$, cela implique définitivement des collisions potentielles. Sans résistance aux collisions, les signatures numériques peuvent être compromises.

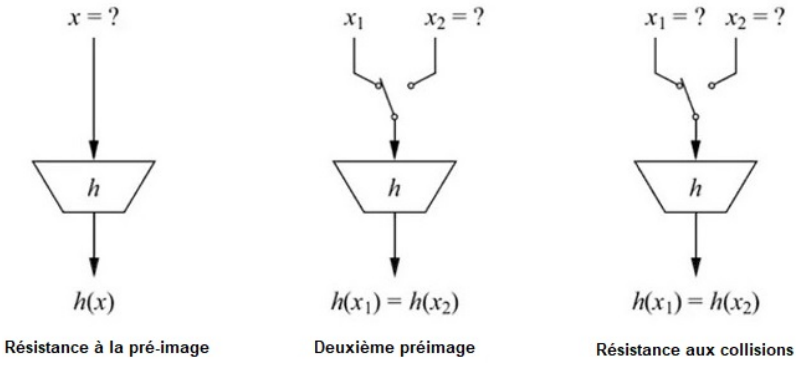


FIGURE 3.5 – Représentation logique des trois caractéristiques fondamentales du hachage cryptographique

3.4.1 La conception de la fonction de hachage cryptographique SHA-512

L'algorithme SHA-512 utilise une fonction mathématique de hachage unidirectionnelle, cette fonction prend la longueur variable d'un message x entrant et le transformer en une séquence binaire de longueur fixe [27]. Le message d'entrée peut avoir une longueur allant jusqu'à 2^{128-1} *bits* et le traitement de l'information se fait par blocs de 1024 *bits*. Pour un message x d'une taille quelconque, d'abord, il est nécessaire de convertir les caractères de ce dernier en *ASCII*, puis en binaire. Par la suite, le message est complété

avec des chiffres binaires commençant par (1), ensuite le reste est affecté par des (0) pour que sa longueur soit congruente à *896 modulo 1024* ou $L_x \, mod \, 1024 = 896$. L'ajout d'un bloc de 128 *bits* se fait au message. Ce denier contient la valeur de la longueur du message d'origine avant le rembourrage afin de compléter une taille de *(896 + 128) = 1024 bits*. Les huit variables des registres, notées a_0, b_0, c_0, d_0, e_0, f_0, g_0, h_0 (Tab. 3.2) sont initialisées à certaines constantes fixes, puis le message d'entrée est divisé en 16 blocs de 1024 *bits* étendu à $y = 80$ mots w_x avec $0 \leq x \leq y-1$ (Fig. 3.6). Voici la formule descriptive de cette procédure :

$$\sigma_{0,i} = (w_{i-15} \ggg 1) \oplus (w_{i-15} \ggg 8) \oplus (w_{i-15} \ggg 7) \tag{3.2}$$

$$\sigma_{1,i} = (w_{i-2} \ggg 19) \oplus (w_{i-2} \ggg 61) \oplus (w_{i-2} \ggg 6) \tag{3.3}$$

$$w_i = \sigma_{0,i} + \sigma_{1,i} + w_{i-16} + w_{i-7}. \tag{3.4}$$

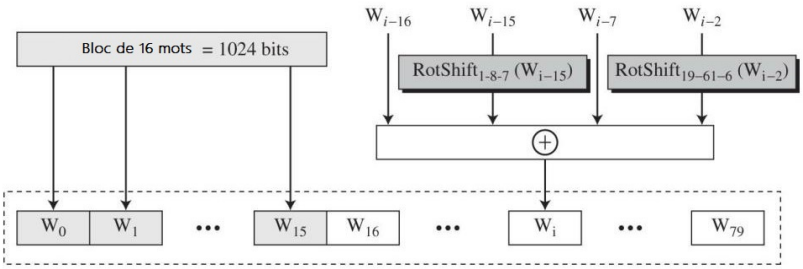


FIGURE 3.6 – Schéma représentatif de l'extension des mots de la fonction de hachage SHA-512

Ensuite, les huit variables des registres sont mises à jour à l'aide d'une fonction de compression composée de 80 tours. Le traitement d'un bloc de message de 1024 *bits* donne huit valeurs de hachage intermédiaires de 64 *bits*. Une fois que l'ensemble du message m est traité, le résumé de 512 *bits* est généré en concaténant les huit valeurs de hachage intermédiaires (Fig. 3.7).

Registres	**Valeurs *Hex***
a_0	6A09E667F3BCC908
b_0	BB67AE8584CAA73B
c_0	3C6EF372FE94F82B
d_0	A54FF53A5F1D36F1
e_0	510E527FADE682D1
f_0	9B05688C2B3E6C1F
g_0	1F83D9ABFB41BD6B
h_0	5BE0CD19137E2179

TABLE 3.2 – Les valeurs initiales des registres de la fonction de hachage SHA-512 en hexadécimales

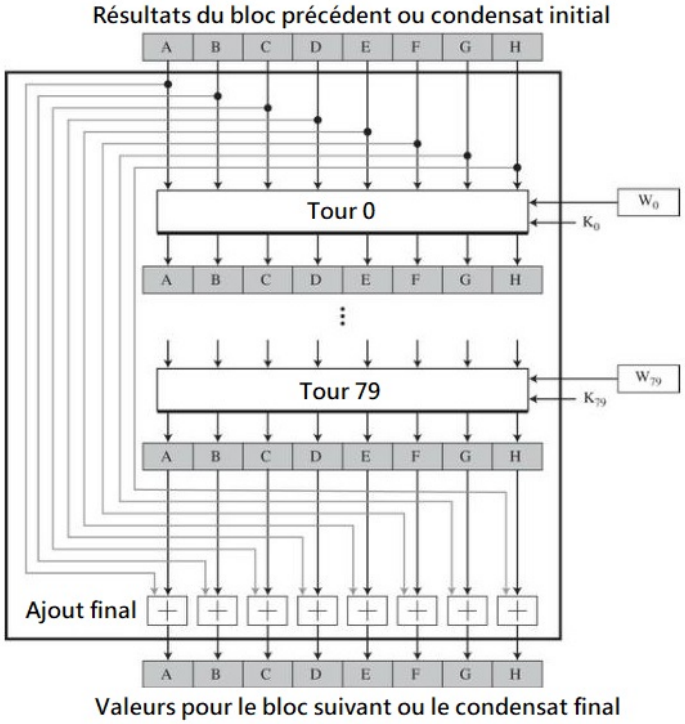


FIGURE 3.7 – Schéma représentatif de la fonction de compression du système de hachage SHA-512

Chaque tour de la fonction de compression se compose de quatre opérations : *Maj Majority*, *Ch Choice*, Σ_0 et Σ_1 (Fig. 3.8). Ces derniers sont composés d'une séquence de calculs principalement arrangés par les opérateurs logique $AND : \wedge$, $XOR : \oplus$ et l'opération de rotation à droite $ROTR : \ggg$.

$$\Sigma_{0,i} = (a_i \ggg 28) \oplus (a_i \ggg 34) \oplus (a_i \ggg 39) \tag{3.5}$$

$$\Sigma_{1,i} = (e_i \ggg 14) \oplus (e_i \ggg 18) \oplus (e_i \ggg 41) \tag{3.6}$$

$$Ch_i = (e_i \wedge f_i) \oplus (\bar{e}_i \wedge g_i) \tag{3.7}$$

$$Maj_i = (a_i \wedge b_i) \oplus (a_i \wedge c_i) \oplus (b_i \wedge c_i) \tag{3.8}$$

$$T_{1,i} = \Sigma_{1,i} + Chi_i + h_i + w_i + k_i \tag{3.9}$$

$$T_{2,i} = \Sigma_{0,i} + Maj_i. \tag{3.10}$$

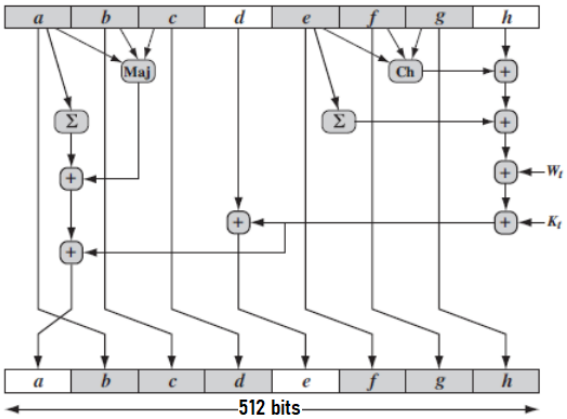


FIGURE 3.8 – Le mécanisme d'un seul tour de la fonction de hachage SHA-512

À chaque tour R de la fonction de compression, l'ensemble des huit variables du registre est mis en rotation par une variable k qui est l'une des quatre-vingts constantes de 64 *bits* (Fig. 3.9).

428A2F98D728AE22	7137449123EF65CD	B5C0FBCFEC4D3B2F	E9B5DBA58189DBBC
3956C25BF348B538	59F111F1B605D019	923F82A4AF194F9B	AB1C5ED5DA6D8118
D807AA98A3030242	12835B0145706FBE	243185BE4EE4B28C	550C7DC3D5FFB4E2
72BE5D74F27B896F	80DEB1FE3B1696B1	9BDC06A725C71235	C19BF174CF692694
E49B69C19EF14AD2	EFBE4786384F25E3	0FC19DC68B8CD5B5	240CA1CC77AC9C65
2DE92C6F592B0275	4A7484AA6EA6E483	5CB0A9DCBD41FBD4	76F988DA831153B5
983E5152EE66DFAB	A831C66D2DB43210	B00327C898FB213F	BF597FC7BEEF0EE4
C6E00BF33DA88FC2	D5A79147930AA725	06CA6351E003826F	142929670A0E6E70
27B70A8546D22FFC	2E1B21385C26C926	4D2C6DFC5AC42AED	53380D139D95B3DF
650A73548BAF63DE	766A0ABB3C77B2A8	81C2C92E47EDAEE6	92722C851482353B
A2BFE8A14CF10364	A81A664BBC423001	C24B8B70D0F89791	C76C51A30654BE30
D192E819D6EF5218	D69906245565A910	F40E35855771202A	106AA07032BBD1B8
19A4C116B8D2D0C8	1E376C085141AB53	2748774CDF8EEB99	34B0BCB5E19B48A8
391C0CB3C5C95A63	4ED8AA4AE3418ACB	5B9CCA4F7763E373	682E6FF3D6B2B8A3
748F82EE5DEFB2FC	78A5636F43172F60	84C87814A1F0AB72	8CC702081A6439EC
90BEFFFA23631E28	A4506CEBDE82BDE9	BEF9A3F7B2C67915	C67178F2E372532B
CA273ECEEA26619C	D186B8C721C0C207	EADA7DD6CDE0EB1E	F57D4F7FEE6ED178
06F067AA72176FBA	0A637DC5A2C898A6	113F9804BEF90DAE	1B710B35131C471B
28DB77F523047D84	32CAAB7B40C72493	3C9EBE0A15C9BEBC	431D67C49C100D4C
4CC5D4BECB3E42B6	4597F299CFC657E2	5FCB6FAB3AD6FAEC	6C44198C4A475817

FIGURE 3.9 – Liste des constantes k de tours de la fonction de hachage SHA-512

Voici un exemple pour le message $m = UQTR$. La conversion du message en question sous la forme *ASCII* puis en binaire est : $m = 85\,81\,84\,82 \rightarrow [01010101\,01010001\,01010100\,01010010]$. La représentions du concept en fonction *m_ bin* et dans l'annexe D. La taille totale du message est de $(8 \cdot 4) = 32\,bits$ donc il faut réaliser une transformation pour que la longueur du message soit congruente à *896 modulo 1024* ou $(32 + y)\,mod\,1024 = 896$. Donc : $y = 896 - 32 = 864$; $(32 + 864)\,mod\,1024 = 896$ ainsi, pour réaliser une longueur de 1024 *bits*, il est essentiel d'ajouter 864 *bits* en commençant par (1) et compléter le reste avec 863 zéros :

$$[01010101\,01010001\,01010100\,01010010] + [(1000)0000 \ldots 0000000(0)_{863}].$$

De ce fait, vient la nécessité d'ajouter les 128 *bits* manquants représentant la taille du message originale à 32 *bits* qui a une représentation de 20 en hexadécimale. Cela permet de compléter la forme totale du message en 1024 *bits*. Chaque chiffre hexadécimal représente quatre *bits*. Donc la forme en hexadécimale du message finale devient :

$$[(55515452)(8)0000000]_1\,[0000000000000000]_2 \ldots [00000000000000(20)]_{16}$$

le *8* affiché dans le schéma *Hex* est justifié à partir de la représentation binaire *1000* qui vient directement après celle du message m. La représentions du concept en fonction *preprocess_ data* est dans l'annexe D. Les résultats intermédiaires et finaux correspondant à la fonction de hachage sont conservés

par des registres de 512 *bits*, ces derniers peuvent être représentés sous la forme de huit registres 64 *bits* : a_0, b_0, c_0, d_0, e_0, f_0, g_0, h_0. Il est indispensable de diviser le message de 1024 *bits* en $n = 16$ mots de 64 *bits* : $w_0 \rightarrow w_{n-1}$, puis utiliser la formule discutée précédemment w_i pour calculer le reste des mots pour les 80 tours R :

$$[w_0 \quad + \quad w_1 \quad ... \quad w_{n-1}] \quad + \quad [w_n \quad ... \quad w_i \quad ... \quad w_{R-1}].$$

1 - Premier tour :
Dans le premier tour, on calcule $t_1 = h_0 + Ch(e_0, f_0, g_0) + \sum_1^{512}(e_0) + w_0 + k_0$ et $t_2 = \sum_0^{512}(a_0) + Maj(a_0 + b_0 + c_0)$:

$$k_0 = 428A2F98D728AE22$$

$$Ch(e_0, f_0, g_0) = (e_0 \wedge f_0) \oplus (-e_0 \wedge g_0) \tag{3.11}$$

$$\sum_1^{512} = ROTR^{14}(e_0) \oplus ROTR^{18}(e_0) \oplus ROTR^{41}(e_0) \tag{3.12}$$

$$Maj(a_0 + b_0 + c_0) = (a_0 \wedge b_0) \oplus (a_0 \wedge c_0) \oplus (b_0 \wedge c_0) \tag{3.13}$$

$$\sum_0^{512} = ROTR^{28}(a_0) \oplus ROTR^{34}(a_0) \oplus ROTR^{39}(a_0) \tag{3.14}$$

$$a_1 = t_1 + t_2. \tag{3.15}$$

La représentation du concept en code *t1_ t2*, voir l'annexe D. Depuis la (Fig. 3.8), on remarque que les valeurs (Tab. 3.3) sont avancées d'une position vers la droite donc :

a_1	$b_1 = a_0$	$c_1 = b_0$	$d_1 = c_0$	$e_1 = d_0$	$f_1 = e_0$	$g_1 = f_0$	$h_1 = g_0$

TABLE 3.3 – Transformations du premier tour

2 - Deuxième tour :
En ce qui concerne le deuxième tour, ce dernier est similaire au premier, on calcule t_1 et t_2 en utilisant les mêmes formules vu précédemment sauf que l'on applique les nouvelles valeurs générées pour effectuer le calcul.

$$k_1 = 3956C25BF348B538$$

$$t_1 = h_1 + Ch(e_1, f_1, g_1) + \sum_1^{512}(e_1) + w_1 + k_1 \tag{3.16}$$

$$t_2 = \sum_0^{512}(a_1) + Maj(a_1 + b_1 + c_1) \tag{3.17}$$

$$a_2 = t_1 + t_2. \tag{3.18}$$

Après le calcule de la valeur a_2 la progression de la position des variables (Tab. 3.4) change comme suit :

a_2	$b_2 = a_1$	$c_2 = b_1$	$d_2 = c_1$	$e_2 = d_1 + t_1$	$f_2 = e_1$	$g_2 = f_1$	$h_2 = g_1$

TABLE 3.4 – Transformations du deuxième tour

Cependant, ce processus continuera de la même façon jusqu'au tour 79. En concaténant les valeurs de la sortie *ABCDEFGH*, on obtient le résultat (Tab. 3.5), (Tab. 3.6) du hachage final qui a une taille de *512-bits*.

$A = a_0 + a_{79}$	$B = b_0 + b_{79}$	$C = c_0 + c_{79}$	$D = d_0 + d_{79}$
$E = e_0 + e_{79}$	$F = f_0 + f_{79}$	$G = g_0 + g_{79}$	$H = h_0 + h_{79}$

TABLE 3.5 – Transformations au tour 80

$A = 81FE2138A71E7A50$	$B = 8A0DDC6E8F643F48$
$C = A926365748E08497$	$D = CEE6F599235A8FC7$
$E = 23949F40648CCE56$	$F = 2000A6327DB681F7$
$G = 2C1DE8308DB005CE$	$H = 2B379D7B270ED085$

TABLE 3.6 – Résultats Finaux

3.5 La complexité computationnelle et la transformée de Fourier

La transformée de Fourier quantique est à la base de plusieurs algorithmes, notamment l'algorithme de factorisation quantique de Shor. RSA est l'un des cryptosystèmes asymétriques les plus utilisés, dont la sécurité repose sur la difficulté computationnelle de factoriser le produit de deux grands nombres premiers. RSA est largement utilisé dans les communications TLS (anciennement SSL, pour « Secure Sockets Layer »). Il sert principalement à chiffrer la clé de session, qui est ensuite utilisée pour chiffrer et déchiffrer les données échangées entre un client et un serveur. Soit k la longueur de la valeur d'entrée n. Puisque $k = log_2 n$, une complexité de $O(\sqrt{n})$ est équivalente à $O(\sqrt{2^k})$. Etant donné que $\sqrt{2^k} = 2^{\frac{k}{2}}$, cette complexité est évidemment exponentielle en termes de longueur d'entrée. L'algorithme de Shor exploite la périodicité, car un problème de factorisation peut être converti en un problème de recherche de période en temps polynomial. Cet algorithme quantique factorise efficacement des nombres entiers via une méthode qui calcule la période de $f(x) = a^x \, mod \, N$.

La factorisation des grands nombres entiers est un problème mathématique pour lequel il n'existe pas actuellement de solution connue efficace en temps polynomial sur un ordinateur classique. Soit *F(x)* un problème correspondant à trouver un facteur d'un grand nombre donné. La classification exacte de ce problème dans le cadre de P vs NP n'a pas été déterminée, bien qu'il ne soit pas prouvé comme étant NP-complet. L'algorithme de Shor permet une factorisation en temps polynomial sur un ordinateur quantique. Il appartient à la classe de complexité *BQP*, qui englobe les problèmes solvables en temps polynomial avec une probabilité d'erreur bornée sur un ordinateur quantique. Cependant, les ordinateurs quantiques ne peuvent pas résoudre tous les problèmes de type NP. Toutefois, l'algorithme de Shor est un exemple spécifique où un avantage quantique est démontré.

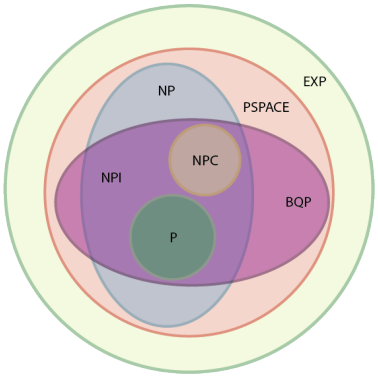


FIGURE 3.10 – Diagramme de la complexité computationnelle

La transformée de Fourier est une fonction mathématique qui décompose une forme d'onde périodique en une représentation alternative, caractérisée par les fonctions sinus et cosinus de fréquences variables. Cette représentation s'appelle le domaine fréquentiel et la forme d'onde d'origine s'appelle le domaine temporel. La transformée de Fourier démontre que toute forme d'onde périodique peut être réécrite comme la somme des sinusoïdes, et elle est utilisée dans de nombreux domaines de la science et de l'ingénierie, y compris le traitement du signal, le traitement d'image et l'analyse de données.

La transformée de Fourier discrète *DFT* est un algorithme de traitement numérique du signal qui calcule le spectre d'un signal discret de durée finie [45]. La *DFT* prend en entrée une séquence finie de nombres complexes et produit une séquence de même longueur de nombres complexes. La *DFT* est une transformation mathématique qui convertit une séquence finie d'échantillons équidistants d'une fonction en une séquence de même longueur d'échantillons équidistants de la transformée de Fourier en temps discret *DTFT*, qui est une fonction de fréquence à valeurs complexes. La transformée de Fourier rapide *FFT* est un algorithme efficace pour calculer la *DFT*. La *FFT* réduit le temps de calcul de la *DFT* en exploitant les symétries dans la matrice *DFT* et en utilisant un algorithme récursif efficace. La *FFT* est largement utilisée dans la pratique, car elle permet de calculer rapidement la *DFT* pour de grandes tailles d'entrée.

Pour démontrer l'importance de la transformée de Fourier rapide (*FFT*) une expérience est mise à l'épreuve visant à évaluer les performances de la transformée de Fourier quantique (*QFT*) à celles de la *FFT* (Fig. 3.11) en simulant l'application de la *QFT* à l'aide de *Qiskit* qui est un framework open-source dédié à la programmation quantique. Développé par IBM, *Qiskit* permet de créer, simuler et exécuter des algorithmes quantiques sur des ordinateurs classiques et des systèmes quantiques réels, néanmoins d'une manière limitée. Les principales mesures de comparaison sont le temps de traitement et l'utilisation des ressources, (voir l'annexe E). Contrairement à la *FFT* classique, qui opère sur des points de données classiques, la *QFT* opère sur un état quantique. L'état quantique est une superposition de tous les états possibles dans lesquels le système quantique peut se trouver, cependant, la *QFT* transforme cette superposition en une nouvelle superposition qui représente les fréquences présentes dans les données originales. La *QFT* est un composant clé de nombreux algorithmes quantiques, notamment l'algorithme de Shor pour la factorisation de grands nombres et l'estimation de phase quantique dont cette recherche abord en profondeur 3.7.

La *FFT* est un algorithme classique avec une complexité de calcul de $O(n \log n)$, qui est très efficace et largement utilisé dans le traitement du signal numérique. Dans notre expérience, lorsque nous exécutons la *FFT* sur des données bidimensionnelles $(n \times n)$ de différentes tailles avec $n = [4, 8, 16, 32]$, nous observons des temps de traitement cohérents avec cette complexité attendue. Par exemple, une taille de (16×16) donne $n = 256$, avec un nombre de *bits* de $256 \times (32 \times 2) = 16384$, donc la complexité temporelle est de $O(256 \log 256)$. Dans cette expérience théorique, nous simulons des opérations quantiques sur une machine classique à cause des limitations d'usage d'un calculateur quantique. Pour une image de

(16×16), nous aurions théoriquement besoin de 8 *qubits* puisque $\log_2(16 \times 16) = 8$. Depuis les résultats de la simulation, la *FFT* a clairement un temps de traitement meilleurs que ceux de la *QFT*. Pourtant, cela n'est pas tout à fait vraisemblable en théorie, car la surcharge de la simulation quantique sur un ordinateur classique n'est pas évidente. Un véritable ordinateur quantique n'aurait pas cette surcharge et pourrait facilement surpasser la *FFT* pour de très grands ensembles de données où l'avantage quantique devient distinct. Ainsi, l'ordinateur quantique peut théoriquement effectuer les calculs sur cet ensemble de données exponentiellement plus grand avec un nombre polynomial d'opérations, c'est là que réside la puissance de l'informatique quantique.

La *QFT* exploite les *qubits* en s'appuyant sur des propriétés fondamentales de la mécanique quantique, notamment la superposition et l'intrication 2.4.3. La complexité de la mise en œuvre de la *QFT* dépend du nombre de portes quantiques nécessaires, qui croît quadratiquement avec l'augmentation du nombre de *qubits*. Ce phénomène s'explique par le fait que chaque *qubit* ajouté double la dimension de l'espace d'état utilisé par l'ordinateur quantique, amplifiant ainsi exponentiellement la capacité de représentation des données. Ainsi, pour un système de n *qubits*, il est possible de représenter 2^n configurations différentes, permettant à un ordinateur quantique de manipuler un volume de données exponentiellement supérieur à n *bits* classiques. Il est important de noter que la *QFT* peut être appliquée à un nombre quelconque de *qubits*, et elle effectue une transformation linéaire sur les amplitudes des états quantiques.

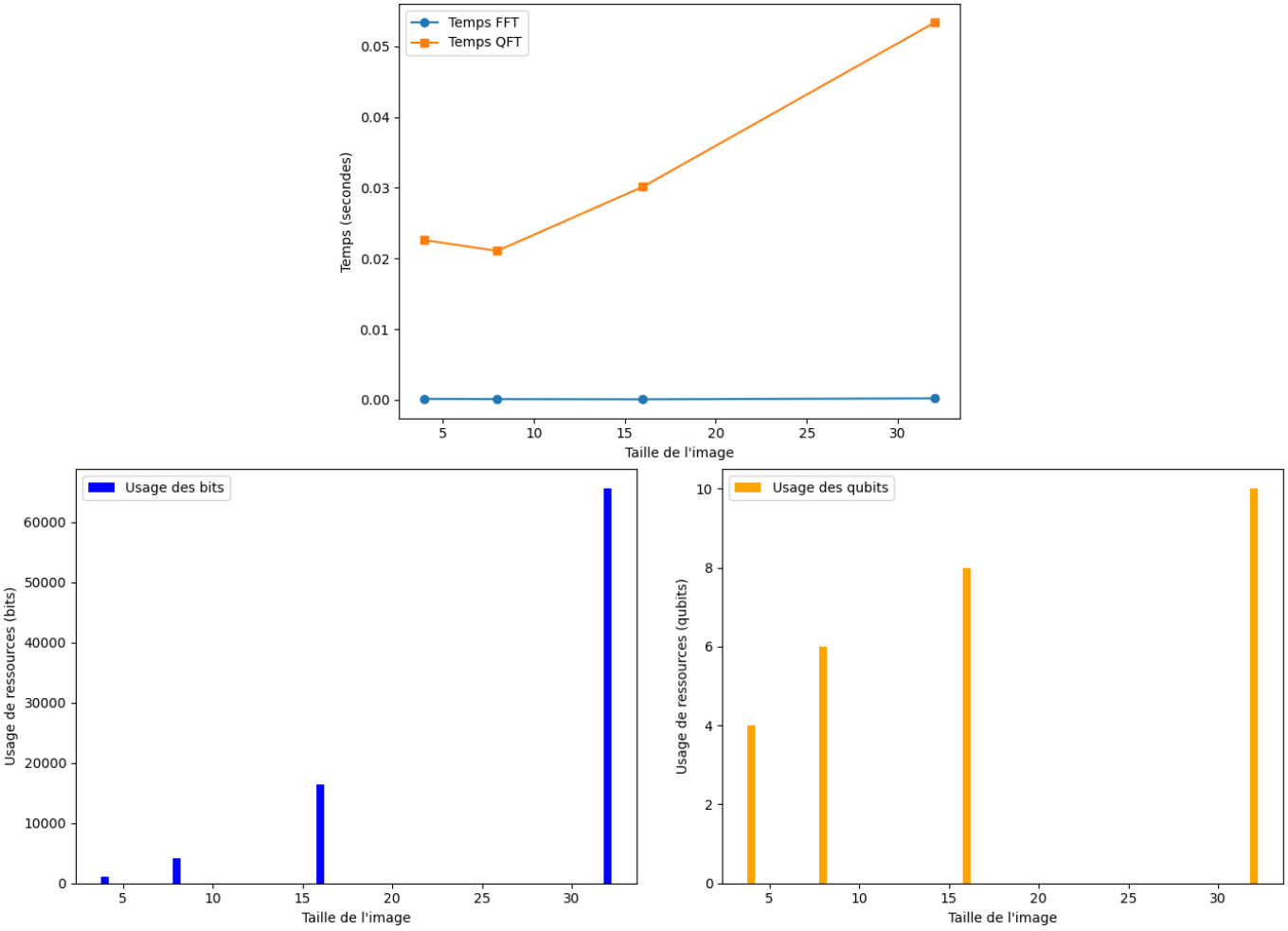


FIGURE 3.11 – Expérience comparative des vitesses et taux d'utilisation de ressource du QFT et du FFT sur des machines classiques.

Cette expérience doit être considérée comme une observation théorique illustrant le potentiel des algorithmes quantiques plutôt que comme une référence de performance pratique. Les véritables capacités de l'informatique quantique ne pourraient être réalisées que lorsque les processeurs quantiques seraient capables d'exécuter efficacement des algorithmes quantiques comme la *QFT* sur une grande échelle, qui sont actuellement limités par des contraintes technologiques telles que le nombre de *qubits* et la gestion du bruit quantique 2.4.10. Essentiellement, bien que la simulation fournisse un cadre utile pour comprendre les principes théoriques de l'informatique quantique, elle n'est pas représentative de la puissance ou de l'efficacité réelle du calcul quantique. Les résultats obtenus sont spécifiques à l'environnement de la simulation et ne doivent pas être extrapolés aux capacités informatiques quantiques réelles.

La formule de la transformée de Fourier (FT) :

$$X(\omega) = \int_{-\infty}^{\infty} x(t) e^{-i\omega t} dt \tag{3.19}$$

où $x(t)$ est le signal en temps continu et $X(\omega)$ est sa transformée de Fourier continue.

La formule de la transformée discrète de Fourier (DFT) :

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i k n}{N}} \tag{3.20}$$

où x_n est le signal discret de longueur N et X_k est sa DFT.

Tel que mentionné précédemment, la transformée de Fourier rapide FFT est une méthode algorithmique permettant de calculer la Transformée Discrète de Fourier DFT de manière plus efficace. Basé sur le principe de « diviser pour mieux régner », cet algorithme exploite les propriétés de périodicité et de symétrie des fonctions exponentielles complexes pour décomposer récursivement la DFT d'un signal de taille composite, représentée par $N = N_1 N_2$, en des DFTs de dimensions réduites N_1 et N_2. L'algorithme de *Cooley-Tukey* [6] demeure l'une des implémentations les plus reconnues et utilisées de la FFT.

3.6 La transformée de Fourier quantique *QFT*

La transformée de Fourier quantique *QFT* est l'implémentation quantique de la *DFT* sur les amplitudes d'une fonction d'onde. La *QFT* est un opérateur unitaire qui agit sur les états de base de calcul d'un système quantique et les transforme en états définis dans la base de Fourier. La *QFT* a de nombreuses applications en informatique quantique, et utilisée dans divers algorithmes quantiques, y compris l'algorithme de Shor pour la factorisation des grands nombres entiers et l'algorithme d'estimation de phase quantique. La transformation de fourier discrète prend en entrée un vecteur de N nombres complexes $[x_0, x_1, \ldots x_{N-1}]$ et sort N nombres complexes transformés $[y_0, y_1, \ldots y_{N-1}]$ [36]. Voici la représentation mathématique du concept :

$$U_{QFT} = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} \sum_{k=0}^{N-1} e^{\frac{2\pi i j k}{N}} |j\rangle\langle k|. \tag{3.21}$$

La transformée de Fourier quantique sur un seul *qubit* (QFT_1) est simplement la matrice d'identité, puisqu'un seul *qubit* n'a qu'un seul état de base et qu'il n'y a pas de transformation non triviale à effectuer :

$$QFT_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I. \tag{3.22}$$

La transformée de Fourier quantique est une généralisation de la transformation d'Hadamard qui est exposée dans la partie 2.4.6.2 sauf que la *QFT* traite aussi la phase du système quantique. Prenons le cas de QFT_2 avec $M = 2$ et $\omega = e^{2\pi i/M} = e^{\pi i} = -1$ alors :

$$QFT_2 = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & \omega \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{3.23}$$

Ce qui est remarquable est que QFT_2 est égal à H_1. Voici autres exemples de QFT_n avec différentes valeurs de *qubits* :

$$QFT_3 = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 & 1 \\ 1 & \omega^2 & \omega \\ 1 & \omega & \omega^2 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & i \\ 1 & i & -1 \end{bmatrix}, \tag{3.24}$$

où $\omega = e^{2\pi i/3}$ est la racine cubique de l'unité.

La matrice représentative de QFT_4, en utilisant les racines de l'unité $\omega_4 = e^{2\pi i/4}$; $\omega_4 = i$, est :

$$QFT_4 = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix}. \tag{3.25}$$

Comme il est observable depuis les démonstrations matricielles du QFT, les types particuliers de phases sont les racines primitives de l'unité ω représenté par un nombre complexe $\omega = e^{\frac{2\pi i}{n}}$ [53], voici un exemple (Fig. 3.12) pour $n = 5$.

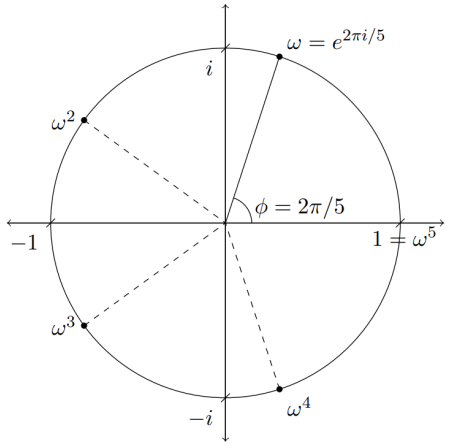


FIGURE 3.12 – Diagramme représentant les cinq nombres complexes ω pour $n = 5$

Cela nous permet de reformuler le QFT en opérateur linéaire :

$$QFT_N = \frac{1}{\sqrt{N}}\begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2N-2} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3N-3} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2N-2} & \omega^{3N-3} & \cdots & \omega^{(N-1)^2} \end{bmatrix}. \tag{3.26}$$

La QFT est réversible, puisque la transformation est unitaire :

$$\hat{F}^\dagger \hat{F} = \hat{I}. \tag{3.27}$$

3.7 La recherche de la période et l'estimation de la phase

La recherche de la période « r » est la base de l'algorithme de Shor, si la période est connue, il est possible de factoriser rapidement des nombres entiers. Trouver la période dépend fortement de la capacité de l'ordinateur quantique à réaliser la superposition. La transformée de Fourier quantique nous aide à trouver les résultats sous forme de probabilités. La recherche de la période peut également être effectuée en utilisant l'approche classique, mais peut être effectuée de manière plus efficace et rapide avec des ordinateurs quantiques. L'approche classique pour déterminer la période d'une fonction consiste à analyser la fonction elle-même pour identifier tout motif ou cycle répétitif. Cela peut être fait en utilisant certaines techniques mathématiques clés. Une de ces techniques consiste à utiliser le concept de périodicité, qui est la propriété d'une fonction qui se répète après un certain intervalle de temps.

Mathématiquement, une fonction *f(x)* est dite périodique de période P si *f(x) = f(x+P)* pour toutes les valeurs de x [50], en d'autres termes la fonction a les mêmes valeurs en chaque point séparé par une distance de P. Cela signifie que si nous pouvons trouver la plus petite valeur de P pour laquelle cette propriété est vraie, nous avons trouvé la période de la fonction. Une autre technique consiste à utiliser l'analyse de Fourier, qui est une méthode de décomposition d'une fonction complexe en parties plus simples appelées séries de Fourier. La série de Fourier [51] représente une fonction comme une somme de fonctions sinus et cosinus et peut-être utilisée pour analyser la périodicité de la fonction.

Voici les équations qui sont utilisées dans l'approche classique pour déterminer la période d'une fonction classiquement :

Condition de périodicité :

$$f(x) = f(x+P), \quad \forall x \in \mathbb{R}. \tag{3.28}$$

Représentation en série de Fourier :

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty}\left[a_n \cos\left(\frac{2\pi n x}{P}\right) + b_n \sin\left(\frac{2\pi n x}{P}\right)\right], \tag{3.29}$$

où a_0, a_n et b_n sont les coefficients de Fourier, qui déterminent la forme de la fonction. La période de la fonction est donnée par P, et la somme est prise sur tous les entiers positifs n de 1 à l'infini. En utilisant ces équations, nous pouvons analyser la fonction et déterminer sa période en utilisant une approche classique.

En ce qui concerne l'algorithme de Shor, il est possible de le représenter en 5 étapes :

Étape 1 :
Choisir un nombre $1 < a < N$ tel qu'il soit relativement premier avec « N » , ce qui signifie pgcd(a, N) = 1, où la fonction pgcd() désigne le plus grand diviseur commun. Si le $pgcd(a, N) \neq 1$ alors k est un facteur de N avec l'autre facteur étant $\frac{N}{k}$ et l'algorithme se termine ; sinon, passez à l'étape 2.

Étape 2 :
Avec la valeur « a » déterminer la période « r » de l'équation $a \bmod N$ telle que $a^r \bmod N \equiv 1$.

Étape 3 :
Si r est impair, retourner a l'étape 1. Si r est pair, passez à l'étape 4.

Étape 4 :
Depuis $a^r - 1 \equiv 0 \bmod N$, cela signifie que $a^r - 1$ est un multiple de N, donc il doit exister une valeur entière k , telle que $a^r - 1 = Nk$. Puisque r est pair, l'équation ci-dessus peut être réécrite, comme :

$$(a^{\frac{r}{2}} - 1).(a^{\frac{r}{2}} + 1) = pq. \tag{3.30}$$

Étape 5 :
Enfin, puisque ni $a^{\frac{r}{2}}-1$ ni $a^{\frac{r}{2}}+1$ sont congrues à $0\,mod\,N$, et $N=\frac{a^{\frac{r}{2}}-1}{p}.\frac{a^{\frac{r}{2}}+1}{q}$, alors on peut dire que :

$$p=gcd(a^{\frac{r}{2}}-1,N);\ q=gcd(a^{\frac{r}{2}}+1,N). \tag{3.31}$$

Ainsi p et q ont été déterminés. L'étape 2 est considérée comme la plus importante et aussi la plus difficile à exécuter classiquement, surtout lorsque N est un grand nombre. Cependant, trouver la période de N est opéré sur un ordinateur quantique grâce à la superposition qui accélére le temps de la fonction de recherche de la période. Principalement, l'utilisation de la propriété de superposition dans ce cas permet de réaliser plusieurs tests rapides $a^r\,mod\,N\equiv 1$ où a et N sont des entiers positifs, a est inférieur à N et ils n'ont aucun facteur commun. Comme il est mentionné précédent, l'analyse de la période est un problème mathématique qui consiste à trouver la période d'une fonction. Une période r est le plus petit entier positif p tel que la fonction se répète après p itérations. Par exemple ; la fonction $f(x)=x\,mod\,3$ a une période de 3, car elle se répète toutes les 3 itérations : $f(1)=1, f(2)=2, f(3)=0, f(4)=1, f(5)=2, f(6)=0, f(7)=1, f(8)=2,\ldots$, et ainsi de suite.

Pour trouver la période r de la fonction $f(x)=a^x \bmod N$ avec l'algorithme d'estimation de la phase quantique *QPE* [26], il est possible de décrire la procédure en 5 étapes [37] :

1. Création d'une superposition de toutes les entrées possibles de *f(x)*. Cela implique d'appliquer la transformée d'Hadamard au premier registre de l'état quantique $|u_s\rangle$ pour créer une superposition de tous les états possibles.

2. Application de la fonction quantique U à cette superposition. On applique la fonction U à l'état $|u_s\rangle$ pour obtenir une superposition d'états de la forme $|a^k \bmod N\rangle$, avec des coefficients multipliés par un facteur de phase.

3. Application de la transformée de Fourier quantique inverse $QFT^\dagger$ au premier registre à l'ensemble de l'état quantique résultant de l'opération $U|u_s\rangle$ pour obtenir une nouvelle superposition d'états avec des coefficients associés qui contiennent des informations sur la période r de la fonction *f(x)*.

4. Mesure du premier registre dans la base de calcul de la superposition d'états obtenus à l'étape 3 pour obtenir une estimation de la phase θ.

5. Utilisation de la technique de la fraction continue à la valeur estimée de θ obtenue à partir de la mesure de l'étape 4 pour obtenir la période r de la fonction *f(x)*

Le circuit quantique utilisé comprend deux registres quantiques (Fig. 3.13) : un registre de contrôle et un registre cible. Le registre de contrôle est initialisé dans une superposition de toutes les valeurs possibles de « s », où s est un entier compris entre 0 et r-1, et « r » est la période de la fonction *f(x)*. Le registre cible est initialisé à l'état $|1\rangle$.

$$|u_s\rangle=\frac{1}{\sqrt{r}}\sum_{j=0}^{r-1}e^{\frac{2\pi isj}{r}}|j\rangle. \tag{3.32}$$

L'équation 3.32 est la définition générale de l'état $|u_s\rangle$. L'état $|u_s\rangle$ est un état quantique utilisé dans l'algorithme de Shor pour trouver la période de la fonction $f(x)=a^x \bmod N$. L'état $|u_s\rangle$ est défini comme une superposition d'états de la forme $|a^k \bmod N\rangle$, où k varie de 0 à r-1. Les coefficients de chaque état de base dans la superposition sont des nombres complexes qui dépendent des entiers s et k qui reflètent la périodicité de la fonction.

Démarche :

$$U|u_s\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{\frac{2\pi i s k}{r}} |a^k \bmod N\rangle \tag{3.33}$$

Pour appliquer la fonction $f(x) = a^x \bmod N$ à $|u_s\rangle$, on utilise l'opérateur unitaire U défini comme $U|j\rangle = |aj \bmod N\rangle$. L'application de U à $|u_s\rangle$ donne un nouvel état qui est une superposition d'états de la forme $|a^k \bmod N\rangle$, mais avec des coefficients différents. Plus précisément, le coefficient associé à $|a^k \bmod N\rangle$ est multiplié par un facteur de phase $e^{2\pi i s k/r}$. Ainsi, la formule 3.33 représente l'état obtenu en appliquant l'opérateur unitaire U à $|u_s\rangle$.

$$QFT^\dagger : |x\rangle \to \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-\frac{2\pi i s k}{r}} |k\rangle \tag{3.34}$$

Lorsque nous appliquons la $QFT^\dagger$ à l'ensemble de l'état quantique issu de l'opération $U|u_s\rangle$ 3.33, une métamorphose de l'état initial se produit. Cette application transforme l'état quantique original en un nouvel état caractérisé par une superposition d'états de la forme $|k\rangle$. Cette transformation n'est pas simplement un changement d'état, mais une réinterprétation complète des probabilités et des phases associées à l'état quantique. Le résultat de cette transformation est fondamental pour extraire des informations pertinentes concernant la période r de la fonction étudiée. En mesurant ce nouvel état quantique, des données cruciales sur la structure périodique sous-jacente de la fonction peuvent être déduites, ouvrant ainsi la voie à une compréhension plus profonde des dynamiques quantiques impliquées [39].

Pour illustrer cet algorithme, considérons l'exemple de la fonction $f(x) = 7^x \bmod 15$, avec $a = 7$ et $N = 15$ [25].

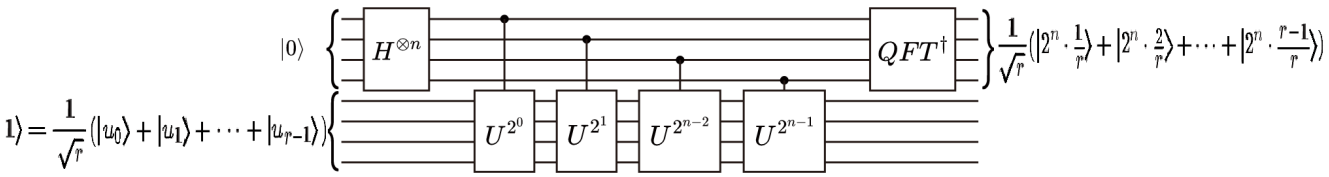


FIGURE 3.13 – Schéma du circuit de l'algorithme quantique de Shor

La (Fig. 3.13) montre le schéma du circuit quantique utilisé dans l'algorithme de Shor.

$$\begin{aligned}
\Big(\quad |u_0\rangle &= \frac{1}{\sqrt{r}}(|1\rangle + |7\rangle + |4\rangle + |13\rangle) \\
+|u_1\rangle &= \frac{1}{\sqrt{r}}(|1\rangle + e^{-\frac{2\pi i}{r}}|7\rangle + e^{-\frac{4\pi i}{r}}|4\rangle + e^{-\frac{6\pi i}{r}}|13\rangle) \\
+|u_2\rangle &= \frac{1}{\sqrt{r}}(|1\rangle + e^{-\frac{4\pi i}{r}}|7\rangle + e^{-\frac{8\pi i}{r}}|4\rangle + e^{-\frac{12\pi i}{r}}|13\rangle) \\
+|u_3\rangle &= \frac{1}{\sqrt{r}}(|1\rangle + e^{-\frac{6\pi i}{r}}|7\rangle + e^{-\frac{12\pi i}{r}}|4\rangle + e^{-\frac{18\pi i}{r}}|13\rangle) \quad \Big) = |1\rangle,
\end{aligned} \tag{3.35}$$

$$\begin{aligned}
|U_0\rangle \quad k = 0, s = 0,1,2,3 \quad &; |7^0 mod 15> = |1> \\
|U_1\rangle \quad k = 1, s = 0,1,2,3 \quad &; |7^1 mod 15> = |7> \\
|U_2\rangle \quad k = 2, s = 0,1,2,3 \quad &; |7^2 mod 15> = |4> \\
|U_3\rangle \quad k = 3, s = 0,1,2,3 \quad &; |7^3 mod 15> = |13>.
\end{aligned}$$

Les quatres états $|u_s\rangle$ qui apparaissent dans l'équation sont les superpositions d'états de la forme $|7^k \bmod 15\rangle$ pour k=0,1,2,3. Les coefficients devant chaque état sont déterminés par la $QFT^{\dagger}$. Voici l'implémentation du circuit (Fig. 3.14) sur un ordinateur quantique :

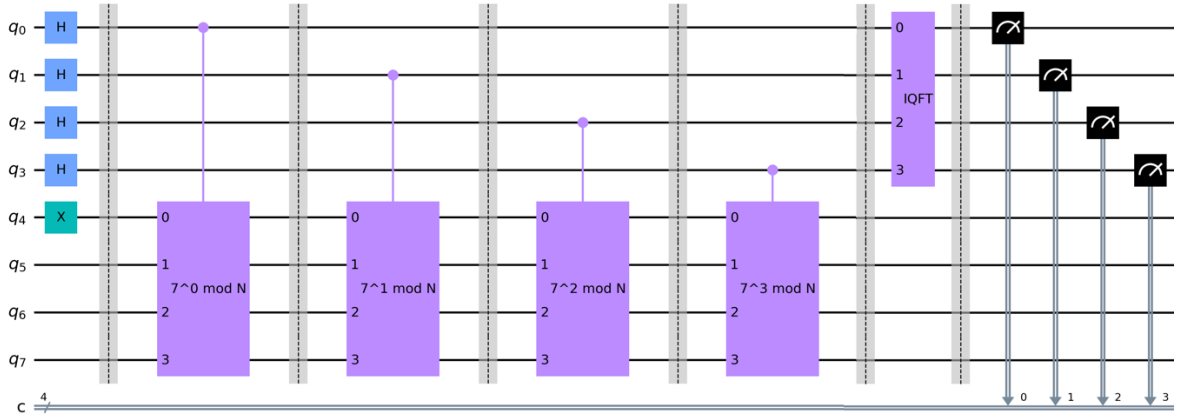


FIGURE 3.14 – Circuit quantique pour $N = 15$ avec $a = 7$ et 4 qubits pour le registre cible et de contrôle

Suite à la mesure du registre de contrôle, et en s'appuyant sur les états quantiques obtenus, il devient envisageable de déduire la périodicité intrinsèque de la fonction *f(x)* à travers l'extraction de la valeur de θ. Cette phase, θ, entretient une relation mathématique déterminante avec la période r de la fonction *f(x)*, s'exprimant sous la forme : $\theta = \frac{s}{r}$, où s est une valeur entière aléatoirement choisie dans l'intervalle [0, r-1] [25]. Dans le but d'extraire la période r à partir de l'estimation de θ, on fait appel à la méthode sophistiquée des fractions continues. En substance, nous représentons θ au moyen d'un développement en fraction continue, qui s'articule sous la forme : $a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \ddots}}}$.

Il est alors possible de circonscrire ce développement après un nombre défini de termes, afin d'obtenir une approximation de θ caractérisée par un dénominateur de faible magnitude.

Les dénominateurs trouvés par l'algorithme d'estimation de phase (voir l'annexe F) donnent 2^4 candidates de période potentiel, car l'opération est faite sur 4 *qubits* (Tab. 3.7). Après une simplification, on obtient $r = [16, 4, 1, 2, 8]$. Dans notre cas, la meilleure période pour la fonction $f(x) = 7^x \bmod 15$ est la période $r = 4$.

Phase	*Fraction*	*Supposition pour r*
0.0625	1/16	16
0.5625	9/16	16
0.1875	3/16	16
0.75	3/4	4
0	0/1	1
0.5	1/2	2
0.3125	5/16	16
0.875	7/8	8
0.375	3/8	8
0.9375	15/16	16
0.4375	7/16	16
0.8125	13/16	16
0.25	1/4	4
0.6875	11/16	16
0.125	1/8	8
0.625	5/8	8

TABLE 3.7 – Fractions de phase et suppositions correspondantes pour r

Le dictionnaire des comptages (Fig. 3.15) représente les 16 résultats de mesure qui sont obtenus lors de l'exécution de la transformée de Fourier quantique inverse sur l'état quantique en question. Chaque clé du dictionnaire correspond à une représentation binaire du résultat de la mesure, et la valeur associée à chaque clé représente la fréquence ou le nombre de ce résultat particulier (Tab. 3.8). En analysant les résultats de mesure et en extrayant les informations de phase, l'algorithme peut identifier des valeurs potentielles pour la période et tenter de trouver les facteurs de N sur la base de ces valeurs.

Par exemple :

Résultat de mesure	*Nombre d'occurrences*
'0000'	61
'0001'	62
'0010'	61
'0011'	73
'0100'	63
⋮	⋮

TABLE 3.8 – Occurrences des résultats de mesure

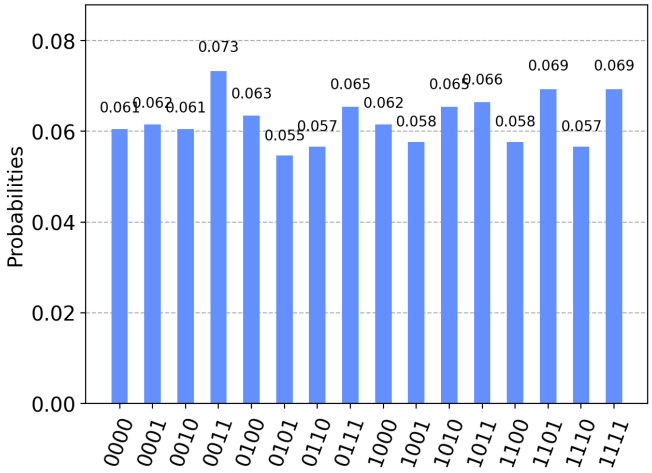


FIGURE 3.15 – Graphe des occurrences des résultats de mesure pour la factorisation de N avec N = 15, a = 7 et 4 qubits pour le registre cible et de contrôle

En utilisant cette information, on peut alors appliquer les deux formules de l'algorithme de factorisation de Shor (voir 3.31) pour déterminer les facteurs de N, L'annexe G décrit les tests rapides concernant les périodes trouvés (Tab. 3.9).

r	*Valeurs*
16	$q = 1$, $p = 15$
4	$q = 5$, $p = 3$
1	$q = 1$, $p = 15$
2	$q = 1$, $p = 3$
8	$q = 1$, $p = 15$

TABLE 3.9 – Résultats de la factorisation

Depuis les résultats simplifiés (Tab. 3.9), les mauvaises périodes dans ce cas sont $r = [16, 1, 8]$, tandis que la meilleure est $r = 4$. Pour extraire l'autre facteur de la période $r = 2$, il est possible d'utiliser «N// le facteur trouvé» alors 15//3 = 5. Dans ce cas, les valeurs p et q qui représentent le nombre N avec $r = 4$ sont obtenues par les formules suivantes :

$$p = gcd(7^{4/2} - 1, 15) = 3 \tag{3.36}$$

$$q = gcd(7^{4/2} + 1, 15) = 5 \tag{3.37}$$

3.8 Conclusion

Dans ce chapitre, nous nous sommes penchés sur le domaine de la cryptographie, en explorant ses objectifs et ses fondements. Nous avons effectué une analyse comparative de trois systèmes cryptographiques : RSA, AES et schémas hybrides. Notre attention s'est ensuite déplacée vers un examen détaillé du fonctionnement mathématique du cryptosystème RSA.

Nous avons également discuté de l'authentification et de l'intégrité des données, en soulignant l'utilisation du hachage et du cryptosystème RSA à ces fins. Plus précisément, nous avons exploré les opérations mathématiques impliquées dans le système de hachage SHA512.

De plus, nous avons exploré la complexité de calcul, en considérant à la fois les transformées de Fourier classiques et quantiques. Notre étude nous a amenés à approfondir les étapes, la méthodologie mathématique et la mise en œuvre de l'algorithme de Shor, qui est utilisé pour factoriser de grands nombres et trouver des nombres premiers. Nous avons soigneusement examiné comment l'algorithme utilise la recherche de périodes et l'estimation de phases, ce qui a entraîné une factorisation accélérée.

Pour illustrer ces concepts, nous avons fourni une explication théorique et pratique détaillée, ainsi que les circuits et les résultats correspondants, pour un exemple impliquant $N = 15$.

Ce chapitre témoigne de l'immense potentiel de l'informatique quantique et des algorithmes quantiques pour résoudre des problèmes mathématiques et informatiques complexes. Il met en évidence les capacités remarquables qu'ils offrent pour surmonter de tels défis.

Chapitre 4

Simulation et résultats

L'algorithme de *Shor* est un algorithme quantique visant à trouver les facteurs premiers d'un nombre composé. Cette méthode quantique a le potentiel d'accélérer considérablement le processus de la factorisation, qui est une étape clé dans de nombreux algorithmes de chiffrement moderne. L'algorithme fonctionne en utilisant des ordinateurs quantiques pour trouver la période d'une fonction, qui peut être utilisée pour trouver les facteurs premiers d'un nombre composé [25]. Je suis fier de vous présenter QURSA (Qubits + RSA), une application que j'ai créée et qui est maintenant hébergée et prête à être utilisée pour des simulations (Visiter l'application). QURSA est une implémentation avancée de l'algorithme de Shor, incluant la correction d'erreurs quantiques. J'ai développé cette application en utilisant le langage de programmation *Python* et la bibliothèque quantique *Qiskit* pour les opérations sur un ordinateur quantique IBM, comme présenté dans l'annexe H. L'algorithme de Shor, utilisé dans QURSA, demande un nombre significatif de *qubits*, car il s'agit d'un algorithme probabiliste nécessitant souvent plusieurs itérations pour obtenir le résultat correct.

L'application peut être divisée en trois parties principales : la partie classique, la partie quantique et la partie d'exécution parallèle pour accélérer la recherche des deux facteurs cibles.

4.1 Partie classique :

La partie classique consiste à définir le problème, à générer des valeurs aléatoires et à effectuer certains calculs classiques tels que l'inverse multiplicatif modulaire et le plus grand diviseur commun (pgcd).

4.1.1 Les fonctions classique :

Crypter : Cette fonction chiffre un message à l'aide d'une clé publique donnée RSA.

Modular_ multiplicative_ inverse : Cette fonction calcule l'inverse multiplicatif modulaire de deux entiers a et n pour générer la clé privé d.

value_ a : Cette fonction génère un entier aléatoire a tel que pgcd(a, N) = 1.

p_ q_ finder : Cette fonction trouve les facteurs P et Q de l'entier donné N grâce aux deux formules vu précédemment dans le chapitre III - Étape 5 (voir 3.31) de l'algorithme de Shor, puis calcule la clé privée d en utilisant p et q trouvé grâce à la période r.

4.2 Partie quantique :

La partie quantique implique la mise en place et l'exécution d'un circuit quantique qui implémente l'algorithme de Shor.

4.2.1 Les fonctions quantique :

initialize_ qubits : Cette fonction initialise le circuit quantique en appliquant des portes Hadamard aux *qubits* de contrôle et une porte X au *qubit* cible.

c_ modN : Cette fonction crée une porte U contrôlée qui effectue une exponentiation modulaire dans le circuit quantique.

modular_ exponentiation : Cette fonction applique les portes U contrôlés pour l'exponentiation modulaire dans le circuit quantique.

qft_ dagger : Cette fonction applique la transformée de Fourier quantique inverse *qft_ dagger* aux *qubits* de mesure.

error_ correction : Cette fonction applique une correction d'erreur par retournement de *bit* au circuit quantique en utilisant des portes $CNOT$ et des portes X pour corriger les erreurs potentielles dans les *qubits*. Cette méthode garantit que les *qubits* sont correctement préparés pour l'étape de la mesure et sépare les étapes de correction des erreurs par des barrières.

mesure : Cette fonction mesure la sortie du circuit quantique.

period_ finder : Cette fonction configure et exécute le circuit quantique qui implémente l'algorithme de Shor.

4.3 Partie d'exécution parallèle :

La partie d'exécution parallèle permet d'exécuter simultanément plusieurs instances de la fonction *p_ q_ finder* pour accélérer le processus de factorisation. Les composants principaux sont :

factor_ found : Une variable partagée qui indique si un facteur a été trouvé.

factor_ lock : Un verrou pour synchroniser l'accès à la variable partagée.

executor : Un *ThreadPoolExecutor* qui exécute plusieurs instances de la fonction *p_ q_ finder* simultanément.

4.4 Fonctionnement et approche globale :

Le code commence par importer les bibliothèques nécessaires pour les calculs quantiques et classiques. La variable partagée *factor_ found* et le verrou *factor_ lock* sont initialisés à des fins de synchronisation. L'entier N est l'entier cible à factoriser. Le nombre de *qubits* de contrôle *controll_ qubits*, de *qubits* cibles *target_ qubits* et la précision de fraction *fraction_ accuracy* sont définis en fonction des exigences du problème.

Le message est chiffré à l'aide de la clé publique RSA (65537, N) et de la fonction *Crypter*. Le message crypté est ensuite imprimé. La fonction *period_ finder* est appelée avec les paramètres *controll_ qubits*, *target_ qubits* et une valeur aléatoire a. La fonction renvoie un circuit quantique qui implémente l'algorithme

de Shor. Le circuit est exécuté sur un calculateur quantique et les comptages de sortie sont obtenus. Les phases mesurées sont extraites des comptages de sortie et converties en une liste de périodes simplifiées *r_ simplifier*.

La fonction *p_ q_ finder* est appelée avec le paramètre *r_ simplifier*. La fonction tente de trouver les facteurs P et Q de l'entier N en utilisant les périodes simplifiées. Si des facteurs sont trouvés, la clé privée d est calculée à l'aide de la fonction *Modular_ multiplicative_ inverse* puis le message crypté peut être facilement revlé. La fonction *p_ q_ finder* est exécutée simultanément à l'aide de *ThreadPoolExecutor*. Le programme attend que toutes les instances soient terminées et imprime les facteurs et la clé privée.

Par exemple ; voici un cas concret pour $N = 95441829405190059581$ qui a une longueur de 20. La clé publique RSA est dans ce cas : [95441829405190059581, 65537] (Fig. 4.1). Dans cette démonstration, le mot UQTR sera chiffré avec la clé publique qui donne la série : [81555774160116936128, 85692886208336599067, 28963831429370944553, 25683588750565219215], seul la clé privé peut déchiffrer ce message. La fonction qui va être la cible de la recherche de la période est : $U(x) = a^x \, mod \, 95441829405190059581$, voici les résultats :

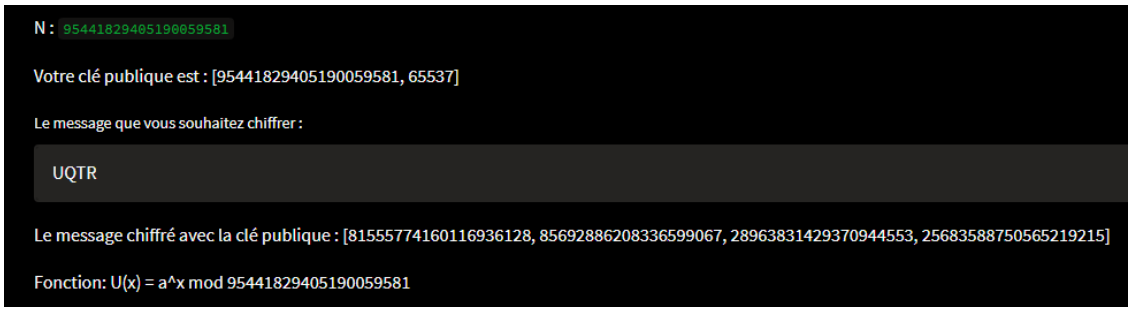
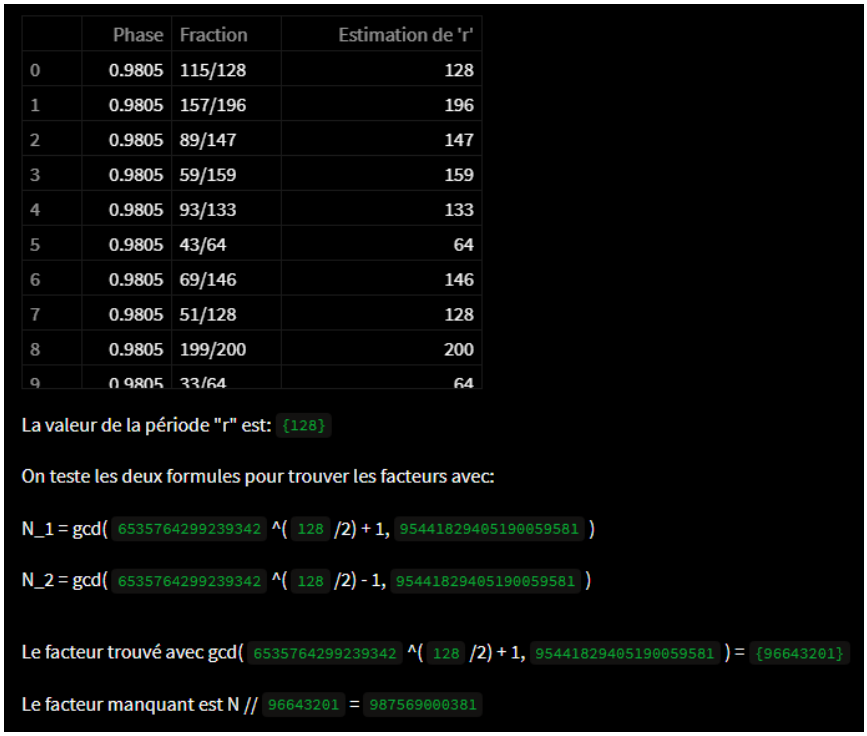


FIGURE 4.1 – Résultat du chiffrement RSA avec N = 95441829405190059581

Dans cette expérience, la partie classique de l'application a été accéléré avec 5 processus parallèles et avec une précision de 200 concernant l'opération des fractions continue. L'application donne les résultats suivants $a = 6535764299239342$ et la période est sélectionné depuis la table des périodes estimées par l'application, dans ce cas la période est de $r = 128$ donc, on teste les deux formules pour trouver les facteurs avec :

$$N_\alpha = gcd(6535764299239342^{(128/2)} + 1, 95441829405190059581)$$

$$N_\beta = gcd(23043750080952^{(128/2)} - 1, 95441829405190059581.)$$



	Phase	Fraction	Estimation de 'r'
0	0.9805	115/128	128
1	0.9805	157/196	196
2	0.9805	89/147	147
3	0.9805	59/159	159
4	0.9805	93/133	133
5	0.9805	43/64	64
6	0.9805	69/146	146
7	0.9805	51/128	128
8	0.9805	199/200	200
9	0.9805	33/64	64

FIGURE 4.2 – Le résultat de la période r depuis les phases calculé

Par la suite, dans ce cas, on trouve un seul facteur avec $N_\alpha = gcd(6535764299239342^{(128/2)}+1, 95441829405190059581)$ 96643201 (Fig. 4.2) alors avec l'opérateur *// : floor division operator* on peut trouver le deuxième facteur premier $95441829405190059581 \;//\; 96643201 = 987569000381$. Cependant, on conclut que N = 987569000381 × 96643201 et que la clé privée d = 26501829402354537473, calculé grâce à la fonction $Modular\_multiplicative\_inverse(65537, \phi)$ avec $\phi = (P-1) \times (Q-1)$ (Fig. 4.3). Alors avec la clé privée trouvée, on peut facilement déchiffrer le message et le révéler en clair, dans ce cas notre message est UQTR.

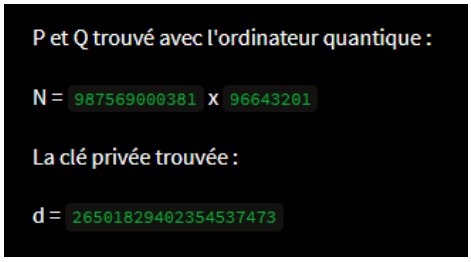


FIGURE 4.3 – Les facteurs et la clé privée trouvés pour l'expérience avec N = 95441829405190059581

Voici le circuit quantique (Fig. 4.4 & 4.5) responsable de la recherche de la période avec 4 *qubits* pour le registre cible (partie inférieure du circuit) et 8 *qubits* pour le registre de contrôle (partie supérieure du circuit).

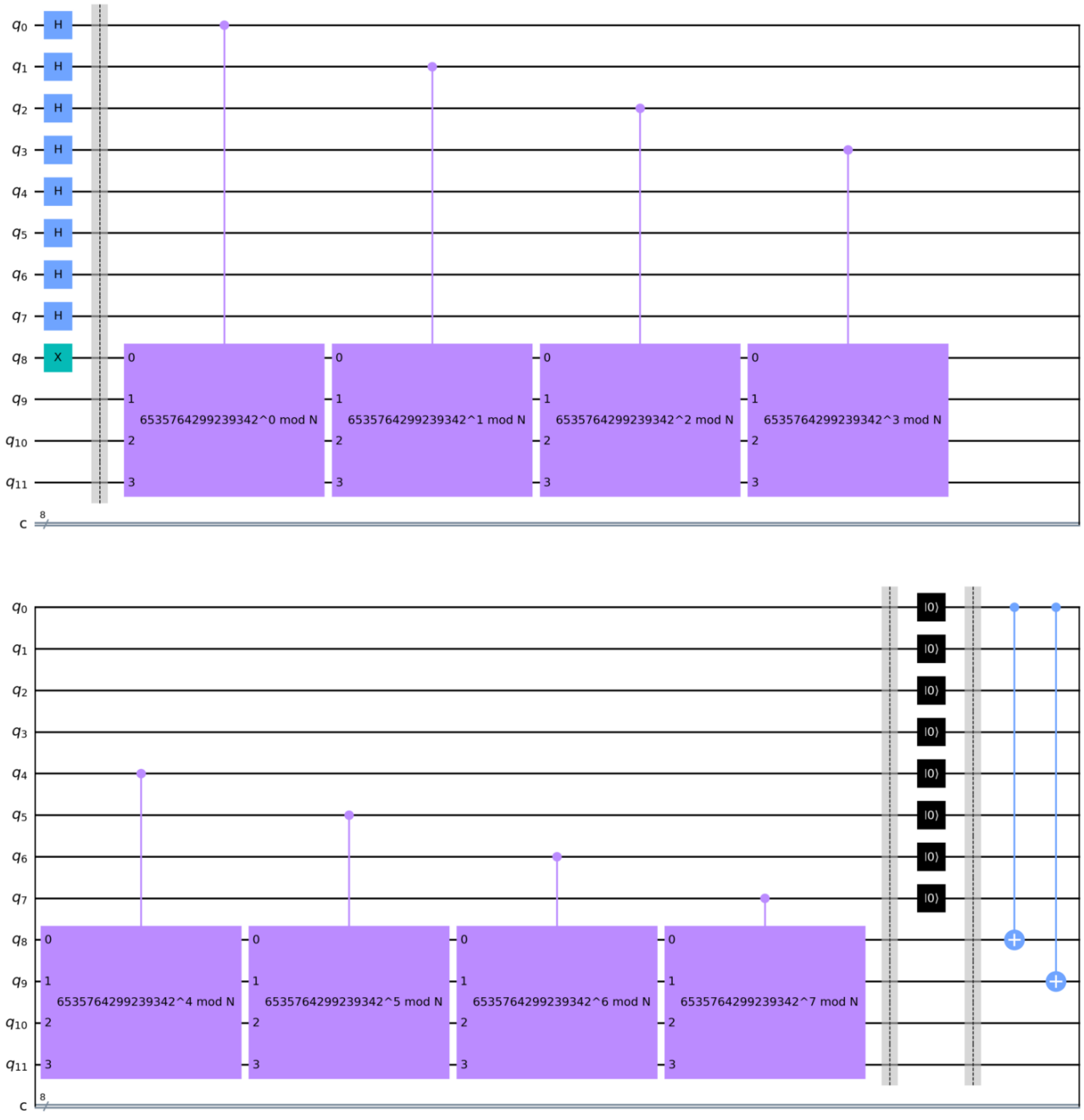


FIGURE 4.4 – Le circuit quantique de l'expérience

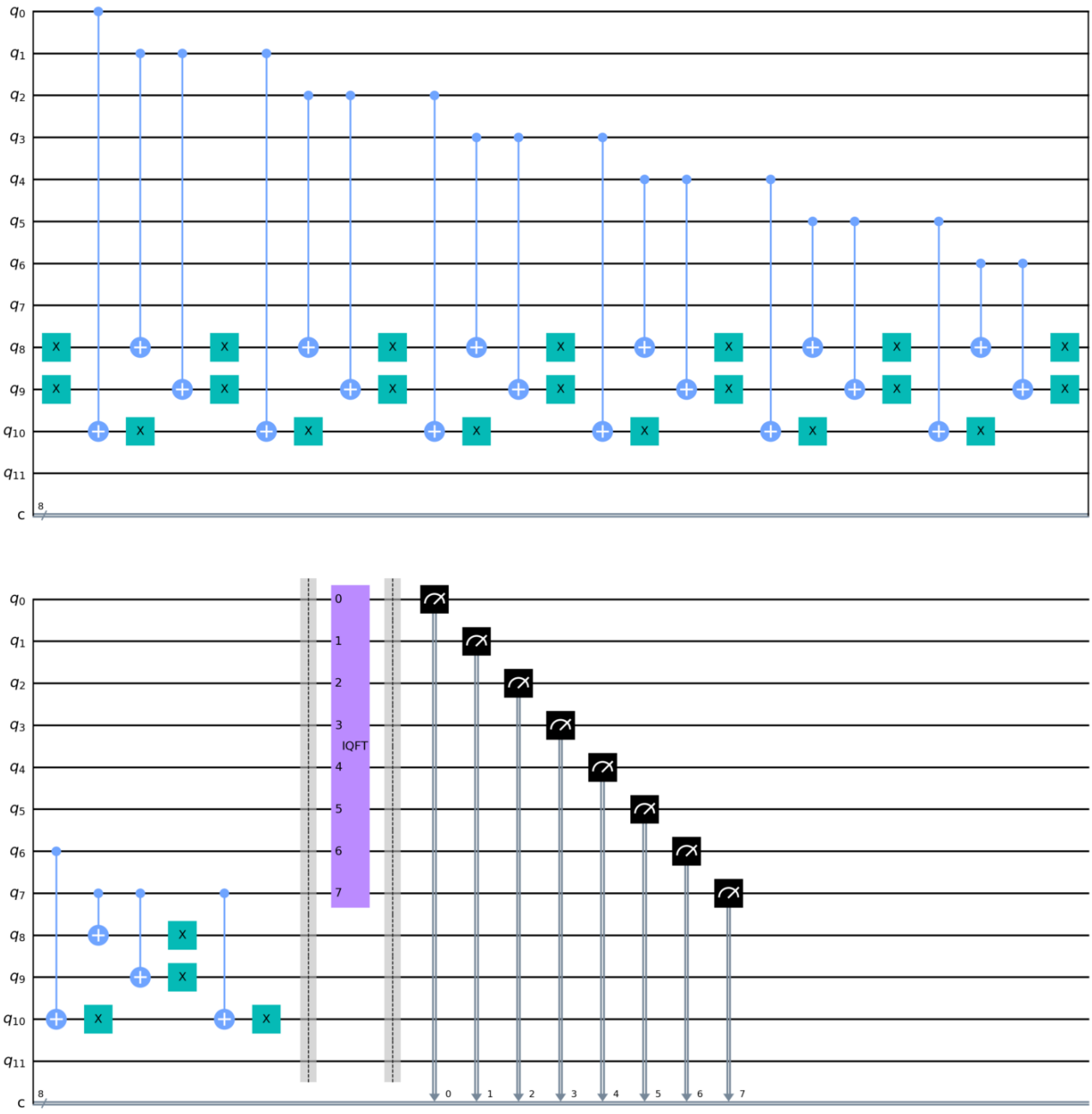


FIGURE 4.5 – Le circuit quantique de l'expérience - suite

4.5 Conclusion

Tout au long de ce chapitre, une analyse rigoureuse a été menée concernant l'exploitation de l'application QURSA dans le but de factoriser un nombre N comportant 20 chiffres, en s'appuyant à la fois sur des techniques quantiques et classiques. Au sein de cette exploration approfondie, l'application a habilement encodé le terme « UQTR », puis procédé à son décodage sans la moindre référence à la clé privée. Ce tour de force a été rendu possible grâce à la mise en œuvre méticuleuse de l'algorithme de Shor, conjuguée aux accélérations classiques, culminant dans la rupture d'une clé RSA de longueur 20. Les

données expérimentales recueillies témoignent de la capacité à factoriser des nombres d'une magnitude modeste, lorsque mis en parallèle avec des normes de sécurité moderne telles que RSA 2048 ou 4096.

Toutefois, il convient d'insister sur le fait que, bien que l'adaptation de QURSA à la factorisation de nombres de plus grande envergure soit théoriquement envisageable, sa concrétisation repose sur des avancées technologiques majeures dans le domaine des ordinateurs quantiques. Il s'agirait, en particulier, de disposer d'une quantité accrue de *qubits*, caractérisés par une forte intrication, une faible sensibilité au bruit quantique et à la décohérence, et épaulés par des mécanismes de correction d'erreurs hautement performants. En attendant ces avancées majeures, l'étude ici dévoilée s'établit comme une pierre angulaire dans la compréhension des potentialités et des frontières actuelles des algorithmes quantiques en matière de cryptographie contemporaine.

Annexe A

Simulation de la Croissance de Puissance Calculatoire : Ordinateurs Classiques vs Quantiques

```
import matplotlib.pyplot as plt

num_qubits_transistors = list(range(1, 30))

classical_power = num_qubits_transistors
quantum_power = [2**n for n in num_qubits_transistors]

plt.figure(figsize=(10, 6))

plt.plot(num_qubits_transistors, classical_power,
marker='o', linestyle='-', color='b', label='Puissance de calcul classique')

plt.plot(num_qubits_transistors, quantum_power, marker='o',
linestyle='-', color='r', label='Puissance de calcul quantique')

plt.yscale('log')
plt.xlabel('Nombre de qubits/transistors')
plt.ylabel('Puissance de calcul (echelle logarithmique)')
plt.legend()
plt.grid(True, which="both", ls="--")
plt.xticks(num_qubits_transistors)
plt.tight_layout()
plt.show()
```

Annexe B

RSA-AES-Hybrid

```
import time
import os
import matplotlib.pyplot as plt
from Crypto.Cipher import AES, PKCS1_OAEP
from Crypto.PublicKey import RSA
from Crypto.Random import get_random_bytes

def measure_time(func, *args):
    start_time_ns = time.process_time_ns()
    func(*args)
    end_time_ns = time.process_time_ns()
    return end_time_ns - start_time_ns

def rsa_chunk_encrypt_decrypt(key, message, chunk_size):
    cipher = PKCS1_OAEP.new(key)
    chunks = [message[i:i + chunk_size] for i in range(0, len(message),
    chunk_size)]
    encrypted_chunks = [cipher.encrypt(chunk) for chunk in chunks]

    cipher = PKCS1_OAEP.new(key)
    for chunk in encrypted_chunks:
        cipher.decrypt(chunk)

def aes_encrypt_decrypt(key, message):
    cipher = AES.new(key, AES.MODE_EAX)
    ciphertext, tag = cipher.encrypt_and_digest(message)
    cipher = AES.new(key, AES.MODE_EAX, nonce=cipher.nonce)
    cipher.decrypt_and_verify(ciphertext, tag)

def hybrid_encrypt_decrypt(rsa_key, aes_key, message):
    # Encrypt the AES key using RSA
    cipher_rsa = PKCS1_OAEP.new(rsa_key)
    enc_aes_key = cipher_rsa.encrypt(aes_key)

    # Encrypt the message using AES
    cipher_aes = AES.new(aes_key, AES.MODE_EAX)
```

```
    ciphertext, tag = cipher_aes.encrypt_and_digest(message)

    # Decrypt the AES key using RSA
    cipher_rsa = PKCS1_OAEP.new(rsa_key)
    dec_aes_key = cipher_rsa.decrypt(enc_aes_key)

    # Decrypt the message using AES
    cipher_aes = AES.new(dec_aes_key, AES.MODE_EAX, nonce=cipher_aes.nonce)
    cipher_aes.decrypt_and_verify(ciphertext, tag)

def test_encryption(sizes, rsa_key, aes_key):
    rsa_times, aes_times, hybrid_times = [], [], []
    max_rsa_chunk_size = 256 - 42  # For 2048-bit RSA key with OAEP padding

    for size in sizes:
        message = os.urandom(size)
        rsa_times.append(measure_time(rsa_chunk_encrypt_decrypt, rsa_key,
        message, max_rsa_chunk_size))
        aes_times.append(measure_time(aes_encrypt_decrypt, aes_key, message))
        hybrid_times.append(measure_time(hybrid_encrypt_decrypt,
        rsa_key, aes_key, message))
    return rsa_times, aes_times, hybrid_times

# Generate keys
rsa_key = RSA.generate(2048)
aes_key = get_random_bytes(16)  # 128-bit AES key

# Sizes in bytes
sizes = [64, 128, 256, 512, 1024, 2048]

# Testing the encryption
t_enc = test_encryption(sizes, rsa_key, aes_key)
rsa_results, aes_results, hybrid_results = t_enc

# Plotting the results
plt.plot(sizes, rsa_results, 'r', label='RSA')
plt.plot(sizes, aes_results, 'g', label='AES')
plt.plot(sizes, hybrid_results, 'b', label='Hybrid RSA+AES')
plt.xlabel('File Size (bytes)')
plt.ylabel('Time (ns)')
plt.title('Encryption Time Comparison')
plt.legend()
plt.show()
```

Annexe C

Algorithme - RSA (Simplifié)

```
from math import gcd

def Primes(p,q,e):
  phi = (p - 1) * (q - 1)
  if gcd(e, phi) == 1 and 1 < e < phi:
    print(True)
  else:
    print(False)

def Modular_multiplicative_inverse(e, phi):
  for k in range(1, totient):
    if (((exposant % totient) * (k % totient)) % totient == 1):
      return k

def Crypter(clef_pub, m):
  key, n = clef_pub
  m_Crypt = [(ord(char) ** key) % n for char in m]
  return m_Crypt

def Decrypter(clef_pri, c):
        key, n = clef_pri
        m_Dcrypt = [chr((char ** key) % n) for char in c]
        return ''.join(m_Dcrypt)
```

Annexe D

Algorithme - SHA 512 (Simplifié)

```
def m_bin(data_string):
  unicode_points = [ord(char) for char in data_string]
  binary_values = ['{0:08b}'.format(point) for point in unicode_points]
  binary_data = ''.join(binary_values)
  return binary_data
binary_data = m_bin('UQTR')

def preprocess_data(binary_data):
  data_length = '{0:0128b}'.format(len(binary_data))
  padding = '0' * (1024 - (len(binary_data) + 1 + 128) % 1024)
  binary_string = binary_data + '1' + padding + data_length
  hex_values = hex(int(binary_string, 2))
preprocessed_data = preprocess_data(binary_data)

def t1_t2(a, b, c, e, f, g, h, w, k):
  Ch_efg = (e & f) ^ (~e & g)
  Sigma_1 = (e >> 14) ^ (e >> 18) ^ (e >> 41)
  t_1 = h + Ch_efg + Sigma_1 + w + k
  Sigma_0 = (a >> 28) ^ (a >> 34) ^ (a >> 39)
  Maj_abc = (a & b) ^ (a & c) ^ (b & c)
  t_2 = Sigma_0 + Maj_abc
  a1 = t_1 + t_2
```

Annexe E

Expérience FFT vs QFT

```
import numpy as np
import qiskit
from qiskit import Aer, execute
from qiskit.circuit.library import QFT
import time
import matplotlib.pyplot as plt

def apply_fft(image):
    """Application de la FFT classique a une image 2D
    puis renvoie l'image transformee."""

    return np.fft.fft2(image)

def apply_qft(image):

    """Simulation de l'application de QFT a une image 2D a
    l'aide de Qiskit, renvoie le vecteur d'etat."""

    # Normaliser et preparer l'etat initial

    flattened_image = image.flatten() / np.linalg.norm(image)

    num_pixels = image.size  # Nombre total de pixels dans l'image
    # Nombre de qubits necessaires
    num_qubits = int(np.ceil(np.log2(num_pixels)))

    circuit = qiskit.QuantumCircuit(num_qubits)
    circuit.initialize(flattened_image, range(num_qubits))
    circuit.append(QFT(num_qubits), range(num_qubits))

    simulator = Aer.get_backend('statevector_simulator')
    result = execute(circuit, simulator).result()
    statevector = result.get_statevector(circuit)
    return statevector

image_sizes = [4, 8, 16, 32]  #(n * n)
```

```
fft_times, qft_times, errors, bit_usages, qubit_usages = [], [], [], [], []

for size in image_sizes:
    image = np.random.rand(size, size)

    start_time = time.time()
    fft_image = apply_fft(image)
    fft_times.append(time.time() - start_time)
    bit_usages.append(image.size * 2 * 32)

    start_time = time.time()
    qft_statevector = apply_qft(image)
    qft_times.append(time.time() - start_time)

        # Utilisation reelle des bits pour FFT
    qubit_usages.append(int(np.ceil(np.log2(size * size))))

fig, axs = plt.subplots(1, 3, figsize=(20, 5))

axs[0].plot(image_sizes, fft_times, label='Temps FFT', marker='o')
axs[0].plot(image_sizes, qft_times, label='Temps QFT', marker='s')
axs[0].set_xlabel("Taille de l'image")
axs[0].set_ylabel("Temps (secondes)")
axs[0].legend()

axs[1].bar(image_sizes, bit_usages, width=0.4,
label='Usage des bits', color='blue')

axs[1].set_xlabel("Taille de l'image")
axs[1].set_ylabel("Usage de ressources (bits)")
axs[1].legend()

axs[2].bar(image_sizes, qubit_usages, width=0.4,
label='Usage des qubits', color='orange')

axs[2].set_xlabel("Taille de l'image")
axs[2].set_ylabel("Usage de ressources (qubits)")
axs[2].legend()

plt.tight_layout()
plt.show()
```

Annexe F

Algorithme Estimation de phase quantique *QPE*

```
import math
from fractions import Fraction
from tabulate import tabulate

N = 15
a = 7
controll_qubits = 4
fraction_accuracy = 16

counts = {'0001': 72, '1001': 63, '0011': 69, '1100': 60, '0000': 74,
'1000': 57, '0101': 58, '1110': 68, '0110': 59, '1111': 60, '0111': 58,
'1101': 59, '0100': 73, '1011': 68, '0010': 60, '1010': 66}

factors = set()
target_period = 0
factor_found = False

# Continuous Fraction Part
rows, measured_phases = [], []
for output in counts:
    decimal = int(output, 2)  # convert binary numbers to decimal
    phase = decimal / (2 ** controll_qubits)  # find eigenvalues
    measured_phases.append(phase)

for phase in measured_phases:
    frac = Fraction(phase).limit_denominator(fraction_accuracy)
    rows.append([phase, "%i/%i" % (frac.numerator, frac.denominator),
    frac.denominator])
    guesses = [
        math.gcd(int((a ** int(frac.denominator/2))) + 1, N),
        math.gcd(int((a ** int(frac.denominator/2))) - 1, N)
    ]

    for guess in guesses:
```

```
        # Ignore trivial factors
        if guess != 1 and guess != N and N % guess == 0:
            factors.add(guess)
            target_period = int(frac.denominator)
            factor_found = True

# Print the final table
print(tabulate(rows, headers=["Phase", "Fraction", "Guess for r"],
colalign=('right','right','right')))

# Check if any factors were found
if factor_found:
    print("Factors found:", factors)
    print("Best period:", target_period)
else:
    print("No factors found.")
```

Annexe G

Algorithme pour tester les périodes r trouvé grâce au *QPE*

```
import math

# Define the exponent values
r = [16, 4, 1, 2, 8]

# Define the modulus
N = 15
a = 7

for i in r:
    # Calculate q and p using the given formulas
    q = math.gcd(int((a ** int(i//2))) + 1, N)
    p = math.gcd(int((a ** int(i//2))) - 1, N)

    # Print the results for each value of r
    print("For r =", i)
    print("q =", q)
    print("p =", p)
    print()
```

Annexe H

Algorithme quantique de Shor avec une implémentation de gestion de bruit quantique utilisant un ordinateur quantique de chez IBM

```
"""
@author: Nasr Akram
"""

# imports for Quantum part
from qiskit import QuantumCircuit, Aer, execute
from qiskit.circuit.library import QFT

#imports for Classical part
from fractions import Fractio
from sympy import sieve
import pandas as pd
import random
import math

#Classical acceleration
import concurrent.futures
import threading

#utils
from tabulate import tabulate

# Shared variable to indicate if a factor is found
factor_found = False

# Lock to synchronize access to the shared variable
factor_lock = threading.Lock()

N = ?
```

```
print(f"␣N␣=␣{N}")

controll_qubits = ?
target_qubits = ?

fraction_accuracy = ?

message = "␣"

# Authentication to IBM

IBMQ.save_account("YOUR_API")
IBMQ.load_account()

# Get the provider and backends
provider = IBMQ.get_provider('ibm-q')
backend = provider.get_backend("QUANTUM_COMPUTER_NAME")


def Crypter(clef_publique, mon_message):
  key, n = clef_publique
  msg_chiffre = [(ord(char) ** key) % n for char in mon_message]
  return msg_chiffre

msg_ssl = Crypter((65537, N), message)

def Modular_multiplicative_inverse(a, n):
  t = 0
  newt = 1
  r = n
  newr = a
  while newr != 0:
    quotient = r // newr
    t, newt = newt, t - quotient * newt
    r, newr = newr, r - quotient * newr
    if r > 1:
      return "a␣is␣not␣invertible"
    if t < 0:
      t = t + n
    return t

def value_a(N):
  while True:
    a = random.randrange(2, N-1)
    if math.gcd(a, N) == 1:
      return a

def initialize_qubits(qc, n, m):
  qc.h(range(n))  # apply hadamard gates
  qc.x(n)
```

```
def c_modN(a, k):
  U = QuantumCircuit(target_qubits)
  for _ in range(k):
    if a % 2 != 0:
      for q in range(target_qubits):
        U.x(q)
  U = U.to_gate()
  U.name = "%i^%i mod N" % (a, k)
  c_U = U.control()
  return c_U

def modular_exponentiation(qc, n, m, a):

  # Precompute c_modN values
  c_modN_values = [c_modN(a, k) for k in range(n)]

  for k, c_modN_value in enumerate(c_modN_values):
    if k >= 0:
      qc.barrier()  # Add a barrier between iterations if needed
      qc.append(c_modN_value, [k] + list(range(n, n + m)))

def qft_dagger(qc, measurement_qubits):
  qc.append(QFT(len(measurement_qubits),
                do_swaps=False).inverse(),
                measurement_qubits)
  qc.name = 'IQFT'

def measure(qc, n):
  qc.measure(n, n)

def error_correction(qc, n,m):
  qc.reset(range(n))
  qc.barrier()
  err_control_qubits = range(n)
  err_target_qubits = range(n,n+m)
  for i in err_control_qubits:

  # apply CNOT gates to correct bit flip errors
    for j in err_target_qubits:
      qc.cx(i, j)
    for j in err_target_qubits:
      qc.x(j)
  qc.barrier()

def period_finder(n, m, a):

    # set up quantum circuit
    qc = QuantumCircuit(n + m, n)
```

```
    # initialize the qubits
    initialize_qubits(qc, n, m)
    qc.barrier()

    # apply modular exponentiation
    modular_exponentiation(qc, n, m, a)
    qc.barrier()

    # apply error correction
    error_correction(qc, n, m)

    # apply inverse QFT
    qft_dagger(qc, range(n))
    qc.barrier()

    # measure the n measurement qubits
    measure(qc, range(n))
    return qc

list_r_val, measured_phases = [], [], []

a = value_a(N)
print("Finding the phases...")

qc = period_finder(controll_qubits, target_qubits, a)
counts = execute(qc, backend=backend).result().get_counts(qc)

print(f'Counts = {counts}')

for output in counts:
  decimal = int(output, 2)  # convert binary numbers to decimal
  phase = decimal / (2 ** controll_qubits)  # find eigenvalues
  measured_phases.append(phase)

for estimated_phase in measured_phases:
  frac = Fraction(estimated_phase).limit_denominator(fraction_accuracy)
  list_r_val.append(frac.denominator)

r_simplifier = list(set(list_r_val))

print(f'Estimated Phases = {r_simplifier}')

def p_q_finder(r_simplifier):
  global factor_found
  attempt = 0
  print("Calculating...")
  while not factor_found:
    try:
      attempt += 1
      a = value_a(N)
```

```
        factors = set()
        target_period = 0
        for r_val in r_simplifier:
          power_val = int(r_val/2)
          power_result = a ** power_val
          guesses = [
              math.gcd(power_result + 1, N),
              math.gcd(power_result - 1, N)
                  ]

        for guess in guesses:
          # Ignore trivial factors
          if guess != 1 and guess != N and N % guess == 0:
            factors.add(guess)
            target_period = r_val

        if len(factors)!=0:
          with factor_lock:
            if not factor_found:
              factor_found = True
              r = target_period
              P = factors.pop()
              Q = factors.pop() if len(factors) else N // P

              print("\nTentative %i:" % attempt)

              print('r = ', {r})

              phi = (P - 1) * (Q - 1)
              cle_p = Modular_multiplicative_inverse(65537, phi)

              print('d = ', cle_p)

          for future in futures:
            future.cancel()
          break

      except Exception as e:
        print(f"An error occurred: {str(e)}")
        break

# Number of instances to run in parallel
num_instances = ?

# Create a ThreadPoolExecutor with the desired number of workers
executor = concurrent.futures.ThreadPoolExecutor(max_workers=num_instances)

# Submit the function multiple times to the executor
futures = [executor.submit(p_q_finder(r_simplifier))
                           for _ in range(num_instances)]
```

```
# Wait for all the futures to complete
concurrent.futures.wait(futures)
```

Bibliographie

[1] Alan Willner et al. (Octobre, 2019) *Optical Fiber Telecommunications V11 (Chapter XII - 12.2.4.3 No-cloning theorem* .

[2] Alexander Streltsov (Novembre, 2014) *Quantum Entanglement* .

[3] Asher Peres (Octobre, 1993) *Quantum Theory: Concepts and Methods (Chapter VI - Bell's Theorem).*

[4] Azure Microsoft (S.D) *Présentation d'un qubit.*

[5] Asha Ambhaikar et al. (Mars, 2021) *AES AND RSA-BASED HYBRID ALGORITHMS FOR MESSAGE ENCRYPTION & DECRYPTION - (A Hybrid Encryption Scheme for AES and RSA).*

[6] Amente Bekele (2016) *Cooley-Tukey FFT Algorithms.*

[7] Bernard Pire (Mai, 2011) *QUANTUM DE PLANCK.*

[8] Benjamin Bahr et al. (Mars, 2016) *Quantum Tunneling.*

[9] Université Carleton (S.D) *Matrices symétriques et hermitiennes.*

[10] Chathranee Anumitha Jayathilake et al. (Janvier, 2013) *Discrete Walsh-Hadamard Transform in Signal Processing - Walsh Functions.*

[11] Daniel A. Lidar et al. (September, 2013) *Quantum Error Correction - Chapter I.*

[12] David J. Griffiths (1984) *The Collapse of the Wave Function - Chapter III (3.4. GENERALIZED STATISTICAL INTERPRETATION).*

[13] Daniel A. Lidar et al. (September, 2013) *Quantum Error Correction - Chapter II (2.1 Error correction).*

[14] Daniel A. Lidar et al. (Avril, 2013) *Quantum Error Correction - Chapter I (1.1.4 Noisy gate)* .

[15] Daniel J. Bernstein et al. (Octobre, 2006) *A general number field sieve implementation.*

[16] Gavin E.Crooks (Janvier, 2022) *Gates States and Circuits - 3.1 Pauli gates.*

[17] Gavin E.Crooks (Janvier, 2022) *Gates States and Circuits - 3.5 Hadamard gates.*

[18] Georgia Wood (Juin, 2022) *Encryption Security for a Post Quantum World.*

[19] Flevina Jonese D'souza et al. (Mai, 2017) *Advanced encryption standard (AES) security enhancement using hybrid approach* .

[20] JEREMY WOHLWEND (S.D) *ELLIPTIC CURVE CRYPTOGRAPHY: PRE AND POST QUANTUM.*

[21] Huyai Chen(2016) *ENDOMORPHISMES AUTO-ADJOINTS ET AUTOMORPHISMES ORTHOGONAUX.*

[22] Ivan Djordjevic (Avril, 2012) *Quantum Information Processing and Quantum Error Correction - Chapter III - 3.2 TWO-QUBIT OPERATIONS (CNOT SWAP)* .

[23] Ivan S.Oliveira et al. (Mars, 2007) *Fundamentals of Quantum Computation and Quantum Information - 3.4. QUANTUM LOGIC GATES* .

[24] IBM Quantum Composer (S.D) *Shor's algorithm.*

[25] IBM Qiskit (Décembre, 2022) *Shor's Algorithm* .

[26] IBM Qiskit (Décembre, 2022) *Quantum Phase Estimation*.

[27] Imtiaz Ahmad (Septembre, 2005) *Hardware implementation analysis of SHA-256 and SHA-512 algorithms on FPGAs*.

[28] John Walker (Mars, 2001) *DOES THE INERTIA OF A BODY DEPEND UPON ITS ENERGY-CONTENT? Albert einstein (September 1905)* .

[29] Javier Yanes (Mai, 2018) *Richard Feynman the Physicist Who Didn't Understand his Own Theories*.

[30] Kenichi Morita (Novembre, 2017) *Theory of Reversible Computing - Chapter IV Reversible Logic Gates)*.

[31] Liesbeth De Mol - Stanford Encyclopedia of Philosophy (Septembre 2018) *Turing Machines* .

[32] Lloyd Walden (Décembre, 2021) *What Is Superposition in Quantum Computing?*.

[33] Lo'ai Twalbeh. (Juin, 2007) *INCS 741 CRYPTOGRAPHY* .

[34] Mike Gianfagna (Juin, 2021) *Mathematics: How are Complex Numbers used in the Real World?*.

[35] Matt Peckham (Mai, 2012) *The Collapse of Moore's Law: Physicist Says It's Already Happening*.

[36] Nielsen et Chuang (Octobre, 2000) *Quantum Computation and Quantum Information - 5.1 The quantum Fourier transform* .

[37] Nielsen et Chuang (Octobre, 2000) *Quantum Computation and Quantum Information - 5.4.1 Period-finding*.

[38] PURPOSE FOCUS COMMITMENT (Novembre, 2018) *Wisdom story: The king the con artist the chessboard and rice*.

[39] Phillip Kaye et al. (Novembre, 2006) *An Introduction to Quantum Computing 7.1 Quantum Phase Estimation and the Quantum Fourier Transform (7.1.28)* .

[40] Qiskit (Mai, 2020) *2. Multiple Qubits and Entanglement - 2.4 More Circuit Identities*.

[41] Robert B. Griffiths (Janvier, 2014) *Hilbert Space Quantum Mechanics - Chapter I 1.1 Hilbert space*.

[42] Ravi Teja (Avril, 2021) *Implementation of Boolean Functions using Logic Gates*.

[43] Ronald V. Book (Janvier, 1973) *Comparing complexity classes*.

[44] Rui Zhang et al. (Février, 2022) *Loss-tolerant all-photonic quantum repeater with generalized Shor code* .

[45] robots(Novembre, 2000) *The Discrete Fourier Transform*.

[46] Spaceandmotion (Février, 2005) *Quantum Physics / Mechanics: Max Born*.

[47] Timothy Stapko (Mai, 2008) *Security Protocols and Algorithms - Chapter III* .

[48] Thomas N. Theis et al. (Avril, 2017) *The End of Moore's Law: A New Beginning for Information Technology*.

[49] Université de St Andrews (Avril, 2017) *Bloch sphere representation of quantum states for a spin $\frac{1}{2}$ particle*.

[50] Université de Cambridge (Septembre, 2020) *Periodic function*.

[51] Université de Californie (Août, 2018) *Fourier Series*.

[52] Université de Lyon (Mai, 2011) *Espaces de Hilbert - Chapter IV 4.1 Produits scalaires et notion d'espace de Hilbert* .

[53] Université Stanford (S.D) *Roots of Unity* .

[54] Université Laval (Janvier, 2001) *Quantification d'un moment angulaire* .