Hidden Structures and Quantum Cryptanalysis

Xavier Bonnetain

Sorbonne Université

École doctorale Informatique, Télécommunications et Électronique (Paris)

Inria de Paris / Équipe-projet SECRET

Hidden Structures and Quantum Cryptanalysis

Thèse de doctorat d'informatique

présentée par

Xavier Bonnetain

dirigée par María Naya-Plasencia

soutenue publiquement le 15 novembre 2019

devant un jury composé de :

María Naya-Plasencia	Inria	Directrice
Gilles Van Assche	STMicroelectronics, Belgique	Rapporteur
André Chailloux	Inria	Examinateur
Luca De Feo	IBM Research, Suisse	Examinateur
Henri Gilbert	ANSSI	Examinateur
Gregor Leander	Ruhr-Universität Bochum, Allemagne	Examinateur
Damien Vergnaud	Sorbonne Université	Examinateur
Yu Sasaki	NTT, Japon	Examinateur

rapportée par :

Bart Preneel	Katholieke Universiteit Leuven, Belgique
Gilles Van Assche	STMicroelectronics, Belgique

Illustration de couverture : Elizabeth Fearne Bonsall, frontispice de *The Fireside Sphinx*, 1901.

Version 28112019, 2nde édition.

À Paul, Maxence, Ghislain. Aux autres.

Remerciements

En premier lieu, je souhaiterais remercier María Naya-Plasencia, ma directrice de thèse, qui fin 2015 me proposa de travailler sur des attaques quantiques, alors que je savais à peine ce qu'était un qubit. Merci pour cette chance que tu m'as donnée, pour ce temps que tu n'avais pas et que tu m'as tout même consacré, pour m'avoir accompagné et guidé, pour ton indéfectible optimisme. Pour ces années, merci.

Je tiens à remercier Bart Preneel et Gilles Van Assche, qui ont accepté la lourde tâche de rapporter les pages qui vont suivre.

Je remercie du même coup Gilles Van Assche, André Chailloux, Henri Gilbert, Gregor Leander, Damien Vergnaud, Luca De Feo et Yu Sasaki pour avoir accepté de faire partie de mon jury.

De plus, Henri, avec Ludovic Perret, a assuré mon suivi doctoral, et je les en remercie.

Je remercie mon école doctorale et plus largement les services administratifs universitaires ainsi que le royaume des Pays-Bas, sans qui cette thèse aurait probablement été significativement plus morne.

Je souhaite remercier ceux grâce à qui j'ai pris le chemin de la thèse, et en particulier Hugues Randriam, dont les cours à Télécom m'ont redonné le goût des mathématiques, et Anne Canteaut, qui par une conférence à l'X m'a donné l'envie de faire de la cryptographie.

Ces années auraient été bien différentes sans l'ambiance extraordinaire de l'équipe SECRET (qui dit-on, rejoindra bientôt les étoiles), et je souhaite remercier, par ordre lexicographique et en espérant n'oublier personne : Adrien, Anais, André (l'Ancien), André (le Jeune), Andrea, Anirudh, Anne, Anthony, Antoine, Antonio, Augustin, Christelle, Christina, Daniel, Ferdinand, Florian, Gaëtan, Ghazal, Ivan, Jean-Pierre, Kaushik, Kévin, Léo, María, Mariem, Mathilde, Matthieu, Matthieu, Nicky, Nicolas, Nicolas, Pascale, Pierre, Rémi, Rocco, Rodolfo, Sébastien, Simon, Shizhu, Shouvik, Sristy, Thomas, Tim, Valentin, Virginie, Vivien et Yann.

La recherche, ça se ne fait pas seul, et je voudrais remercier ceux avec qui j'ai eu l'occasion de collaborer, bien évidemment María, mais aussi Akinori, André, Gaëtan, Jean-Francois, Léo, Rémi, Shizhu, Yixin et Yu.

Plus généralement, je souhaiterais remercier ceux qui m'ont montré que tout le monde pouvait être membre de la communauté des cryptographes, et je n'oublie pas ceux qui m'ont montré que n'importe qui pouvait en faire partie.

J'ai une pensée particulière pour les membres du bureau C201 : tout d'abord, à ceux qui m'ont précédé, Virginie (ton bureau est super), Sébastien (le tableau est quand même plus vide, depuis que tu n'es plus là) et Yann (dont on ne peut que regreter

l'immense verve), et qui m'ont accueilli chaleureusement. Puis à ceux qui ont suivi, Antonio, Daniel et Nicolas. Je vous laisse juge de mon accueil, et j'espère que vous porterez haut les couleurs de ce fier bureau[1].

Je ne t'oublie pas, Thomas, cher camarade. On a été en stage ensemble, puis en thèse ensemble, on a rempli des formulaires de mots de passe ensemble, on a donné cours ensemble. J'espère que tout ira bien en perfide Albion, et que tu n'oublieras pas l'essentiel : L'important, c'est les valeurs.

Léo, ce fut un plaisir, que ce soit pour affronter les audaces culinaires des marchands d'opium ou les audaces statistiques de la Loubianka. J'espère qu'on aura l'occasion de remettre ça, et que tu ne finiras pas par faire de la plongée pour baliser le passage du Nord-Est.

Merci Jean-Pierre, futur grand sachem[2] qui se comprend très bien, pour toutes ces discussions, et pour le babyfoot[3]. Je sais que ta porte restera toujours ouverte.

Enfin, au sein de l'équipe, je voudrais remercier André (le Jeune), qui ~~m'a emmerdé durant tout son stage avec ses foutues question~~ s'est tout de suite montré très curieux, et se révéla être un collègue d'une redoutable efficacité, qui sût se dépasser, et sans aucun angle mort. Cette thèse, finalement, elle est aussi un peu à toi, et j'espère vraiment qu'on continuera à collaborer après mon départ.

Merci, pêle-mèle, à la clique faërixienne, à #doctorat, aux rustacées parisiennes, qui m'ont accompagnés durant ces années.

Je remercie ma famille, qui m'a toujours soutenu et accompagné, malgré ma fatigue, malgré les deadlines qui tombent mal, malgré les emplois du temps compliqués.

Finalement, je te remercie, toi qui prends la peine de lire ces mots, et j'espère que tu trouveras quelque intérêt dans la prose qui suit.

[1] Je vous souhaite notamment d'en maintenir le ratio de prix de thèse.

[2] Ou peut-être *grand groumpf* ?

[3] N'oublions pas que 0–8, ça se remonte.

Contents

Présentation des travaux

Les travaux de cette thèse portent sur la cryptanalyse quantique, c'est à dire les attaques de systèmes cryptographiques utilisant des moyens quantiques. Après des préliminaires de cryptographie au Chapitre 1 et de calcul quantique au Chapitre 2, cette thèse s'organise en deux parties : des algorithmes quantiques sont présentés dans la Partie I, et de nombreuses applications sont présentées dans la Partie II.

Chapitre 3. Ce chapitre présente les algorithmes quantiques de recherche non structurée (l'algorithme de Grover) et de recherche de collisions (l'algorithme de Brassard, Høyer et Tapp). Ma contribution dans ce chapitre est la proposition d'un framework algorithmique permettant de décrire de façon unifiée les recherches imbriquées tant classiques que quantiques, afin d'aider la création et la description d'attaques quantiques, qui est appliqué dans le Chapitre 11. Ce framework fait partie de l'article « Quantum security analysis of AES », qui est un travail commun avec María Naya-Plasencia et André Schrottenloher, et a été accepté au journal ToSC en 2019 [BNS19b].

Chapitre 4. Ce chapitre présente l'algorithme quantique de Simon, qui résout le problème suivant :

Problème 1 (Problème de Simon)**.** *Soit n un entier, $s \in \{0,1\}^n$ et X un ensemble. Soit $f : \{0,1\}^n \to X$ une fonction, avec la promesse que pour tout $(x,y) \in (\{0,1\}^n)^2$, $[f(x) = f(y) \Leftrightarrow x \oplus y \in \{0,s\}]$. Étant donné un accès en boite noire à f, trouver s.*

L'algorithme de Simon résout ce problème en temps polynomial, et est le premier exemple de problème pour lequel l'ordinateur quantique est exponentiellement plus rapide que l'ordinateur classique, sous réserve d'avoir accès à un oracle quantique implémentant f, ce qui est le modèle de *requêtes quantiques*, ou *requêtes en superposition*. Cet algorithme est à la base de nombreuses cryptanalyses, présentées dans les Chapitres 7, 8 et 9.

Chapitre 5. Ce chapitre est lui consacré à un problème plus général, et souvent plus difficile :

Problème 2 (Décalage caché abélien)**.** *Soit n un entier, $\mathcal{G}$ un groupe abelien, X un ensemble, $f, g : \mathcal{G} \to X$ deux fonctions injectives et $s \in \mathcal{G}$, avec la promesse que pour tout x, $f(x) = g(s + x)$. Étant donné un accès en boite noire à f et g, trouver s.*

Dans le cas général, les algorithmes pour résoudre ce problème sont sous-exponentiels. Trois algorithmes sous-exponentiels ont été proposés dans la littérature, deux par Kuperberg, et un par Regev. Mes contributions à ce chapitre sont multiples. Tout d'abord, j'ai étudié le premier algorithme de Kuperberg, je l'ai optimisé et je l'ai simulé, ce qui m'a permis d'en estimer le coût à $5 \times 2^{1.8\sqrt{n}}$ si $\mathcal{G}$ est de taille environ 2^n. J'ai aussi étudié le comportement de ces algorithmes lorsque la promesse du problème est relachée, ce qui est souvent nécessaire pour les applications en cryptanalyse, et pour le groupe $(\mathbb{Z}/(2^w))^p$, pour lequel le problème est plutôt plus facile. Cela fait partie de l'article « Hidden Shift Quantum Cryptanalysis and Applications », coécrit avec María Naya-Plasencia et accepté à Asiacrypt 2018 [BN18].

Ensuite, j'ai étudié la variante de Regev, et j'ai pu montrer que ses exposants connus dans la littérature n'étaient pas optimaux, et que celui-ci dépend directement du coût d'algorithmes de subset-sum. De plus, j'ai pu présenter de nombreux compromis entre temps classique, temps quantique, mémoire classique et nombre de requêtes à f et g, ce qui rend cet algorithme extrèmement versatile, et adaptable selon les ressources disponibles. Les différents compromis de l'algorithme de Regev sont décrits dans le manuscrit « Improved Low-qubit Hidden Shift Algorithms », qui n'est pas encore publié [Bon19b]. Des compromis à faible coût quantique, temps classique important et mémoire classique polynomiale sont utilisés dans « Trade-off between classical and quantum circuit size of the attack against CSIDH », article coécrit avec Jean-François Biasse, Benjamin Pring, André Schrottenloher et William Youmans et accepté au Journal of Mathematical Cryptology en 2019 [Bia+19].

Enfin, j'ai étudié le second algorithme de Kuperberg, qui est le plus efficace en temps, mais nécessite une mémoire classique exponentielle. J'ai pu proposer de nouveaux compromis entre temps classique, temps quantique, mémoire classique et nombre de requêtes, qui permettent de proposer des compromis plus intéressant que le précédent algorithme quand la mémoire classique peut être importante. Ces nouveaux compromis sont présentés dans « Quantum Security Analysis of CSIDH and Ordinary Isogeny-based Schemes », coécrit avec André Schrottenloher et actuellement en soumission [BS18].

Ces algorithmes sont utilisés pour différentes cryptanalyses dans les Chapitres 7, 9 et 10.

Chapitre 6. Ce chapitre présente une méthode générale pour appliquer les algorithmes de Simon et de Kuperberg y compris lorsque l'on a seulement un accès en boite noire classique à la fonction vérifiant $f(x) = f(x \oplus s)$ (ou aux fonctions vérifiant $f(x) = g(x + s)$). Le coût des algorithmes devient alors exponentiel ($2^{n/3}$ si la taille du domaine de f est de n bits), avec une faible mémoire. Si son coût est plus important, l'algorithme reste meilleur que les autres approches pour résoudre le même problème avec un accès classique à la fonction. De plus, cet algorithme est le premier exemple d'utilisation de l'algorithme de Simon dans lequel on ne fait pas directement l'hypothèse d'avoir un accès quantique à une fonction périodique, ce qui était un problème ouvert en informatique quantique. Cet algorithme provient de l'article « Quantum Attacks without Superposition Queries : the Offline Simon's Algorithm », coécrit avec Akinori Hosoyamada, María Naya-Plasencia, Yu Sasaki et André Schrottenloher, et ac-

cepté à Asiacrypt 2019 [Bon+19]. Il est utilisé pour différentes cryptanalyses dans les Chapitres 7 et 9.

Chapitre 7. Ce chapitre recense les différentes attaques basées sur une période ou un décalage caché en cryptographie symétrique, depuis leur introduction par Kuwakado et Morii en 2010. J'ai pu étendre les attaques basées sur une période cachée aux constructions équivalentes, mais utilisant des additions au lieu de ou exclusifs, et étudier leur efficacité dans [BN18]. L'estimation du coût de l'attaque permet de déduire les tailles de paramètres requises pour obtenir une construction sûre. Dans ce cas, les tailles nécessaires sont largement supérieures aux tailles des designs de constructions symétriques qui ont été proposés à l'heure actuelle. J'ai aussi pu étendre bon nombre de ces attaques au cas où l'on n'a qu'un accès classique à la construction cryptographique, avec la méthode du Chapitre 6 dans [Bon+19].

Chapitre 8. Ce chapitre présente les différentes cryptanalyses du chiffrement authentifié AEZ depuis sa version 4. AEZv4 a été analysé classiquement par Chaigneau et Gilbert, qui ont montré qu'une recherche de collisions permettait d'en recouvrer les clés. Néanmoins, le coût de cette attaque est supérieur aux allégations de sécurité d'AEZ. De plus, j'ai montré avec Patrick Derbez, Sébastien Duval, Jérémy Jean, Gaëtan Leurent, Brice Minaud et Valentin Suder qu'une erreur de conception permettait trivialement de briser l'authentification d'AEZv4, ce qui a mené à AEZv5. Enfin, j'ai pu réécrire l'analyse en collision de Chaigneau et Gilbert sous la forme d'une période cachée, et la généraliser à AEZv5 et sa variante renforcée AEZ10, qui sont donc vulnérables à une attaque quantique polynomiale utilisant des requêtes en superposition, et à une attaque classique en collision. Le contenu de ce chapitre correspond à l'article « Quantum Key-Recovery on Full AEZ », que j'ai présenté à SAC 2017 [Bon17].

Chapitre 9. Ce chapitre présente les *slide attacks* quantiques, qui sont une variante des *slide attacks* classiques. Ces attaques quantiques se répartissent en deux catégories : celles qui peuvent s'écrire sous la forme d'un décalage caché, et sont drastiquement améliorées, et les autres, pour lesquelles le gain est bien plus maigre. Tout comme leur parent classique, les *slide attacks* quantiques utilisent des propriétés d'auto-similarité de constructions itérées pour les attaquer indépendamment de leur nombre de tours, ce qui les rend particulièrement dévastatrices. Les *slide attacks* classiques n'étant pas particulièrement récentes, la plupart des constructions modernes s'en prémunissent, ce qui les protège aussi des *slide attacks* quantiques. De façon surprenante, ce n'est pas le cas du chiffrement MiMC-2n/n et de certaines versions de sa généralisation GMiMC, pour lesquels j'ai pu montrer qu'une *slide attack* quantique était applicable, qui était aussi applicable avec des requêtes classiques au chiffrement, et qui correspondait à une attaque classique en collision permettant de contredire les allégations de sécurité de MiMC-2n/n et des versions concernées de GMiMC. Le contenu de ce chapitre correspond à l'article « On Quantum Slide Attacks », coécrit avec María Naya-Plasencia et André Schrottenloher, et présenté à SAC 2019 [BNS19a], ainsi qu'à la note « Collisions on Feistel-MiMC and univariate GMiMC », non publiée à ce jour [Bon19a].

Chapitre 10. Ce chapitre s'éloigne de la cryptographie symétrique, et présente des schémas d'échanges de clé à base d'isogénies. Ceux-ci ont la particularité de correspondre à un décalage caché, et donc d'être vulnérables à une attaque quantique sous-exponentielle. J'ai en particulier étudié la sécurité asymptotique et concrète de l'instance la plus prometteuse, CSIDH, et montré que les jeux de paramètres proposés n'offrent pas la sécurité annoncée. L'étude asymptotique provient de [Bia+19], et l'étude concrète de [BS18].

Chapitre 11. Ce chapitre étudie la sécurité quantique du chiffrement par bloc le plus utilisé de nos jours, AES. Celui-ci ne dispose pas de structure permettant d'appliquer un algorithme de période ou de décalage caché, et les attaques proposées sur des versions réduites d'AES se basent sur des adaptations des cryptanalyses classiques d'AES, en utilisant le framework de recherche structurée du Chapitre 3. Les attaques quantiques proposées sont plus efficaces que la recherche quantique, là où les attaques classiques de la littérature ne doivent battre que la recherche classique. Cela a pour conséquence que ces attaques quantiques concernent des versions d'AES avec un tour de moins que les meilleures attaques classiques, et suggère donc que la marge de sécurité quantique d'AES, qui estime le nombre de tours restant avant une attaque sur la version complête, est plus importante que la marge de sécurité classique. Le contenu de ce chapitre correspond à l'article « Quantum Security Analysis of AES », coécrit avec María Naya-Plasencia et André Schrottenloher, et accepté à ToSC 2019 [BNS19b].

Autres travaux. Enfin, j'ai eu la chance de travailler avec Léo Perrin et Shizhu Tian sur l'étude de propriétés théoriques des Boîte-S, avec une application intéressante à la boite-S de la fonction de hachage Streebog et du chiffrement par bloc Kuznyechik, ce qui a mené à l'article « Anomalies and Vector Space Search : Tools for S-Box Analysis », accepté à Asiacrypt 2019 [BPT19]. Ce travail étant très éloigné des autres, il n'est pas détaillé dans ce manuscrit.

Main publications

[Bon+19] Xavier Bonnetain, Akinori Hosoyamada, María Naya-Plasencia, Yu Sasaki, and André Schrottenloher. "Quantum Attacks without Superposition Queries: the Offline Simon's Algorithm". In: *ASIACRYPT 2019*. Ed. by Steven Galbraith and Shiho Moriai. LNCS. Springer, Heidelberg, Dec. 2019 (cit. on pp. 9, 87, 90, 99).

[BPT19] Xavier Bonnetain, Léo Perrin, and Shizhu Tian. "Anomalies and Vector Space Search: Tools for S-Box Analysis". In: *ASIACRYPT 2019*. Ed. by Steven Galbraith and Shiho Moriai. LNCS. Springer, Heidelberg, Dec. 2019 (cit. on p. 10).

[BNS19a] Xavier Bonnetain, María Naya-Plasencia, and André Schrottenloher. "Quantum Security Analysis of AES". In: *IACR Trans. Symm. Cryptol.* 2019.2 (2019), pp. 55–93. ISSN: 2519-173X (cit. on pp. 7, 10, 35, 40, 165).

[BNS19b] Xavier Bonnetain, María Naya-Plasencia, and André Schrottenloher. "On Quantum Slide Attacks". In: *SAC 2019*. Ed. by Kenneth G. Paterson and Douglas Stebila: LNCS. Springer, Heidelberg, Aug. 2019 (cit. on pp. 9, 99, 129).

[Bia+19] Jean-François Biasse, Xavier Bonnetain, Benjamin Pring, André Schrottenloher, and William Youmans. "Trade-off between classical and quantum circuit size of the attack against CSIDH". In: *J. Mathematical Cryptology* (2019) (cit. on pp. 8, 10, 153).

[BN18] Xavier Bonnetain and María Naya-Plasencia. "Hidden Shift Quantum Cryptanalysis and Implications". In: *ASIACRYPT 2018, Part I*. Ed. by Thomas Peyrin and Steven Galbraith. Vol. 11272. LNCS. Springer, Heidelberg, Dec. 2018, pp. 560–592 (cit. on pp. 8, 9, 57, 77–79, 81, 82, 99, 110, 112).

[Bon17] Xavier Bonnetain. "Quantum Key-Recovery on Full AEZ". In: *SAC 2017*. Ed. by Carlisle Adams and Jan Camenisch. Vol. 10719. LNCS. Springer, Heidelberg, Aug. 2017, pp. 394–406 (cit. on pp. 9, 99, 117, 124).

Preprints

[Bon19a] Xavier Bonnetain. *Improved Low-qubit Hidden Shift Algorithms*. 2019. arXiv: 1901.11428 (cit. on pp. 8, 57, 70, 72).

[Bon19b] Xavier Bonnetain. *Collisions on Feistel-MiMC and univariate GMiMC*. 2019 (cit. on pp. 9, 129).

[BS18] Xavier Bonnetain and André Schrottenloher. *Quantum Security Analysis of CSIDH and Ordinary Isogeny-based Schemes*. 2018 (cit. on pp. 8, 10, 57, 77, 80, 153, 159).

Chapter 1
Introduction to Cryptography

Cryptography is the science that studies the protection of information. Protection has multiple meanings. The most natural notion is to prevent an illegitimate person (or adversary) to have access to a message, which is *confidentiality*. The other main classical notions of security are *integrity*: ensuring that nobody has tampered with an encrypted message and *authenticity*: ensuring that the emitter of the message is the expected one.

Contents

1.1 History

Protecting communications is not a recent need, and encryption techniques have been used for a long time, in particular to secure military communications or to protect craft recipes from competitors. The ancient Greeks are known to have used the scytale, which is a wooden stick whose dimensions are only known from the sender and the receiver of the message. The message was written on a strip of parchment rolled around the stick. An attacker who would like to read the message would then have to guess the stick's dimensions. Another notable early encryption method was Caesar's cipher. Julius Caesar is said to have encrypted his correspondence by shifting each one of the 26 letters of the alphabet by a fixed value [Sue21]. There are 25 possible secret values, or *keys*, to encrypt the message. Many other encryption methods that rely on the same principle, but with more complex operations and larger keys have later been

proposed. We can cite for example the Vigenere cipher [Vig86], proposed in 1586, in which multiple shifts are performed, which makes that a given letter can be encrypted by different letters. More generally, this is an example of a *substitution cipher*, where a symbol in a message (or *plaintext*) is replaced by another in the encrypted message (or *ciphertext*), depending on the key. Today's *block ciphers* are similar, with an alphabet which is the set of n-bit strings (in practice, n is often 128) and a complex relation between the plaintext, the ciphertext and the key.

The use of these ciphers had led to the development of methods to break them, that is, to obtain the message without the prior knowledge of the key. This is *cryptanalysis*. Its first recorded occurrence is the description of *frequency analysis* to attack the Caesar cipher by Al-Kindi in the IXth century [Kin09]. This method relies on the fact that each letter is always encrypted by the same letter. Hence, the most common letter in the ciphertext corresponds to the most common letter in the plaintext. For example, in English, the most common letter is **e**, and it is likely to correspond to the most common letter in the ciphertext. A generalization of this approach to break the Vigenere cipher has been proposed by Kasiski in 1863 [Kas63].

Mechanization introduced in modern era has reshaped the landscape of cryptography, and the main principles and ideas of today's cryptography date from this period.

Kerckhoffs's principle. Kerckhoffs stated in 1863 [Ker83] what would become the first design principle of a cryptographic system: its security shall not rely on the secrecy of its design, and it shall not be an issue if it falls into enemies hands. This principle is the opposite of the concept of security through obscurity. The rationale in the XIXth century was that the capture of a soldier shall not threaten the cryptographic system as a whole, and the security of the military communications shall be restored after a change of an easy-to-update key. Nowadays, this approach allows third-party cryptanalysis, that is, the analysis of a design by the cryptographic community beyond the designers. This is at the core of the trust we have in the security of a design: we only trust a design after a deep and continuous public scrutiny has not exhibited any weakness. Such trust is by definition temporary: new ideas arise and the attacks are improved with time. The scenario we try to avoid at all cost is the one where an attack exists, has been found, is used in practice, but is not publicly known.

One-time-pad. The one-time-pad, or Vernam's cipher, has been proposed in 1882 by Miller [Mil82] and was rediscovered in a simpler version and patented by Vernam in 1919 [Ver19]. In Vernam's form, it takes an n-bit plaintext and an n-bit key, and outputs the XOR of them. A given key shall only be used once. Another way to see it is like Caesar's cipher limited to one letter of a huge alphabet. This encryption method is not very practical, as it requires the knowledge of a shared secret key as long as the message. Nevertheless, as was proven by Shannon in 1949 [Sha49] this cipher is unconditionally (or perfectly, or information-theoretically) secure, that is, given the knowledge of the ciphertext, it is impossible to recover the plaintext without some information on the secret key. Today's *stream ciphers* can be seen as a more practical version of the one-time-pad, where a long keystream of arbitrary length is derived from

a shorter, fixed-length key. These ciphers are not unconditionally secure, but some are used in practice, for example in mobile phone protocols.

Computational security. In the same article, Shannon proposed the weaker notion of *computational security*, that is, for a given cipher, it shall be hard enough to recover the secret. This is the security notion used by almost all of today's cryptographic constructions. This leads to the notion of *generic attacks*: some attacks can be applied regardless of the design of a construction, and their cost only depends on some general parameters, such as the number of different keys. In order to estimate how hard it is to break a cipher, we must model what an attacker can do (this is the *attacker model*) and how much an action costs (this is the *cost model*). These models allow to compare different analyses and for the designers to make refutable claims on the security of their construction. These claims often correspond to the cost of the best known generic attack (that is, the construction is as secure as we can hope it to be), but this is not always the case.

Public-key cryptography. The encryption methods previously presented required the knowledge of a shared secret. In 1976, Diffie and Hellman [DH76] have proposed a protocol which allows for two parties to obtain a shared secret key only by exchanging some public messages, and without any prior common knowledge of a secret. The security of this protocol relies on the hardness of computing discrete logarithms. This is the inception of public-key cryptography (or asymmetric cryptography), which, contrary to symmetric cryptography, does not need the prior knowledge of a secret. This adds more constraints, which makes the public-key cryptosystems less efficient. In practice, hybrid approaches are used, with public-key cryptography establishing a shared secret key to be used to encrypt the messages with a symmetric algorithm.

RSA. RSA, from the name of its authors Rivest, Shamir and Adleman [RSA78] is the first and one of the most widely used signature algorithms, which allows to check integrity and authenticity. Each party can sign a message with his *private key*, and anyone can check using the corresponding *public key* if a message has been emitted by the private key holder. The security of RSA relies in practice on the hardness of factoring.

Quantum-safe cryptography. Shor's algorithm [Sho94] is a quantum algorithm that happens to solve in polynomial time the problems of factoring and computing discrete logarithms. This would effectively break almost all the public key cryptosystems used in practice nowadays, once a quantum computer is built. This existential threat on most protocols has led to a quick response of the cryptographic community, with a call for candidates for new standards of key exchange and signatures that resist a quantum computer issued by the NIST only 22 years later [NIST16].

1.2 Constructions in symmetric cryptography

This section presents the general designs in symmetric cryptography which will be studied in Part II. Symmetric encryption constructions contain two families: block ciphers, which are permutations on a fixed-size state (the block size), and stream ciphers, which generate a keystream, a pseudo-random stream of arbitrary length which can be xored to the message. Block ciphers can encrypt messages of fixed size, and need to be combined with a mode of operation to be able to encrypt messages of variable length.

Hash functions are generally considered to be symmetric primitives. They are functions that map an arbitrary-length message to a value (or hash) of fixed size. They do not contain any secret material, and can be used to check integrity. A key-dependent hash function is called a Message Authentication Code, or MAC. It can be used to check authenticity and integrity.

More recently, some designs with a more integrated approach have been proposed. This is the case of authenticated encryption ciphers, which combine the functionality of a symmetric cipher and a MAC, or the versatile sponge construction, which can be used to make an encryption function, a hash function or a MAC.

1.2.1 Block ciphers

Definition 1.1. A block cipher is a family of permutations on n bits parametrized by its key K on κ bits: $\{E_K : \{0,1\}^n \to \{0,1\}^n | K \in \{0,1\}^\kappa\}$.

In practice, we cannot consider a set of 2^κ random permutations, and the block cipher is structured. It is often an iterated construction, whose form is described in Figure 1.1. It consists of r iterations of a round function F_k. Each round uses a different round key k_i which is derived from the master key K with a key schedule algorithm. The existence of this key schedule is critical for the security of the construction. This is developed in Chapter 9. As a message is not always a single block of n bits, a block cipher has to be combined with a mode of operation, which defines how to handle messages of arbitrary length.

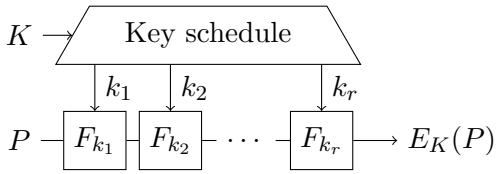


Figure 1.1: Iterated block cipher with r rounds

There are generally two approaches to construct the round function, which are described below.

1.2.1.1 Feistel Network

The first proposed construction for a round function was the Feistel network, introduced by Horst Feistel for the early block cipher Lucifer. This construction has become popular after its use in DES [DES], which was a 64-bit block, 56-bit key block cipher and was the first encryption standard in the United States. In a Feistel network, the state is split in two halves (or branches) and the round function operates as in Figure 1.2. The same principle can be used with more branches, which gives the generalized Feistel construction. The Feistel construction allows to construct a permutation which is easy to invert (given the knowledge of the key), even if f_{k_i} is not a permutation. A common special case is the Key-alternating Feistel, where the round function if of the form $f_{k_i}(x) = f(x \oplus k_i)$. Some results on the generic security of the Feistel and generalized Feistel constructions in a classical and a quantum setting are presented in Chapter 7, and many attacks on Feistel ciphers with a weak key schedule are presented in Chapter 9.

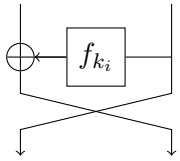


Figure 1.2: Feistel round function.

1.2.1.2 Substitution-permutation network (SPN)

The other main approach for block cipher designs is to use substitution-permutation networks. Its round function is described in Figure 1.3. It alternates between a substitution layer that consists in the parallel application of a small non-linear function, or S-Box, which has to make the mathematical relation between the bits of the plaintext and the bits of the ciphertext more complex, a permutation or linear layer which has to end up creating a dependency between each bit of the plaintext and each bit of the ciphertext, and a key addition, which prevents the inversion of the function. In practice, the substitution and linear layers are invertible even without the knowledge of the key, and they are omitted if they are not between to round keys. Hence, the first and last operation of an SPN is always the key addition. The most notable example of SPN is AES [AES], which became the current standard block cipher after DES had become obsolete due to its too short key size. The resistance of AES against quantum attacks is studied in Chapter 11.

1.2.2 Hash functions & MACs

Hash functions are mappings from messages of arbitrary length to pseudo-random messages of fixed length, and can be used to check integrity. Contrary to block ciphers, they do not have a key parameter.

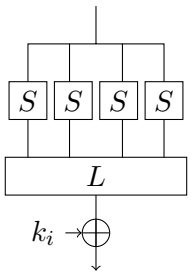


Figure 1.3: SPN round function with a layer of 4 S-Boxes.

MACs, for message authentication code, are constructions which allow to compute a fixed-length value (the tag) from an arbitrary-length message and a secret key. Any other holder of the secret key can then check if the value associated to the message is correct. They are meant to ensure authenticity, and can be seen as a symmetric equivalent of a signature, with the main difference that any person able to verify the message is also able to produce a valid tag.

A standard way to make MACs is the Wegman-Carter-Shoup construction [WC81; Sho96] which authenticates a message m associated with a nonce n and using the keys k_1, k_2:

$$\mathrm{MAC}(m, n) = H_{k_1}(m) + E_{k_2}(n)$$

with H a family of universal hash functions, and E a block cipher. This construction is used by GMAC [GCM07] and Poly1305 [Ber05], which are the two MACs available in TLS [Res18] to authenticate the communications on the web. The quantum security of Poly1305 is discussed in Section 7.7.3.

1.2.3 Authenticated encryption

To ensure the confidentiality and authenticity of a communication, one can combine a block cipher with a mode and a MAC. Another approach is to define a scheme which does everything at once. This is what does an authenticated encryption scheme. Integrating everything in one construction may allow for a lighter design or for a better-understood security than with a combination of independent constructions.

The CAESAR competition, which began in 2014, aimed at selecting a portfolio of such algorithms. Fifty-six candidates were submitted, including AEZ [HKR15], which is analyzed in Chapter 8. Five years later, the competition has finally ended, and two candidates were selected for each of the three uses cases: Ascon [Dob+19] and ACORN [Wu16] for constrained environment, AEGIS-128 [WP16] and OCB [KR16] for high-performance applications, and Deoxys-II [Jea+16] and COLM [And+16] for defense in depth.

In 2018, the NIST launched a competition to standardize some lightweight authenticated encryption scheme. Fifty-six candidates were also submitted, and the competition is still ongoing.

1.2.4 Sponges

Sponges [Ber+07] are a way of constructing symmetric primitives from a permutation. They are typically used to make hash functions, the most prominent example is [SHA3], or authenticated ciphers. The construction is described in Figure 1.4. It has two parts: in the absorption part, an arbitrary-length input is processed by the sponge, and symmetrically, in the squeezing part, an arbitrary-length output is produced. The speed depends on r, which is the *rate*, or outer part of the sponge, as one iteration of P absorbs or squeezes r bits. The security depends on c, the *capacity*, or inner part of the sponge. The quantum security of some sponge-based constructions is discussed in Section 7.8.

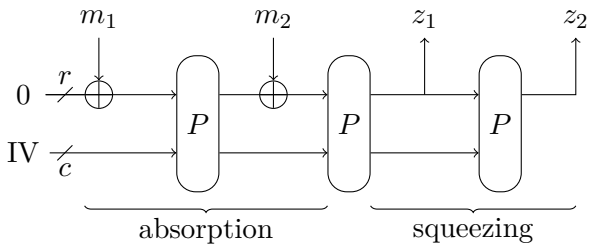


Figure 1.4: Sponge construction

1.3 Cryptanalysis

Cryptanalysis aims at estimating the security of a design, that is, how much does it cost to attack it. An attack can mean different things. For ciphers, it often refers to key recovery, and most of the attacks presented in this manuscript are key recovery attacks. Weaker attacks are also possible, such as the recovery of a partial information of a plaintext from the ciphertext. For hash functions, we generally consider collisions (finding two inputs with the same output) and preimage (finding one input corresponding to a fixed output). For MACs, we can consider key recovery or forgeries, that is, the ability to produce a valid tag for a message whose tag has not been queried. The message can be chosen beforehand (this is a universal forgery), or constrained by the attack (this is an existential forgery). The reference points for the attack cost are the generic attacks, which can always be applied. To compare different attacks, we need an attack model, which defines what an attacker can do, and a cost model, which will define the resources required.

1.3.1 Generic attacks

For almost all cryptographic systems, there are *always* applicable attacks. Such attacks depend on the type of primitive, but not on the concrete instance. In general, we want a design to be as secure as possible, and if an attack that beats the best generic attack is found, then the construction is considered broken, and no further analysis on

it is required. Even non-practical attacks may qualify as a break: indeed, they show a weakness of the design, which may lead to a more practical attack.

Exhaustive search on block ciphers. The attack generally considered for block ciphers is the exhaustive search of the key of size κ, at a cost of 2^κ encryptions. There are other less naive ways of performing the exhaustive search, as for example bicliques, that have a cost marginally below the naive way.

Security margin. While no attack beating the best generic attack has been found, an important measure is the security margin, that is, for an iterated construction, the number of rounds that separate the full cipher from the longest round-reduced version known to be broken. This margin allows to estimate how likely is that the full construction is broken in the future, as improvements of existing attacks are expected to be incremental.

Quantum generic attacks. Quantum search, developed in Chapter 3, allows to reduce the cost of an exhaustive search by a square root, that is, a key of size κ only offers $\kappa/2$ bits of quantum security. Hence, the baseline for classical and quantum security is not the same, and a relevant quantum attack has to use less resource, which also means that the security margin is different. Moreover, the relative cost of the different attacks is not the same quantumly. In particular, collision search, also presented in Chapter 3, is less improved than exhaustive search.

Quantum security margin. In a long-term future, we can presume that the expected security of a primitive will be given by its best generic attack (i.e. quantum search), and that the security margin of this primitive will be determined by the highest number of rounds cryptanalyzed with any attack more efficient than this exhaustive search. Therefore, the logical evolution is that the classical or quantum epithet for attacks would become irrelevant, and the most efficient attacks, possibly using quantum tools, will be the most important information regarding how far a primitive is from being broken.

1.3.2 Attack models

The attack model defines what we consider the attacker can do. In particular, it defines how the attacker can access (or query) a primitive (or oracle) parametrized with an unknown key, when applicable. We list below a few standard attack models.

Known plaintext. In this model, the attacker can obtain a list of couples of plaintexts and ciphertexts, but cannot choose them.

Chosen plaintext/ciphertext. In this model, the attacker can choose the plaintext to be encrypted, or the ciphertext to be decrypted. The attack can be non-adaptative,

if all the queries can be chosen before the attack, or adaptative, if some queries depend on an intermediate computation of the attack.

Related-key. This model assumes that the attacker is allowed to query the primitive with a key that depends on the secret key (for example, to query the cipher $E_{k\oplus\delta}$, with δ a chosen value).

Classical computations (Q0). Up to now, the most frequent models consider that the attacker performs classical computations only. With quantum computers, two other models can also be considered.

Classical queries (Q1). The first quantum model is the most restrictive: it allows for local quantum computation, but considers that the queries the attacker can do are classical. This model is the closest to classical ones, and is often considered more realistic, as it models a threat against *today's* communications: indeed, one may obtain today some queries, and wait until a quantum computer is available to break them.

Quantum queries (Q2). The second model considers that the adversary can query a keyed function in quantum superposition, that is, instead of having an oracle that produces $f_k(x)$ given x, she has access to an oracle that produces $\sum_x |x\rangle |f_k(x)\rangle$ given $\sum_x |x\rangle |0\rangle$ for a secret key k. This is strictly more powerful, as for example, Simon's problem is exponentially easier with a quantum oracle than with a classical oracle.

Even if this last model might appear too strong, it has multiple advantages, and has been studied theoretically in [Gag17]. First, it is a simple model, which encompasses any intermediate model of access to primitives. In particular, it does not force to distinguish between queries to a secret function and queries to a public function, which can help in some security proofs. Moreover, this model is applicable in some white-box scenarios, where an attacker has access to an obfuscated program that implements the secret function. Such program can be reimplemented on a quantum computer, which gives access to the quantum oracle. Next, a primitive secure in this model can be used for more applications, for example to be used in a quantum protocol. As the scope of primitives secure in the Q1 model is more restricted, considering the Q2 model also limits the issues when the users do not respect the cryptographic recommendations. Finally, as shown in Chapter 6, there are links between the quantum attacks with quantum queries and quantum attacks with classical queries.

It is to be noted that the model of quantum queries can be degenerate when associated with related keys, as shown in Section 7.4 and Section 9.2.1.

1.3.3 Cost models

A cost model is a metric for the cost of an attack. It corresponds to the computational cost of the attack and includes the number of queries. In general, we consider a time cost which consists in the number of operations in the attack, a memory cost, which correspond to the minimal amount of memory required to run the attack, and a data

cost, which corresponds to the number of queries required. With parallelism, some other cost can be considered, such as the number of processors, and the time cost may take into account some communication cost between the processors. As the natural cost unit of exhaustive search is a given number of encryptions, the cost of one encryption is generally taken as a unit for the time cost.

With quantum attacks, the time, memory and data can be either classical or quantum. We separate the two costs in the description of the attacks. In some cases, the time cost mainly consists in doing queries, and in that case, we consider that the time cost is the query cost.

Chapter 2
Quantum Computing

This chapter presents the basic notions of quantum computing required for the following chapters. It first presents the main differences with classical computing, then introduces the quantum circuit model, and finally presents some general techniques we will use in the next chapters.The reader interested in a more comprehensive introduction may read the Nielsen and Chuang [NC10].

Contents

2.1 History

Quantum physics provides a model explaining very well a lot of previously unexplained physics experiments, but left physicists with many difficulties: quantum behaviour generally arise at a very small scale, and quantum physics inherently restrict the amount of information we can obtain from a quantum system. These constraints have been seen as a problem to overcome, but also as a useful tool. For example, Wiesner proposed in the 70s to use the no-cloning property (see Theorem 2.1) of quantum states to create unforgeable materials, which may be used for banknotes [Wie83]. Principles very similar to this *quantum money* were used for *quantum key distribution* [BB84], an analogous to a classical key exchange that uses quantum communications and relies on physical hypothesis instead of computational ones. This wave of new applications came with some novel ideas to study quantum systems: some of them cannot be studied directly in a laboratory, and are too complex to be simulated efficiently with classical computers. However, we may create a simpler to analyze quantum system, in a controlled environment, and make it evolve according to the same rules as the original system. This is the idea of a *quantum simulator*, formalized by Feynman in 1982 [Fey82]. A *quantum*

computer is the same idea, but instead of computing the evolution of a quantum system, it is used to perform any computation.

2.2 Differences with classical computations

Classically, we can see a computation as a sequence of instructions that computes intermediate values, until a final result is produced. In practice, these intermediate values are encoded in some ways, being a position in a abacus, a voltage in a circuit or an atom spin in a hard drive. Quantum physics states that there is more than fixed values: a quantum state can be in a *superposition* of values.

Quantum computing uses this additional degree of freedom to perform some operations a classical computer cannot do, and gain in efficiency. However, this added freedom disappears when we measure the state (that is, check a position, measure a voltage,...). A measure produces a classical value, and makes the quantum state collapse to the measured value. This imposes the constraint for the quantum operations that make our quantum state evolve to be reversible.

Moreover, we cannot make a copy of an unknown quantum state. If we want to do so, we need to have access to the quantum circuit that produced the quantum state, and reuse it. The reversibility constraint poses also a problem if we want to erase some information (for example, erase some intermediate values): we need to reapply the quantum circuit that has produced these values, in reverse, to make them disappear. This is called *uncomputing*.

2.3 Qubits

Qubits (for *quantum bits*) are the quantum equivalent of the classical bits.

Definition 2.1 (Finite-dimensional Hilbert Space)**.** A finite-dimensional Hilbert space $\mathcal{H}$ is a vector space over $\mathbb{C}$ equipped with an inner product $\langle \cdot | \cdot \rangle : \mathcal{H} \times \mathcal{H} \to \mathbb{C}$, that satisfies:

- Linearity: $\langle x | \sum_i \alpha_i y_i \rangle = \sum_i \alpha_i \langle x | y_i \rangle$
- Conjugate symmetry: $\langle x | y \rangle = \overline{\langle y | x \rangle}$
- definite-positivity: $x \neq 0 \implies \langle x | x \rangle > 0$

In practice, in quantum computing, we consider the vector space $\mathbb{C}^{2^n}$ with the canonical inner product $\langle x | y \rangle = \sum_{i=0}^{2^n-1} \overline{x_i} y_i$.

Definition 2.2 (Norm)**.** We define the norm $|\cdot|$ of a vector as $|x| = \sqrt{\langle x | x \rangle}$.

Definition 2.3 (Dirac notation)**.** A vector $x = (x_1, \ldots, x_n)$ is noted $|x\rangle = \sum_i x_i |i\rangle$. This notations comes from the inner product $\langle x | y \rangle$, which can be seen as a linear form $\langle x |$ applied to a vector $|y\rangle$. As $\langle | \rangle$ is a "bracket", $\langle |$ is a "bra" and $| \rangle$ is a "ket".

Definition 2.4 (Qubit)**.** A qubit is an element of the Hilbert space $\mathbb{C}^2$, that we note $|\phi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$. α_i is the *amplitude* of $|i\rangle$ in $|\phi\rangle$.

Definition 2.5 (Bases)**.** The set $(|0\rangle, |1\rangle)$ is a basis of $\mathbb{C}^2$,and is called the *canonical* basis, or the *computational basis*. There are other bases, such as the *Hadamard* basis $\left(|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right)$.

Property 2.1 (Normalization)**.** *Qubits are invariant by multiplication by a global constant. We can define a canonical form* $|x\rangle = \alpha_0 |0\rangle + e^{i\theta}\alpha_1 |1\rangle$*, with* $\alpha_0, \alpha_1 \in [0;1]^2, \theta \in [0; 2\pi)$ *and* $||x\rangle| = \alpha_0^2 + \alpha_1^2 = 1$.

Remark 2.1 (Qubit notations)*.* In practice, the normalization factor can often be omitted to lighten the notation.

This normalization is justified by the following property, which links qubits with the classical world.

Property 2.2 (Measure)**.** *A qubit can be measured, which produce a classical bit as an outcome. The probability of measuring 0 is* α_0^2*, and a 1 is measured with probability* α_1^2*. This measurement effectively destroys the qubit: if a 0 is measured, it is transformed in* $|0\rangle$*, and conversely an outcome of 1 transforms the qubit into* $|1\rangle$*.*

Intuitively, a measurement projects the qubit, and transform it into what we have measured. This implies that even if the amplitude can be continuous (and encode any amount of information), a given qubit can only give one bit of information before being destroyed. The whole idea of quantum computing is to ensure that these bits of information are meaningful.

The properties of one qubit generalize to states with multiple qubits. The resulting space is the tensor product of the Hilbert spaces.

Definition 2.6 (Qubits)**.** A state of n qubits is an element of the Hilbert space $\mathbb{C}^{2^n}$, denoted $|\phi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle$.

As before, we can normalize the qubits state with $\sum_i |\alpha_i|^2 = 1$.

Remark 2.2 (Tensor products)*.* If $n = n_1 + n_2$, $\mathbb{C}^{2^n} \simeq \mathbb{C}^{2^{n_1}} \otimes \mathbb{C}^{2^{n_2}}$. In some cases, this tensor product can be reflected on the vectors. For example, the state $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|10\rangle$ is $\left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right) \otimes |0\rangle$, that we note $\left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right)|0\rangle$, or $|+\rangle|0\rangle$. This tensor product reflects an independence of the two qubits, and we say that the state is *separable*, or *unentangled*. Conversely a state that is not separable is *entangled*.

Property 2.3 (Partial measure)**.** *Let* $|\phi\rangle = \alpha_0 |\phi_0\rangle|0\rangle + \alpha_1 |\phi_1\rangle|1\rangle$*. A measure of the last qubit of* $|\phi\rangle$ *will have outcome 0 with probability* $|\alpha_0|^2$*, and project* $|\phi\rangle$ *into* $|\phi_0\rangle|0\rangle$*. Otherwise, with probability* $|\alpha_1|^2$*, the outcome will be 1 and* $|\phi\rangle$ *will be projected into* $|\phi_1\rangle|1\rangle$*. In general, a partial measurement on an entangled state will project thee non-measured part to the subset of values compatible with the outcome of the measure.*

Example 2.1 (Qubits and measure)**.**

- A measure of $|0010\rangle$ has outcome 0010 with probability 1 and does not change the qubit.
- A measure of $\frac{1}{2}|00\rangle + \frac{1}{2}|10\rangle + \frac{1}{\sqrt{2}}|11\rangle$ has outcome 10 with probability $\frac{1}{4}$. A measure of the first qubit has outcome 0 and projects to $|00\rangle$ with probability $\frac{1}{4}$, and has outcome 1 and projects to $|1\rangle\left(\frac{1}{\sqrt{3}}|0\rangle + \sqrt{\frac{2}{3}}|1\rangle\right)$ with probability $\frac{3}{4}$.

2.4 Quantum gates

Now that we have defined our quantum states, we need some tools to modify them. This is the role of *quantum gates*, which are small operators that transform a quantum state. For more complex transformations, we use the term *quantum operator*.

There are many ways to represent these transformations: we can cite mathematical descriptions with matrices (or, equivalently, linear functions), more algorithmic representations with circuits or pseudocode or a categorical representation with a graph in the ZX-calculus. We present here the matrices, linear functions and circuits for the quantum gates, and will use the circuit, linear function and pseudocode representations of quantum operators in the next chapters.

From the previous section, it may seem peculiar that we consider complex amplitude, as the complex phase has no impact on the probability of a measurement outcome, and does not change the actual information we can get from a qubit. This becomes useful here, with some transformations for which the phase matters.

Definition 2.7 (Unitary operators)**.** Unitary operators are invertible linear functions that preserve the inner product.

Definition 2.8 (Quantum operator)**.** A quantum operator on n qubits is a unitary operator in $\mathbb{C}^{2^n}$.

Definition 2.9 (Inverse operator)**.** Any quantum operator O has an inverse, noted $O^\dagger$. Applying an inverse operator after the operator is called *uncomputing*. Contrary to classical computations, we cannot drop some intermediate values of a computations. Uncomputing allows us to erase these intermediate values by reverting the transformation, at the same cost as O.

Remark 2.3 (Matrix Notation). As qubits are unitary vectors, quantum operators are unitary matrices.

Remark 2.4 (Linear function Notation). One of the way to describe a linear function is to use the *ketbra* notation $f = \sum_i |x_i\rangle\langle y_i|$. This can be read as $f|y_i\rangle = |x_i\rangle$.

Theorem 2.1 (No cloning [WZ82])**.** *There is no quantum operator that, given an arbitrary state $|\psi\rangle$ and an auxiliary state $|a\rangle$ as input, can produce $\alpha(\psi)|\psi\rangle|\psi\rangle$, with $\alpha(\psi)$ an arbitrary function.*

Proof. Consider a U such that $U(|\psi\rangle|a\rangle) = \alpha(\psi)|\psi\rangle|\psi\rangle$. As U is unitary, $|\alpha(\psi)| = 1$. Let's consider $|\phi\rangle$ and $|\psi\rangle$ two orthogonal states, and a linear combination $a|\phi\rangle + b|\psi\rangle$. We note $\beta = \alpha(a\psi + b\phi)$.

$$\begin{aligned}
& U(a|\psi\rangle + b|\phi\rangle)|0\rangle = aU|\psi\rangle|0\rangle + bU|\phi\rangle|0\rangle \\
\Leftrightarrow\ & \beta(a|\psi\rangle + b|\phi\rangle)(a|\psi\rangle + b|\phi\rangle) = a\alpha(\psi)|\psi\rangle|\psi\rangle + b\alpha(\phi)|\phi\rangle|\phi\rangle \\
\Leftrightarrow\ & \beta\left(a^2|\psi\rangle|\psi\rangle + b^2|\phi\rangle|\phi\rangle + ab(|\psi\rangle|\phi\rangle + |\phi\rangle|\psi\rangle)\right) \\
& \qquad = a\alpha(\psi)|\psi\rangle|\psi\rangle + b\alpha(\phi)|\phi\rangle|\phi\rangle \\
\Leftrightarrow\ & \begin{cases} \beta a^2 = a\alpha(\psi) \\ \beta b^2 = b\alpha(\phi) \\ \beta ab = 0 \end{cases} \\
\Rightarrow\ & |a| = 0 \text{ or } |b| = 0
\end{aligned}$$

Hence, if we can clone two orthogonal states, then we cannot clone their linear combinations. □

Remark 2.5 (Cloning Classical Information). The classical values $|i\rangle$ are orthogonal to each other, hence it is possible to clone them. This can be done with the CNOT gate (see Section 2.4.1.7). Moreover, an alternative procedure would be to measure (which does not modify these states) and then to construct the same state.

Remark 2.6 (Making Clones). We can make copies of a quantum state if we know the quantum operator that has made it: in that case, we can reapply it to obtain a copy. However, the cost of the copy is at least the cost of creation of the original state. It can be much higher if the process involves measurements, as we would need to obtain the same outcomes.

2.4.1 Notable gates

This section presents the main gates used in the next chapters.

2.4.1.1 Identity Gate

The identity gate is noted I, and leaves any qubit invariant.

2.4.1.2 NOT Gate

The not gate is the equivalent of a classical not: it maps $|0\rangle$ to $|1\rangle$, and conversely, and is noted X or NOT.

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \Big| \quad |b\rangle \text{ ——} \boxed{X} \text{—— } |b \oplus 1\rangle$$

Figure 2.1: NOT gate

2.4.1.3 Z gate

The Z gate leaves $|0\rangle$ invariant, and negates the phase of $|1\rangle$.

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \quad \Big| \quad |b\rangle \text{ —— } \boxed{Z} \text{ —— } (-1)^b |b\rangle$$

Figure 2.2: Z gate

2.4.1.4 Y gate

The Y gate does both a negation and a phase shift.

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad \Big| \quad |b\rangle \text{ —— } \boxed{Y} \text{ —— } i(-1)^b |b \oplus 1\rangle$$

Figure 2.3: Y gate

Definition 2.10 (Pauli gates)**.** The gates X, Y, Z are called the Pauli gates.

2.4.1.5 Phase shift gate

The phase shift gate is a generalisation of the Z gate: it leaves $|0\rangle$ invariant, and rotates the phase of $|1\rangle$ by a given angle θ.

Definition 2.11 (T gate)**.** The gate $R_{\frac{\pi}{4}}$ is called the T gate.

$$R_\theta = \begin{pmatrix} 1 & 0 \\ 0 & e^{2i\pi\theta} \end{pmatrix} \quad \Big| \quad |b\rangle \text{ —— } \boxed{R_\theta} \text{ —— } \exp(2i\pi\theta b) |b\rangle$$

Figure 2.4: R_θ gate

2.4.1.6 Hadamard Gate

The Hadamard Gate is a very useful gate that has no classical equivalent. It maps a qubit in the computational basis to a phase difference. We can also see it as a change of basis operator, between the computational basis and the Hadamard basis. Hence, we have $H = |+\rangle\langle 0| + |-\rangle\langle 1|$

Hadamard gates are often useful to initialize a superposed state. Applying the gate on each qubit of the n-qubit register $|0\rangle$ produces the uniform superposition $\frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle$.

Remark 2.7. The Hadamard gate is involutory: applying it twice leaves a qubit invariant.

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad \Big| \quad |b\rangle \text{ —— } \boxed{H} \text{ —— } \frac{1}{\sqrt{2}}\left(|0\rangle + (-1)^b |1\rangle\right)$$

Figure 2.5: Hadamard gate

2.4.1.7 CNOT gate

The CNOT (for controlled NOT) gate is the quantum analogue of the XOR. It maps $|a\rangle|b\rangle$ to $|a\rangle|a \oplus b\rangle$.

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} I & 0 \\ 0 & X \end{pmatrix} \quad \Big| \quad \begin{array}{l} |a\rangle \text{ ——●—— } |a\rangle \\ |b\rangle \text{ ——⊕—— } |a \oplus b\rangle \end{array}$$

Figure 2.6: CNOT gate

Definition 2.12 (Clifford gates)**.** The Clifford gates are the gates obtainables by any combination of $H, R_{\frac{\pi}{2}}$ and CNOT.

Clifford gates are the "simple" gates, that is, we know how to efficiently implement them with the current quantum computing architecture. Moreover, a circuit composed with only Clifford gates can be simulated easily. This correspond to a small subset of all quantum circuits.

2.4.1.8 Toffoli gate

The Toffoli gate [Tof80] (or CCNOT, for controlled controlled NOT) is the quantum analogue of the logical and.

$$\text{Tof} = \begin{pmatrix} I & 0 \\ 0 & \text{CNOT} \end{pmatrix} \quad \Big| \quad \begin{array}{l} |a\rangle \text{ ——●—— } |a\rangle \\ |b\rangle \text{ ——●—— } |b\rangle \\ |c\rangle \text{ ——⊕—— } |c \oplus (a \wedge b)\rangle \end{array}$$

Figure 2.7: Toffoli gate

2.4.1.9 Quantum Fourier Transform

The Quantum Fourier Transform (QFT) is another gate without a classical equivalent, and is at the core of Shor's algorithm [Sho94]. We note $\omega = e^{\frac{2i\pi}{N}}$ an Nth-root of the unity. It maps a qubit $|x\rangle$ to the superposition $\sum_{\ell=0}^{N-1} \omega^{x\ell} |\ell\rangle$.

$$QFT_N = \frac{1}{\sqrt{N}} \begin{pmatrix} 1 & 1 & \cdots & 1 & \cdots & 1 \\ 1 & \omega & \cdots & \omega^{i} & \cdots & \omega^{N-1} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 1 & \omega^{j} & \cdots & \omega^{ij} & \cdots & \omega^{j(N-1)} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \cdots & \omega^{i(N-1)} & \cdots & \omega^{(N-1)^2} \end{pmatrix}$$

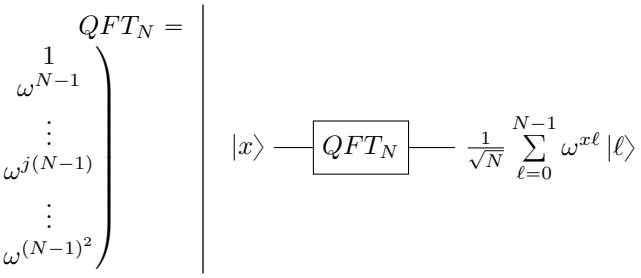


Figure 2.8: Quantum Fourier Transform

Inverse Quantum Fourier Transform. The inverse operator is the same, except that x is replaced by $-x$ or, equivalently, that ω is replaced by ω^{-1}.

Implementations. We know how to make the Quantum Fourier Transform modulo 2^n exactly. Kitaev proposed a way to compute approximately the Quantum Fourier Transform modulo N [Kit96], and an exact version has been proposed by Mosca and Zalka [MZ04].

2.4.2 Tensor product of quantum operators

If we have two quantum registers $|x\rangle|y\rangle$, note $O_1 \otimes O_2$ the operator that applies O_1 to $|x\rangle$ and O_2 to $|y\rangle$.

We denote the application of O on n consecutive registers $O^{\otimes n}$.

2.4.3 Computing classical functions

Any classical function can be computed in a quantum way. The general principle to compute a function f in a quantum way is to use NOT, CNOT and Toffoli gates to implement a reversible equivalent of f, that will output a tuple $(x, f(x))$ given the input $(x, 0)$. In general, we implement $|x\rangle|y\rangle \mapsto |x\rangle|y \oplus f(x)\rangle$. An efficient systematic approach to do so has been proposed by Bennett [Ben89].

Truncated functions. As noted in [HS18], we can compute in superposition the truncated version of a classical function f from the circuit that computes f.

Let's consider a boolean function b, implemented in a quantum circuit as $|x\rangle|y\rangle \mapsto |x\rangle|y \oplus b(x)\rangle$. The state $|x\rangle|+\rangle$ is not affected by the application of b, as $|+\rangle = |0\rangle + |1\rangle = |0 \oplus b(x)\rangle + |1 \oplus b(x)\rangle$.

This principle can be applied to functions with larger outputs: any qubit corresponding to a bit to be kept should be initialized to $|0\rangle$, and qubits corresponding to a bit to be dropped to $|+\rangle$.

Part I
Hide and Seek

Chapters

Chapter 3
Quantum Search

This chapter first presents Grover's algorithm [Gro96], a generic quantum search algorithm and its generalization, amplitude amplification [Bra+02]. These algorithms are then used to introduce the algorithmic framework for nested search we developed in [BNS19b], which is used in Chapter 11. Finally, classical and quantum algorithms for collision search are discussed.

Contents

3.1 Unstructured search

This section presents some algorithms for *unstructured search*, defined as:

Problem 3.1 (Unstructured search)**.** *Let $f : X \to \{0,1\}$ be a test function. Given oracle access to f, find $x \in X$ such that $f(x) = 1$.*

We note Y the set we're interested in:

Definition 3.1 (Good elements)**.** We note $Y = \{x \in X | f(x) = 1\}$ the set of *good elements*, that is, the elements that satisfy f.

3.1.1 Classical resolution

Classically, the best we can do is exhaustive search: testing some values x until we find a good element. We need, on average, $|X|/|Y|$ trials to obtain one random element in Y.

If we note $c_{\text{search}}(X, P)$ the cost of finding a value in X that fulfills P, $c_{\text{sample}}(X)$ the cost of producing an element in X, $c_{\text{test}}(P)$ the cost of checking if an x fulfills P and $c_{\text{trials}}(X, P)$ the expected number of trials we have to perform in order to find a good x, then we have

$$\underbrace{c_{\text{search}}(X,P)}_{\text{Total cost}} = \underbrace{c_{\text{trials}}(X,P)}_{\text{Number of } x \text{ to test}} \left(\underbrace{c_{\text{sample}}(X)}_{\text{Cost to produce an } x} + \underbrace{c_{\text{test}}(P)}_{\text{Cost to test } x} \right).$$

Moreover, we have $c_{\text{trials}}(X, P) = |X|/|Y|$.

3.1.2 Grover's algorithm

Grover's algorithm allows to solve Problem 3.1 at roughly the square root of the classical cost. It uses 3 operators: Hadamard gates, an oracle $O_f : |x\rangle \mapsto (-1)^{f(x)} |x\rangle$ and the inversion around 0: $O_0 : |x\rangle \mapsto (-1)^{x \neq 0} |x\rangle$.

Initial state. The initial state of the algorithm is $H^{\otimes n} |0\rangle = \sum_{i=0}^{2^n-1} |i\rangle = |\psi_0\rangle$.

Grover's operator. From H and O_0, we can compute $H^{\otimes n} O_0 H^{\otimes n}$. This operator is called the *inversion around average*. Let's consider a state $|x\rangle = \sum_{i<2^n} x_i |i\rangle$. If we note $A = \frac{1}{2^n} \sum_i x_i$ the average amplitude, $H^{\otimes n} O_0 H^{\otimes n} |x\rangle = \sum_i (2A - x_i) |i\rangle$. This operator changes each amplitude (x_i) into the value opposite of the average $(2A - x_i)$.

Another way to see this operator is that $O_0 = 2 |0\rangle\langle 0| - I$. Hence, $HO_0H = 2H |0\rangle\langle 0| H - HIH = 2 |\psi_0\rangle\langle\psi_0| - I$. This means this operator is a reflection around the state $|\psi_0\rangle$. As $|\psi_0\rangle$ is the uniform superposition of all elements, this is another way to state the inversion around average.

Similarly, if we note $|\text{Good}\rangle = \sum_{f(x)=1} |x\rangle$ the state containing the values we want and $|\text{Bad}\rangle = \sum_{f(x)=0} |x\rangle$ the state containing the superposition of the other values, $O_f = 2 |\text{Bad}\rangle\langle\text{Bad}| - I$ is a reflection around the state $|\text{Bad}\rangle$.

Grover's operator is the composition of HO_0HO_f. As proven in Lemma 3.1, this operator is a small rotation, which allows to turn away from the superposition of all elements and get closer to the superposition of good elements. This is described in Figure 3.1.

Iteration. Grover's algorithm simply consists in the iterations of Grover's operator (HO_0HO_f) from the initial state $(H^{\otimes n} |0\rangle)$. We note $|\psi_k\rangle$ the state after k iterations. When $|\psi_k\rangle$ is close enough to $|\text{Good}\rangle$, which occurs if $k \simeq \sqrt{\frac{|X|}{|Y|}}$, a measurement of $|\psi_k\rangle$ will have an outcome in Y with a good probability.

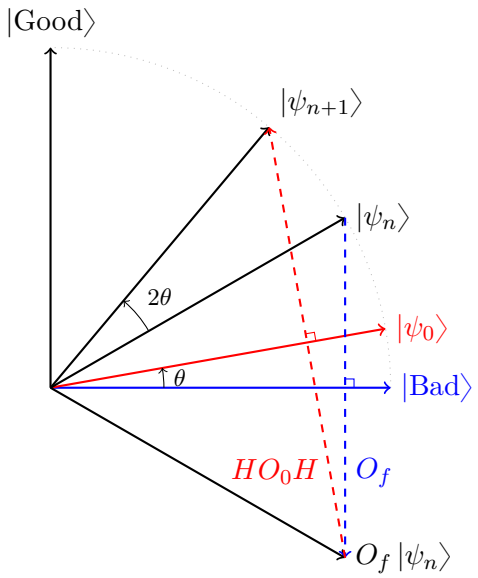


Figure 3.1: One iteration of the Grover operator HO_0HO_f, which transforms $|\psi_n\rangle$ in $|\psi_{n+1}\rangle$. O_f and HO_0H are both reflections, hence their combination is a rotation.

Theorem 3.1 (Grover's complexity)**.** *Let* $Y = \{x | f(x) = 1\}$. *A measurement of the state after* $t = \left\lfloor \pi / \left(4 \arcsin\left(\sqrt{\frac{|Y|}{|X|}}\right)\right)\right\rfloor$ *iterations has an outcome* $x \in Y$ *with a probability greater than* $1 - \frac{|Y|}{|X|}$.

Remark 3.1. In practice, we generally have $|X| \gg |Y|$, hence we take $t = \left\lfloor \frac{\pi}{4}\sqrt{\frac{|X|}{|Y|}}\right\rfloor$.

Lemma 3.1 (Grover's rotation)**.** *The sequence* HO_0HO_f *is a rotation of angle* $2\theta = 2\arcsin\left(\sqrt{\frac{|Y|}{|X|}}\right)$ *in the plane spanned by* $|Good\rangle$ *and* $|Bad\rangle$

Proof.

$$|\psi_0\rangle = \sqrt{\frac{|Y|}{|X|}}\,|\text{Good}\rangle + \sqrt{\frac{|X|-|Y|}{|X|}}\,|\text{Bad}\rangle$$

Hence, both reflections are in the plane spanned by $|\text{Good}\rangle$ and $|\text{Bad}\rangle$.

The angle θ between $|\psi_0\rangle$ and $|\text{Bad}\rangle$ satisfies $\cos(\theta) = \langle\psi_0|\text{Bad}\rangle = \sqrt{\frac{|X|-|Y|}{|X|}}$, hence $\sin(\theta) = \sqrt{\frac{|Y|}{|X|}}$. As the two reflection axes have an angular difference of θ, the composition of the two reflections has angle 2θ. □

Proof of Theorem 3.1. The initial state $\frac{1}{\sqrt{|X|}}\sum_{x\in X}|x\rangle$ is the state $|\text{Bad}\rangle$ after a rotation of angle θ in the plane $(|\text{Good}\rangle, |\text{Bad}\rangle)$. Hence, after t iterations, the state is

$$\sin((2t+1)\theta)\,|\text{Good}\rangle + \cos((2t+1)\theta)\,|\text{Bad}\rangle\,.$$

Hence, the probability of measuring a value not in Y is $\cos((2t+1)\theta)^2$. We have $\frac{\pi}{4}-\theta \leqslant t\theta < \frac{\pi}{4}$. Hence, $\frac{\pi}{2}-\theta \leqslant (2t+1)\theta < \frac{\pi}{2}+\theta$. Hence, $\cos((2t+1)\theta)^2 \leqslant \cos(\frac{\pi}{2}-\theta)^2 = \sin(\theta)^2 = \frac{|Y|}{|X|}$. □

Remark 3.2 (Soufflé property). Any additional iteration will increase the rotation angle and make the state turn *back* from the superposition of good elements to the uniform superposition.

Lower bound. This algorithm allows to perform an exhaustive search in roughly the square root of the classical time. One could ask if we can do better. Bennett, Bernstein, Brassard and Vazirani [Ben+97] proved that the number of queries needed to be in $\Omega\left(\sqrt{N}\right)$, with a search space of size N and a unique solution. This has been later improved by Boyer, Brassard, Høyer and Tapp [Boy+98] with a proof of optimality up to a factor 2. Finally, Zalka settled the problem and showed that Grover's algorithm is exactly optimal [Zal99].

3.1.3 Amplitude amplification

Grover's algorithm assumes that the search space is $\{0,1\}^n$. Amplitude amplification [Bra+02] generalizes it to any search space, and only assumes that we are given a test function f and a quantum circuit C such that $C|0\rangle$ is a superposition of values in the search space. From this circuit and f, the amplitude amplification procedure will produce a state very close to the part of the superposition in $C|0\rangle$ that satisfies f.

In Grover's algorithm, the Hadamard gates $H^{\otimes n}$ fulfill this role, as

$$H^{\otimes n}|0\rangle = \sum_{i=0}^{2^n-1} |i\rangle .$$

Amplitude amplification simply replaces the two application of $H^{\otimes n}$ in Grover's algorithm by the application of C and $C^\dagger$. As in Grover's algorithm, we can split the state $C|0\rangle$ in $\sqrt{a}\,|\text{Good}\rangle + \sqrt{1-a}\,|\text{Bad}\rangle$. The behaviour of the algorithm is then *exactly* the same: begin with $C|0\rangle$, and then iterate $C^\dagger O_0 C O_f$.

Theorem 3.2 (Amplitude amplification [Bra+02])**.** *Let S be a set, $f : S \to \{0,1\}$ be a test function, C be a quantum circuit such that $C|0\rangle = \sqrt{\alpha}\,|Good\rangle + \sqrt{1-\alpha}\,|Bad\rangle$, with f equal to 1 when restricted to the values in $|Good\rangle$ and 0 when restricted to the values in $|Bad\rangle$.*

After $t = \lfloor \pi/(4\arcsin(\sqrt{\alpha})) \rfloor$ iterations of $C^\dagger O_0 C O_f$ from $C|0\rangle$, a measure of the state produces a value x such that $f(x) = 1$ with probability greater than $1-\alpha$.

Remark 3.3 (Explicit cost). If we consider that f is implemented with a circuit T_f that needs to be uncomputed, then the cost of amplitude amplification is

$$\underbrace{\left\lfloor \pi/\left(4\arcsin\left(\sqrt{\alpha}\right)\right)\right\rfloor}_{\text{Number of iterations}} \left(\underbrace{2\text{cost}(C)}_{\text{Sampling cost}} + \underbrace{2\,\text{cost}(T_f)}_{\text{Test cost}} \right) + \underbrace{\text{cost}(C)}_{\text{Initialization cost}}.$$

Classical and quantum sampling. Classically, if the fraction of good values is p, it will cost $1/p$ iterations of the test to obtain one random good value. Quantumly, the cost will be roughly $1/\sqrt{p}$, but the output of the procedure is not one random good value, but the superposition of all good values (and if we measure it, we obtain one random good value).

3.1.4 Approximate test functions

The hypothesis of Grover's algorithm is that we have access to a function f, implemented as a quantum circuit, which will output deterministically 0 or 1 depending on the input. However, in practice, a test may be probabilistic (and depend on some randomness, for example). As long as there is no dependency in some external value, this case is encompassed by amplitude amplification: the search space can be enlarged with all the additional values required for the test. The set of amplified values will contain the correct values minus the false negatives and plus the false positives, in the same relative proportion as before the amplification, and we can apply Theorem 3.2. We can see it as having an always-trivial test function (that tests if a given register is 0 or 1), with all the complexity coming from the search space (which also contains the intermediate values we need to obtain the value in the test register).

We however cannot do this if our test function depends on some external quantum state that we cannot afford to compute at each call of f. This case will occur in Chapter 6. As developed in it, we will have to bound the degradation of the computation induced by the imperfect implementation of f.

Definition 3.2 (Approximate test circuit)**.** If f is a test function, T_f is an approximate test circuit of f if given as input $|x\rangle|\psi\rangle|0\rangle$, T_f computes $(-1)^{f(x)}|x\rangle|\phi\rangle + |\delta\rangle$, with $|\delta\rangle$ an arbitrary vector.

Theorem 3.3 (Amplitude amplification with approximate test)**.** *Let S, f, C and α be defined as in Theorem 3.2, let T_f be an approximate test circuit of f with noise $|\delta\rangle$ and ϵ be such that $||\delta|| < \epsilon$. Then after $t = \lfloor \pi/(4\arcsin(\sqrt{\alpha})) \rfloor$ iterations, a measure of the state produces a value x such that $f(x) = 1$ with probability greater than $(1-\alpha)(1-t\epsilon)^2$.*

Proof. In the amplitude amplification procedure, a call to the inversion around f is replaced by a call to T_f. Hence, each call adds a noise vector $|\delta\rangle$ to the state, and after k iterations, as the operators are linear, the total noise is of amplitude at most $k\epsilon$. Without any noise, after k iterations we would be in the state $|\psi_k\rangle$, but due to it, we are in the state $|\psi_k\rangle + |\psi_{err}\rangle$. We have

$$|\langle\psi_k|\psi_{err}\rangle| \leqslant ||\psi_k\rangle| \times ||\psi_{err}\rangle| \leqslant k\epsilon.$$

Hence, measuring $|\psi_k\rangle + |\psi_{err}\rangle$, the probability of obtaining a value in $|\psi_k\rangle$ is greater than

$$(1 - |\langle\psi_k|\psi_{err}\rangle|)^2 \geqslant (1 - k\epsilon)^2.$$

From Theorem 3.2, we have that the probability of measuring a value such that $f(x) = 1$ from $|\psi_t\rangle$ is greater than $1 - \alpha$. □

Remark 3.4 (Precision of T_f). A T_f with an $\epsilon = \mathcal{O}(1/t)$ is sufficient to obtain a constant success probability. As ϵ is an amplitude norm, this means that the measurement of $T_f |x\rangle |\phi\rangle$ shall fail to produce the correct value $f(x)$ with a probability in $\mathcal{O}(1/t^2)$. As the gain of quantum search against classical search is quadratic, it means that quantum search works when the equivalent classical search (assuming that it has access to an equivalently imperfect test procedure) is not expected to yield any incorrect value.

3.2 Nested search

The previous algorithms solved the problem of *unstructured search*, that is, given a search space X and a predicate P, find $x \in X$ such that $P(x)$. In this section, we propose an algorithmic framework for nested quantum search which can be used to express both classical and quantum search strategies. This framework was introduced in [BNS19b] and is used in Chapter 11 to present the cryptanalyses of AES.

3.2.1 Classical nested search

Classical nested search combines multiple searches to find a good element. This section presents the cases where the test function has a structure that allows to nest the search.

3.2.1.1 Conjunction of tests.

In practice, the test function can be a complicated formula, and may be expressed as the conjunction of smaller test functions, $P = P_1 \wedge P_2$. In that case, performing the complete test for all the values is likely to be suboptimal: if P_1 fails, we do not need to compute further. This approach is often called *early abort*, or *lazy evaluation*, and can be seen as a *nested search*: first, we search for values that fulfill P_1, and then, among those values, we look for one that fulfills P_2.

The naive approach has a cost of

$$c_{\text{search}}(X, P_1 \wedge P_2) = c_{\text{trials}}(X, P_1 \wedge P_2) \left(c_{\text{sample}}(X) + \underbrace{c_{\text{test}}(P_1) + c_{\text{test}}(P_2)}_{\text{cost to test } P_1 \wedge P_2.} \right).$$

If we note $X_{|P_1} = \{x \in X | P_1(x)\}$ the subset of X that satisfies P_1, then the cost of the lazy evaluation is

$$c_{\text{search}}(X, P_1 \wedge P_2) = \underbrace{c_{\text{trials}}(X, P_1 \wedge P_2) \left(c_{\text{sample}}(X) + c_{\text{test}}(P_1)\right)}_{\text{Test of } P_1 \text{ for all values}} + \underbrace{c_{\text{trials}}(X_{|P_1}, P_2) c_{\text{test}}(P_2)}_{\text{Test of } P_2\text{, only for values that satisfy } P_1}.$$

We can remark that $c_{\text{trials}}(X, P_1 \wedge P_2) = c_{\text{trials}}(X, P_1) c_{\text{trials}}(X_{|P_1}, P_2)$. Moreover, the cost of sampling a value in $X_{|P_1}$ is exactly the cost of finding a value in X which satisfies

P_1, that is, $c_{\text{sample}}(X_{|P_1}) = c_{\text{search}}(X, P_1)$. Hence, we can rewrite the previous equation as

$$c_{\text{search}}(X, P_1 \wedge P_2) = c_{\text{trials}}(X_{|P_1}, P_2) \left(\underbrace{c_{\text{trials}}(X, P_1)\,(c_{\text{sample}}(X) + c_{\text{test}}(P_1))}_{c_{\text{sample}}(X_{|P_1})} + c_{\text{test}}(P_2) \right).$$

Definition 3.3 (Filter)**.** Let X be a set, P be a predicate. A filter is an algorithm that samples in X and uses the evaluation of P to produce a sampling in the subset $X_{|P} = \{x \in X | P(x)\}$.

3.2.1.2 The case of product spaces.

Another particular case is when we have independent predicates, that is, $X = X_1 \times X_2$, and we have a predicate P_1 on X_1, and P_2 on X_2. In that case, we can sample easily on $X_{|P_1 \wedge P_2} = X_{1|P_1} \times X_{2|P_2}$, with

$$c_{\text{search}}(X, P_1 \wedge P_2) = \underbrace{c_{\text{trials}}(X_1, P_1)\,(c_{\text{sample}}(X_1) + c_{\text{test}}(P_1))}_{c_{\text{search}}(X_1, P_1)} + \underbrace{c_{\text{trials}}(X_2, P_2)\,(c_{\text{sample}}(X_2) + c_{\text{test}}(P_2))}_{c_{\text{search}}(X_2, P_2)}$$

However, this approach is suboptimal if we want to sample multiple times in $X_{1|P_1} \times X_{2|P_2}$, as we redo the tests for both values each time, and we could afford to only change one of them. In practice, we're interested in the case where only one solution remains in the end, so this may be an issue.

Let's say that we also have a predicate P_3 that only accepts 1 solution (x, y), that $X_1, X_2, X_{1|P_1}, X_{2|P_2}$ have respectively N_1, N_2, M_1, M_2 elements, and that sampling in X_1 and X_2 is free.

The direct search in $X_{1|P_1} \times X_{2|P_2}$ would have a cost of

$$M_1 M_2 \underbrace{\left(\frac{N_1}{M_1} c_{\text{test}}(P_1) + \frac{N_2}{M_2} c_{\text{test}}(P_2) \right)}_{\text{Sampling in } X_{1|P_1} \times X_{2|P_2}} + \underbrace{M_1 M_2 c_{\text{test}}(P_3)}_{\text{Test of } P_3 \text{ in } X_{1|P_1} \times X_{2|P_2}}$$

This direct approach samples for each pair in X_1 and X_2, which is suboptimal. We can do better if instead of searching in $X_{1|P_1} \times X_{2|P_2}$ for the predicate P_3, we can search in $X_{1|P_1}$ for the predicate

$$P'(x) = \exists y \in X_{2|P_2} : P_3(x, y)$$

Such a predicate is fairly easy to check: it is a search in $X_{2|P_2}$. Hence,

$$c_{\text{test}}(P') = c_{\text{search}}(X_{2|P_2}, P_3(x, \cdot))$$

In our example, $c_{\text{test}}(P') = M_2\left(\frac{N_2}{M_2} + c_{\text{test}}(P_3)\right)$. Hence, as we need to perform it M_1 times, the final cost is

$$\underbrace{N_1 c_{\text{test}}(P_1)}_{\text{Test all values in } X_1} + \underbrace{M_1 N_2 c_{\text{test}}(P_2)}_{\text{For each value of } X_{1|P_1}\text{, test all values of } X_2} + \underbrace{M_1 M_2 c_{\text{test}}(P_3)}_{\text{Test } P_3 \text{ only if we are in } X_{1|P_1} \times X_{2|P_2}}$$

Remark 3.5. Here, the predicate depends on the previous value x, and the number of solutions depends on x. If we expect at most one solution, then this is not an issue : we exhaust all $X_{2|P_2}$ for each possible x. If we can have many solutions, then it can be cumbersome, as we may want to abort a search on $X_{2|P_2}$ if, after a test on a fraction of the values, we have failed to find a good one.

3.2.2 Quantum nested search

In quantum computing, we cannot save time by branching, as we need to compute all the possible cases. This makes the notion of lazy evaluation less intuitive. Amplitude amplification with the notion of filters allows us to achieve roughly the same kind of costs than with classical nested searches, but with a square root applied on the terms corresponding to a search. These terms were denoted $c_{\text{trials}}(X, P)$ previously, and are denoted $c_{\text{amplify}}(X, P)$ in this section to reflect the use of amplitude amplification.

Amplitude amplification. Amplitude amplification can be seen as a filter: it takes as input a circuit that samples elements in a search space, a test circuit that checks for a predicate, and samples the set filtered by the predicate. Using the notations of Theorem 3.2 we have

$$c_{\text{search}}(X, P) = \underbrace{c_{\text{amplify}}(X, P)}_{2\frac{\pi}{4}\frac{1}{\sqrt{\alpha}}}\left(\underbrace{c_{\text{sample}}(X)}_{\text{Cost of } C} + \underbrace{c_{\text{test}}(P)}_{\text{Cost of computing } f}\right) + \underbrace{c_{\text{sample}}(X)}_{\text{Initialization cost}}.$$

The number of iterations is $\frac{\pi}{4}\frac{1}{\sqrt{\alpha}}$ if the classical number of iterations was $\frac{1}{\alpha}$. The factor 2 comes from the fact that we have to apply C twice, and we need to uncompute the application of f. The initialization cost comes from the fact that the initial state of amplitude amplification is the superposition of values in X. We will neglect this term, as we consider amplifications with a large number of iterations. This would not be negligible if we performed an asymptotic nesting of quantum searches of fixed size. We can apply directly our method of nested search, with the c_{amplify} terms that cost *almost* the square root of the classical c_{trials}.

Nesting overhead. This small constant factor has some repercussions on the efficiency of quantum search: the quantum search has a multiplicative overhead that depends on the number of nested searches. This overhead can change the best search layout. For the previous example with product spaces, classically, nesting was always an improvement, from

$$\begin{array}{rl} & N_1M_2c_{\text{test}}(P_1) \; + N_2M_1c_{\text{test}}(P_2) + M_1M_2c_{\text{test}}(P_3) \\ \text{to} & N_1c_{\text{test}}(P_1) \quad + N_2M_1c_{\text{test}}(P_2) + M_1M_2c_{\text{test}}(P_3) \end{array}$$

With amplitude amplification, the two approaches have respectively a cost of

$$\begin{array}{rl} & \frac{\pi^2}{4}\sqrt{N_1M_2}c_{\text{test}}(P_1) \; + \frac{\pi^2}{4}\sqrt{N_2M_1}c_{\text{test}}(P_2) + \frac{\pi}{2}\sqrt{M_1M_2}c_{\text{test}}(P_3) \\ \text{and} & \frac{\pi^2}{4}\sqrt{N_1}c_{\text{test}}(P_1) \quad + \frac{\pi^3}{8}\sqrt{N_2M_1}c_{\text{test}}(P_2) + \frac{\pi^2}{4}\sqrt{M_1M_2}c_{\text{test}}(P_3) \end{array}$$

The additional nesting divides the number of iterations of $c_{\text{test}}(P_1)$ by $\sqrt{M_2}$, but multiplies the other terms by $\frac{\pi}{2}$. Hence, depending on their relative cost, nesting can be a bad strategy in a quantum search.

Writing algorithms. In order to write algorithms in this framework, we use the following notation:

1: **Filter** $x \in X$
2: **If** not $P_1(x)$: **Abort** ▷ Produce $X_{|P_1}$
3: **If** $P_2(x)$ **then** ▷ Can assume $P_1(x)$
4: $y \leftarrow f(x)$ ▷ $y \in f(X_{|P_1 \wedge P_2})$
5: **Filter** y
6: **If** not $Q_3(y)$: **Abort** ▷ Checks $P_3(x) = Q_3(f(x))$
7: **End Filter**
8: **Return** x ▷ Sample in $X_{|P_1 \wedge P_2 \wedge P_3}$
9: **End Filter**

This example algorithm is a nested search for x in $X_{|P_1 \wedge P_2}$, where we search in $X_{|P_1}$ before testing for P_2. It can be implemented classically or quantumly. This formalism and these notations are used in Chapter 11 to describe a cryptanalysis of AES.

3.3 Collision search

The previous section presented one of the most versatile quantum algorithms, which can be applied, notably, to any problem where we can test the correct solution. This is generally the case in cryptography: as some operations can only be done with a secret, it is generally easy to check if we know it. We present here some algorithms for a slightly more structured problem which is also pervasive in cryptography: collision search.

Problem 3.2 (Collision Search)**.** *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a function. Find $x \neq y$ such that $f(x) = f(y)$.*

In some cases, we are interested in a collision between two functions, that is given two functions f and g, finding x and y such that $f(x) = g(y)$. We can reduce this problem to the previous one by considering

$$F : b, x \mapsto \begin{cases} f(x) & \text{if } b = 0, \\ g(x) & \text{if } b = 1. \end{cases}$$

A collision on F will correspond to a collision between f and g 50% of the time.

3.3.1 Classical resolution

Multiple classical algorithms have proposed to search for collisions, and two of them are presented below. A survey of collision algorithms can be found in [Jou09].

3.3.1.1 Naive algorithm

We can expect to find a collision on n bits among little more than $2^{n/2}$ values. Indeed, $2^{n/2}$ values make for $2^{n-1} - 2^{n/2-1}$ pairs, and we can expect that one of then will satisfy the n-bit constraint. This is often referred to as the *birthday paradox*. This leads to a simple collision algorithm: query around $2^{n/2}$ values to f, sort the values according to $f(x)$, and then look sequentially for a collision in the sorted list. The time, memory and query cost is in $\mathcal{O}\left(2^{n/2}\right)$.

3.3.1.2 Low-memory algorithm

There is an improved method, which allows to avoid the memory usage with a comparable time cost: Pollard's rho algorithm [Pol75]. Its idea is to use the sequence of iterates of f from a given base point x_0. This sequence is ultimately periodic, as the domain of f is finite. The periodic sequence has length c and begins after t steps. This gives the graph of the sequence its ρ shape, described in Figure 3.2. The graph contains exactly one collision if $t \neq 0$, with $f(x_{t-1}) = f(x_{t+c-1})$.

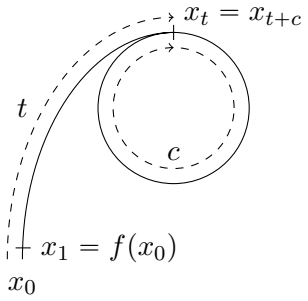


Figure 3.2: ρ-shaped graph of the iterates of x_0 under the function f: $x_{i+1} = f(x_i)$. The collision happens with $f(x_{t-1}) = x_t = x_{t+c} = f(x_{t+c-1})$.

The collision in the graph can be found using the simple Algorithm 3.1. The algorithm first looks for an index k such that $x_k = x_{2k}$. From the structure of the graph,

this implies that the difference between k and $2k$ is a multiple of c. Hence, c divides k. Once the value of x_k is known, the collision can be easily found, as it corresponds to $x_t = x_{t+k}$. It is sufficient to walk from x_0 and x_k until a collision is found.

Algorithm 3.1 Pollard ρ algorithm for collisions.

Input: A function f, a starting value x_0
Output: x_i, x_j such that $x_i \neq x_j$ and $f(x_i) = f(x_j)$

1: $a \leftarrow f(x_0)$
2: $b \leftarrow f(f(x_0))$
3: **While** $a \neq b$ **do** ▷ Cycle detection: $a = f^{(i)}(x_0), b = f^{(2i)}(x_0)$
4: $\quad a \leftarrow f(a)$
5: $\quad b \leftarrow f(f(b))$
▷ $f^{(k)}(x_0) = a = b = f^{(2k)}(x_0)$
6: $a \leftarrow x_0$
7: **While** $f(a) \neq f(b)$ **do** ▷ Collision finding: $f^{(t)}(x_0) = f^{(t+k)}(x_0)$
8: $\quad a \leftarrow f(a)$
9: $\quad b \leftarrow f(b)$
Return (a, b)

Complexity. The complexity of this approach depends on the structure of the graph. It turns out that for random functions and a random x_0, $t = \mathcal{O}\left(2^{n/2}\right)$, $c = \mathcal{O}\left(2^{n/2}\right)$. For a function with a distinct domain and codomain, or if a function is considered too structured, the same algorithm can be applied by composing f with a random permutation g that maps the codomain to the domain of f. Overall, this low-memory algorithm has a time and query cost in $\mathcal{O}\left(2^{n/2}\right)$. Moreover, it can also be efficiently parallelized using distinguished points, with a method proposed by van Oorschot and Wiener [vW99].

3.3.2 Quantum resolution

The quantum algorithm of Brassard, Høyer and Tapp [BHT98] reduces the collision problem to a search problem, on which we can apply Grover's algorithm. It can be seen as a quantum version of the naive classical algorithm. This algorithm has a query complexity in $\mathcal{O}(2^{n/3})$, which matches the quantum lower bound of [AS04].

They consider a list of $2^{n/3}$ inputs $(x_i)_{i<2^{n/3}}$, and compute (classically) the list

$$L = \left\{ f(x_i) | i < 2^{n/3} \right\}$$

Now, the search can be done with the test function $h(x) = \exists i : f(x) = L[i] \wedge x \neq x_i$. If f is a random function, we can expect that any $x \notin (x_i)_{i<2^{n/3}}$ has $f(x) \in L$ with probability $2^{-2n/3}$. Hence, we can expect that roughly $2^{n/3}$ values satisfy $h(x)$ is true. A Grover search on h will then produce an x with $h(x)$ true in $\mathcal{O}(2^{n/3})$ application of h.

As h only needs L and one application of f, a collision on f can be found in $\mathcal{O}(2^{n/3})$ queries to f.

Complexity. This algorithm has a quantum time, quantum memory and query cost in $\mathcal{O}(2^{n/3})$, and can be seen as a variant of the classical naive algorithm. Unfortunately, the classical low-memory variants cannot be improved: indeed, they depend on the structure of the graph of the function, and a quantum algorithm would still need to compute its iterates. In the general case, there is no way to compute an iterate more efficiently than the sequential computation, which annihilates any quantum gain. A similar issue occurs with cycle-based slide attacks, which are presented in Section 9.5. It is possible to avoid the large quantum memory requirement at the expense of a worse time and query cost, in $\mathcal{O}\left(2^{2n/5}\right)$ and using $\mathcal{O}\left(2^{n/5}\right)$ *classical* memory [CNS17], using distinguished points. Overall, the gain for collisions is worse than the gain for generic search, as the time gain is smaller, and some memory is required.

In fact, the Brassard-Høyer-Tapp algorithm solves a harder problem than collision: indeed, the set of values on which we want to collide is fixed beforehand. This problem corresponds to *multi-target preimage search*: given a set $\mathcal{S}$ of L images of f, find a preimage for one of them. Classically, the best we can do is the classical version of the Brassard-Høyer-Tapp algorithm: have $\mathcal{S}$ in memory, and sample some x until we find a matching $f(x)$ in $\mathcal{S}$. If the size of $\mathcal{S}$ is fixed, we recover the quadratic gain of generic search: classically, the time is $2^n/L$ with L memory, quantumly it becomes $\sqrt{2^n/L}$ with L memory.

3.3.3 Structured collisions

The state of quantum algorithms for collision makes this problem comparatively less appealing than generic search for an application in cryptanalysis. This does not mean that a problem that can be reduced classically to collision is hard to solve with a quantum computer. Indeed, factoring [Pol75] and discrete logarithm [Pol78] can be reduced to collision search. Moreover, this is the best known generic approach for the discrete logarithm over an elliptic curve.

This section presents some problems which can be seen as a structured collision search problem. Classically, they reduce to standard collision search, but quantumly, some much more efficient algorithms, presented in the next chapters, can be used.

Problem 3.3 (Hidden Subgroup Problem (HSP))**.** *Let $\mathcal{G}$ be a group, $H \leqslant G$ a subgroup of $\mathcal{G}$ and X a set. Let $f : \mathcal{G} \to X$ be a function with the promise that for all $(x, y) \in \mathcal{G}^2$, $[f(x) = f(y) \Leftrightarrow \exists h \in H : x = yh]$. Given oracle access to f, find a basis of H.*

Remark 3.6 (Hidden Period). If H is a cyclic subgroup, we say that f is *periodic*, and its period is a generator of H.

The most notable examples of the Hidden Subgroup algorithms are Simon's algorithm [Sim94], which is presented in Chapter 4 and Shor's algorithm [Sho94], which is presented in Section 5.3.2, which can be used to solve the factoring and discrete logarithm problems. In general, in the instances we will consider, we have a hidden period, but in some cases the hidden subgroup is larger, as for example in the AEZ cryptanalysis of Chapter 8.

Theorem 3.4 (General query bound [EHK04, Theorem 1])**.** *The quantum query complexity of the hidden subgroup problem in any group $\mathcal{G}$ is in $\mathcal{O}\left(\log^4 |\mathcal{G}|\right)$.*

Problem 3.4 (Hidden Shift Problem)**.** *Let G be a group, X a set, $f, g : G \to X$ two injective functions, $s \in G$, with the promise that for all x, $f(x) = g(s \cdot x)$. Given oracle access to f and g, find s.*

Several algorithms for solving the hidden shift problem are presented in Chapter 5.

Remark 3.7 (Shifts and subgroups). If G is abelian, the hidden shift problem in G is a hidden subgroup problem in the semi-direct product $G \rtimes \mathbb{Z}/(2)$. If G is not abelian, then the hidden shift problem is a hidden subgroup problem in the wreath product $G \wr \mathbb{Z}/(2)$.

Chapter 4
Simon's Algorithm

This chapter presents Simon's hidden subgroup algorithm [Sim94], some results on its applicability in situations where the structure of the hiding function is not exactly as expected which are derived from [Kap+16] and the natural generalization of the algorithm to larger subgroups we will use to attack AEZ in Chapter 8. The usage of Simon's algorithm as a test function is presented in Chapter 6, and multiple attacks leveraging it are presented in Chapter 7.

Contents

Simon's algorithm [Sim94] tackles the Hidden Subgroup problem when the group is $(\mathbb{Z}/(2))^n$, and has originally been stated as a Hidden Period problem.

We can formulate the problem as follows:

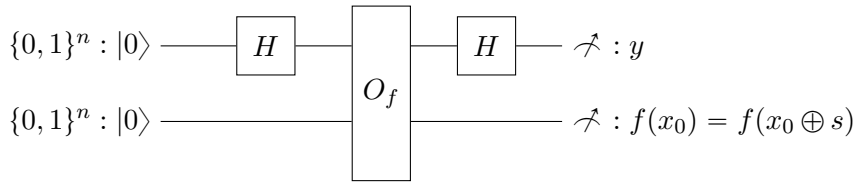
Problem 4.1 (Simon's Problem)**.** *Let n be an integer, $s \in \{0,1\}^n$ and X a set. Let $f : \{0,1\}^n \to X$ be a function with the promise that for all $(x,y) \in (\{0,1\}^n)^2$, $[f(x) = f(y) \Leftrightarrow x \oplus y \in \{0,s\}]$. Given oracle access to f, find s.*

4.1 Algorithm description

While classically this problem is as hard as collision search, Simon's quantum algorithm solves it in polynomial time by using Circuit 4.1. It can be described as Algorithm 4.1, which produces a random value orthogonal to s. This routine constructs the superposition of all input-output pairs of f (step 3). Then, the output is measured and discarded (step 4). This projects the input to a superposition of two values whose xor is s, and the following steps extract some information on s.

The result is a measure of the state

$$\sum_{j=0}^{2^n-1} (-1)^{x_0 \cdot j} \left(1 + (-1)^{s \cdot j}\right) |j\rangle .$$



Circuit 4.1: Simon's circuit

If $s \cdot j \neq 0$, the amplitude of a value j is 0. Hence, this routine samples uniformly a value j orthogonal to s.

Algorithm 4.1 Simon's routine

Input: n, $O_f : |x\rangle|0\rangle \mapsto |x\rangle|f(x)\rangle$ with $f : \{0,1\}^n \to X$ of period s
Output: y with $y \cdot s = 0$

1: Initialize two n-bits registers : $|0\rangle|0\rangle$
2: Apply H gates on the first register, to compute $\sum_{x=0}^{2^n-1} |x\rangle|0\rangle$
3: Apply O_f, to compute $\sum_{x=0}^{2^n-1} |x\rangle|f(x)\rangle$
4: Measure an $f(x_0)$ in the second register, to get $|x_0\rangle + |x_0 \oplus s\rangle$ in the first
5: Reapply H gates on the register, to compute

$$\sum_{j=0}^{2^n-1} (-1)^{x_0 \cdot j} |j\rangle + \sum_{j=0}^{2^n-1} (-1)^{(x_0 \oplus s) \cdot j} |j\rangle$$

6: The register is in the state $\sum_{j=0}^{2^n-1} (-1)^{x_0 \cdot j} \left(1 + (-1)^{s \cdot j}\right) |j\rangle$
7: Measure j in the first register, return it.

The complete algorithm calls the routine until the values span a space of maximal rank or, if the rank is unknown, a fixed T times. In practice, $T = n + \mathcal{O}(1)$ is sufficient to succeed.

Proposition 4.1 (Simon's algorithm complexity)**.** *If $T = n + \alpha$, Simon's algorithm fails with a probability lower than $2^{-\alpha}$.*

Proof. Let $\alpha \geqslant 0$, $y_1, \ldots, y_T$ be the $T = n + \alpha$ outputs of Simon's routine. The y_i are sampled uniformly in the set $\{y | y \cdot s = 0\}$. Let

$$M = \begin{bmatrix} y_1 \\ \ldots \\ y_T \end{bmatrix}$$

be the matrix whose rows are the y_i. Then, the rank of the y_i is the rank of M.

If $s = 0$, the maximal rank is n, and M is of maximal rank if and only if its columns form a free family, as $T \geqslant n$. The first column is non-zero with probability $1 - \frac{1}{2^T}$. If

Algorithm 4.2 Simon's algorithm [Sim94]

Input: n, $O_f : |x\rangle|0\rangle \mapsto |x\rangle|f(x)\rangle$ with $f : \{0,1\}^n \to X$ of period s, T
Output: s

```
V = ∅
For i from 1 to T do
    Get y from Algorithm 4.1
    Add y to V
If rank(V) = n then
    Return 0
Else
    If rank(V) = n − 1 then
        Return The unique s ≠ 0 orthogonal to V.
    Else
        Return Failure
```

$y_1, \ldots, y_{i-1}$ form a free family, y_i is linearly independent from them with probability $1 - \frac{1}{2^{T-i-1}}$, as $|\langle y_1, \ldots, y_{i-1}\rangle| = 2^{i-1}$. Hence, the columns form a free family with probability

$$\prod_{i=0}^{n-1} \left(1 - \frac{1}{2^{T-i}}\right).$$

Taking the log, we obtain

$$\sum_{i=0}^{n-1} \log\left(1 - \frac{1}{2^{T-i}}\right).$$

Developing in power series produces

$$\sum_{i=0}^{n-1} -\sum_{j=1}^{\infty} \frac{1}{j2^{(T-i)j}}$$

Interchanging the sums produces

$$-\sum_{j=1}^{\infty} \frac{1}{j2^{Tj}} \sum_{i=0}^{n-1} 2^{ji}$$

As $\sum_{i=0}^{n-1} 2^{ji} \leqslant 2^{jn}$, the sum is greater than

$$-\sum_{j=1}^{\infty} \frac{1}{j2^{(T-n)j}}$$

Factoring the power series and using $T = n + \alpha$ produces

$$\log\left(1 - \frac{1}{2^{\alpha}}\right)$$

Hence, the success probability is greater than $1 - \frac{1}{2^\alpha}$.

If $s \neq 0$, the maximal rank is $n-1$. The y_i are of maximal rank if the $n-1$ first columns of M forms a free family, which occurs with probability $1 - \frac{1}{2^{\alpha+1}}$. □

Simon's algorithm is generally presented with a hidden period, but as remarked by Brassard and Høyer [BH97], it also works for larger hidden subgroups.

Proposition 4.2 (Simon's Algorithm for Hidden subgroups)**.** *If f hides any subgroup H of $\mathbb{Z}/(2)^n$, Algorithm 4.2 can produce a basis of H in $n+\alpha$ queries with probability greater than $1 - \frac{1}{2^\alpha}$.*

Proof. If the function f hides the subgroup $H = \langle s_1, \dots, s_d \rangle$, then the preimages are no longer $\{x_0, x_0 \oplus s\}$, but $\{x_0 \oplus h | h \in H\}$. In that case, the state at step 4 of Algorithm 4.1 is

$$\sum_{h \in H} |x_0 \oplus h\rangle .$$

Hence, the Hadamard application produces

$$\sum_{h \in H} \sum_{j=0}^{2^n - 1} (-1)^{j \cdot (x_0 \oplus h)} |j\rangle .$$

This can be rewritten as

$$\sum_{j=0}^{2^n - 1} (-1)^{j \cdot x_0} \sum_{h \in H} (-1)^{j \cdot h} |j\rangle .$$

As $(s_1, \dots, s_d)$ is a basis of h, $\sum_{h \in H} (-1)^{j \cdot h} = \prod_{k=1}^{d} \left(1 + (-1)^{j \cdot s_k}\right)$. Hence, the amplitude of j is non-zero if and only if, for all s_k, $j \cdot s_k = 0$. This means we sample uniformly values orthogonal to H. The algorithm is still correct, and will produce a basis of H. □

4.2 Weakening the promise

The promise of Simon's problem is that the function needs to be identical on each coset of the hidden subgroup, and that it is injective between each coset. However, in practice, one might want to apply the algorithm on functions that partially fulfil the promise. This section studies the behaviour of Simon's algorithm when the constraints in each direction of the equivalence $[f(x) = f(y)] \Leftrightarrow x \oplus y \in \{0, s\}$ are relaxed.

4.2.1 Partial period

If some bad inputs do not satisfy the periodicity condition ($f(x) \neq f(x \oplus s)$), then Algorithm 4.1 will fail if at step 4, the x_0 from the measured $f(x_0)$ is bad, as the random vector in the end will not necessarily be orthogonal to the period. Hence, the algorithm will succeed if all the x_0 are good.

Proposition 4.3 (Partial period for Simon's algorithm)**.** *Let n be an integer, $s \in \{0,1\}^n$, $G \in \{0,1\}^n$, $p = |G|/2^n$ and X a set. Let $f : \{0,1\}^n \to X$ be a function with the promise that for all $(x,y) \in (\{0,1\}^n)^2$, $[f(x) = f(y) \Rightarrow x \oplus y \in \{0,s\}], [x \oplus y \in \{0,s\}$ and $x \in G \Rightarrow f(x) = f(y)]$. Simon's algorithm applied on f retrieves s on T steps with a probability lower than $\left(\frac{1}{2}(1+p)\right)^T \left(1 - \frac{1}{2^{T-n}}\right)$*

Proof. For each query, if the promise is fulfilled, then the algorithm behaves correctly. If it is not, then we sample a value orthogonal to s with probability one half. G is the set of inputs on which the promise is fulfilled, hence p is the probability that one query corresponds to an input with a fulfilled promise. Hence, the probability to sample only values orthogonal to s is

$$\sum_{i=0}^{T} \binom{T}{i} p^{T-i}(1-p)^i 2^{-i} = \left(\frac{1}{2}(1+p)\right)^T.$$

We can retrieve s only if the values are both all orthogonal and of maximal rank, which add the term $\left(1 - \frac{1}{2^{T-n}}\right)$. □

For example, if only half of the inputs satisfy the condition, then the success probability will be lower than $\left(\frac{3}{4}\right)^n$, which quickly becomes negligible.

Remark 4.1. If the number of bad inputs is a fixed fraction of the input, the success probability decreases exponentially in n, and the quantum gain of the algorithm disappears. In order to have a fixed success probability, we need to have $1 - p = 2^{-\Omega(n)}$, that is, the *number* of bad inputs shall not be greater than $2^{\alpha n}$, with $\alpha < 1$.

4.2.2 Non-injective functions

Now, we can consider functions that break the other part of the promise: there exists (x,y) s.t $f(x) = f(y)$ and $x \oplus y = t \notin \{0,s\}$. As in the previous section, such t corresponds to a partial period. However, here, they are mere artifacts that we want to rule out, and not the value we want to compute. Hence, from Proposition 4.3, we know that we cannot expect to find a partial period if it does not occur for almost all x. As it cannot identify t, this additional partial period shall not disrupt the computation.

However, we cannot expect to have a functioning algorithm in all cases. Indeed, let's consider

$$f_s : \begin{array}{ccc} \{0,1\}^n & \to & \{0,1\} \\ x & \mapsto & \begin{cases} 1 \text{ if } x \in \{0,s\} \\ 0 \text{ otherwise} \end{cases} \end{array}.$$

The function f_s is periodic, of period s. This is also a test function for Grover's algorithm, which is optimal given only an oracle access to the function. Hence, we cannot hope to be polynomial, or even subexponential, in that case.

To quantify how suitable the function is for Simon's algorithm, we define p_0 as proposed in [Kap+16]

$$p_0 = \max_{t \notin \{0,s\}} \Pr_x[f(x \oplus t) = f(x)].$$

This value estimates the probability that any given t is present as an additional period for some of the output vectors of Algorithm 4.1. It allows to bound the success probability of Simon's algorithm.

Proposition 4.4 (Success probability with more preimages [Kap+16, Theorem 1])**.** *Let f be a periodic function, p_0 be defined as above. After cn steps, Simon's algorithm on f succeeds with probability greater than $1 - \left(2\left(\frac{1+p_0}{2}\right)^c\right)^n$. If $c > 1/(1 - \log_2(1 + p_0))$, Simon's algorithm returns the period with probability exponentially close to 1.*

Proof. We need to estimate the probability that a given output of Algorithm 4.1 is orthogonal to $t \neq s$. There are two cases: if at step 4, the x_0 satisfies $f(x_0) = f(x_0 \oplus t)$, then the output will be orthogonal to t with certainty. If not, then it is orthogonal with probability one half. The first case occur with probability lower than p_0, and the second with probability greater than $1 - p_0$. Hence, the probability that the output is orthogonal to t is lower than $\frac{1+p_0}{2}$. Hence, the probability that all the outputs are orthogonal to t is lower than $(\frac{1+p_0}{2})^{cn}$.

As there are less than 2^n false periods, the outputs will all be orthogonal to a $t \notin \{0, s\}$ with probability lower than $2^n(\frac{1+p_0}{2})^{cn} = \left(2\left(\frac{1+p_0}{2}\right)^c\right)^n$. This is exponentially close to zero if

$$\begin{aligned} & 2\left(\tfrac{1+p_0}{2}\right)^c < 1 \\ \Leftrightarrow\ & c \log_2\left(\tfrac{1+p_0}{2}\right) < -1 \\ \Leftrightarrow\ & c > \tfrac{-1}{\log_2(1+p_0)-1} \\ \Leftrightarrow\ & c > \tfrac{1}{1-\log_2(1+p_0)} \end{aligned}$$

In that case, as we have a vector orthogonal to each non-period, we have a family of maximal rank, hence the period can be extracted. □

Remark 4.2. If $p_0 < 1/\log_2(e)$, we can take $c = 1/(1 - \log_2(e)p_0)$.

Proposition 4.5 (Simon's complexity with more preimages)**.** *Let f be a periodic function, p_0 be defined as above. Simon's algorithm on f fails with probability lower than $2^{-\alpha}$ after $\frac{1}{1-\log_2(1+p_0)}(n + \alpha)$ queries.*

Proof. From Proposition 4.4, after cn queries, the failure probability is at most $\left(2\left(\frac{1+p_0}{2}\right)^c\right)^n$. Hence, we have

$$\begin{aligned}
\tfrac{1}{2^\alpha} &= \left(2\left(\tfrac{1+p_0}{2}\right)^c\right)^n \\
\Leftrightarrow\ -\alpha &= n\left(1 + c\log_2\left(\tfrac{1+p_0}{2}\right)\right) \\
\Leftrightarrow\ \ n+\alpha &= -cn\log_2\left(\tfrac{1+p_0}{2}\right) \\
\Leftrightarrow\ \ cn &= \frac{1}{\log_2\left(\frac{2}{1+p_0}\right)}(n+\alpha)
\end{aligned}$$

As cn is the total number of queries, the lemma holds. □

Random functions. The previous propositions allow to estimate the number of queries given a function, but it requires a specific study of the function. We show here that in the typical case, the overhead is negligible. To do so, we need a notion of random periodic functions.

Definition 4.1 (Random periodic function)**.** Let $f : \{0,1\}^n \to \{0,1\}^n$ be a function of period s. We say that f is a random periodic function if f restricted to $\{0,1\}^n/(s)$ is a random function.

The distribution of p_0 is linked to the number of solutions of the equation $f(x \oplus t) \oplus f(x) = 0$. This corresponds to some coefficients of the Differential Distribution Table of f. The distribution of these coefficients for a random function has been studied by Daemen and Rijmen [DR07], who heuristically obtained that for f from n to m bits, they are independent, and follow a Poisson distribution of parameter $\lambda = 2^{n-m-1}$. This is of course an approximation, as the sum of the values in the DDT is fixed.

Heuristic 4.1 (Overhead for random functions)**.** *For random periodic functions, p_0 converges to 0 and c converges to 1 when n grows.*

Justification. In our case, if $s \neq 0$ the random function has $n-1$ bits of input and n bits of output, and if $s = 0$ it has n bits of input and n bits of output, hence the parameter of the Poisson distribution is either $\lambda = 2^{-1}$ or 2^{-2}. As we have a Poisson distribution, by the Chernoff bound, we have, for all t,

$$Pr[X_t \geqslant a] \leqslant e^{-\lambda}\left(\frac{e\lambda}{a}\right)^a$$

The value p_0 is the maximum of all X_t, divided by 2^{n-2}. Hence, for all c,

$$\begin{aligned}
Pr[p_0 < c] &= Pr[\max_t X_t < c2^{n-2}] = \left(Pr[X_t < c2^{n-2}]\right)^{2^{n-1}} \\
&= \left(1 - Pr[X_t \geqslant c2^{n-2}]\right)^{2^{n-1}} \geqslant \left(1 - e^{-\lambda}\left(\frac{e\lambda}{c2^{n-2}}\right)^{c2^{n-2}}\right)^{2^{n-1}} \\
Pr[p_0 < c] &\geqslant 1 - 2^{n-1}e^{-\lambda}\left(\frac{e\lambda}{c2^{n-2}}\right)^{c2^{n-2}}
\end{aligned}$$

Hence, taking $c = n/2^{n-2}$, we have

$$Pr[p_0 \geqslant \frac{n}{2^{n-2}}] < 2^{n-1} e^{-\lambda} \left(\frac{e\lambda}{n}\right)^n = \frac{e^{-\lambda}}{2} \left(\frac{2e\lambda}{n}\right)^n .$$

□

Remark 4.3 (Rate of convergence). The previous heuristic estimate shows a convergence that is quite fast: for example, for $n = 32$, we have $p_0 \geqslant 2^{-25}$ with a probability smaller than 2^{-78}.

Heuristic 4.1 suggests that except in some degenerate cases, Simon's algorithm behaves perfectly with functions that can have more than 2 preimages per image.

4.2.3 Families of functions

Simon's problem is generally stated with one function f that has a period. However, if each query calls a *different* function, but all of them have the *same* hidden period, then Simon's routine will still output a y orthogonal to the period, and Simon's algorithm will also succeed, as we sample values in a set that only depends on the period, and not on the specific function we query.

Chapter 5
Abelian Hidden Shift Algorithms

This chapter will present the three known approaches to tackle the hidden shift problem in cyclic groups as well as an algorithm from Ettinger and Høyer [EH99] that shows that the quantum query complexity of the problem is linear in the size of the group. Kuperberg [Kup05] proposed the first subexponential algorithm to solve the dihedral hidden subgroup problem, which is another way to present the cyclic hidden shift problem. This article has been quickly followed by a note by Regev [Reg04], who proposed a polynomial-memory variant of the algorithm, and showed some links with subset-sum algorithms. This method was later generalized by Childs, Jao and Soukharev [CJS14] to attack ordinary isogenies in subexponential time. Finally, Kuperberg proposed another variant [Kup13], polynomial in quantum memory and more time-efficient than the two previous ones, which turned out to be very similar to a k-list algorithm. We have proposed some improvements over Kuperberg's first algorithm in [BN18], some improvements over the Childs, Jao and Soukharev algorithm in [Bon19b] and we have proposed different tradeoffs of Kuperberg's second algorithm in [BS18]. The generalizations to the group $(\mathbb{Z}/(2^p))^w$ and nonabelian groups have been proposed in [BN18].

Contents

5.1 The problem

The algorithms of this chapter aim at solving the following problem:

Problem 5.1 (Abelian Hidden Shift Problem)**.** *Let n be an integer, $\mathcal{G}$ an abelian group, X a set, $f, g : \mathcal{G} \to X$ two injective functions, $s \in \mathcal{G}$, with the promise that for all x, $f(x) = g(s + x)$. Given oracle access to f and g, find s.*

The following algorithms all follow the same structure: first, a generation algorithm uses the oracle functions to produce some random *phase elements*. Then, the shift is extracted from them. The most query-efficient algorithm (Section 5.5) directly uses the *phase elements*, while the more time-efficient algorithms use a combination algorithm recursively to produce some specific phase elements. Finally, given a target set of phase elements, the shift value is extracted. The only difference between the three time-efficient algorithms is the combination part.

5.2 Preliminaries: subset-sum and k-list

The hidden shift algorithms of Section 5.6.3 and Section 5.6.4 rely respectively on some algorithms for the subset-sum and k-list problems. This section recalls the relevant classical litterature.

5.2.1 Subset-sum algorithms

The subset-sum problem can be stated as follows:

Definition 5.1 (Subset-sum Problem in $\mathbb{Z}$)**.** Let $(x_1, \ldots, x_k) \in \mathbb{Z}^n, s \in \mathbb{Z}$.
Find $I \subset [1; n]$ such that $\sum_{i \in I} x_i = s$.

This problem is NP-complete [MD79], that is, any NP problem can be reduced to an instance of subset-sum. It can be instanciated with any group, but we're mostly interested in the case of the group $\mathbb{Z}/(2^r)$ with $r \simeq n$, and x_i uniform in $\mathbb{Z}/(2^r)$. In this regime, we expect to have around 1 solution, and this is often denoted as "hard instances", for which the best known algorithms are the slowest. Multiple algorithms have been proposed, both quantum and classical [SS81; BCJ11; Ber+13], all of them with an exponential complexity, in $\widetilde{O}(2^{cn})$, for a given constant $c < 1$.

These algorithms rely on list-merging techniques: the complete solution is contructed from lists of candidate for partial solutions. A summary of these algorithm is proposed in Table 5.1, and some of them are detailed below.

Table 5.1: Algorithms for the subset-sum problem

Algorithm	Type	Time	Memory
Exhaustive search	Classical	2^n	$\mathcal{O}(n)$
Claw-finding		$2^{n/2}$	$2^{n/2}$
[SS81]		$2^{n/2}$	$2^{n/4}$
[BCJ11]		$\mathcal{O}(2^{0.291n})$	$\mathcal{O}(2^{0.291n})$
Cycle-finding		$\mathcal{O}(2^{0.75n})$	$\mathcal{O}(n)$
[BCJ11]		$\mathcal{O}(2^{0.72n})$	$\mathcal{O}(n)$
Quantum search	Quantum	$2^{n/2}$	$\mathcal{O}(n)$
[Ber+13]		$\mathcal{O}(2^{0.241n})$	$\mathcal{O}(2^{0.241n})$
[HM18]		$\mathcal{O}(2^{0.226n})$	$\mathcal{O}(2^{0.226n})$

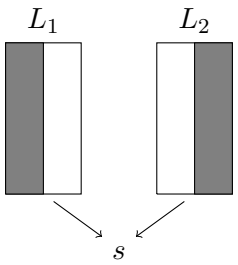


Figure 5.1: Claw finding

5.2.1.1 Claw-finding

A first solution for this problem is to split the inputs in half, and compute the sorted lists L_1 and L_2 of all the subset-sums over respectively $(x_1, \ldots, x_{\lfloor k/2 \rfloor})$ and $(x_{\lfloor k/2 \rfloor+1}, \ldots, x_k)$. Then, one can easily find the solution by iterating over L_1, and for each value, checking if there is a matching value in L_2.

The cost of this algorithm corresponds to the size of the two lists, in $\widetilde{O}(2^{n/2})$ time and memory.

Remark 5.1. Only one list needs to be effectively stored in memory, as the other can be computed on-the-fly.

5.2.1.2 Schroppel-Shamir algorithm [SS81]

The previous algorithm can be refined, and use a lower memory. The space of possible solutions of $\sum_{i=1}^{n} \varepsilon_i x_i = V$ is split into 4 parts represented as 4 lists L_1, L_2, L_3, L_4, with $L_1 = \sum_{i \leqslant n/4} \varepsilon_i x_i$, and so on.

The lists L_1 to L_4 contain all possible partial sums on a fourth of the variables. The intermediate lists L_{12} and L_{34} contain respectively the partial sums on the first and second half of the variables.

Without any other technique, the splitting would not gain anything, as the two intermediate lists would be of size $2^{n/2}$. The Schroeppel-Shamir algorithm gains by guessing $L_{12} \mod 2^{n/4}$. With that guess, the list is expected to be of size $2^{n/4}$. Conversely, $L_{12} \mod 2^{n/4}$ imposes the value $s - L_{12} \mod 2^{n/4}$ for L_{34}. Hence all the lists are expected to be of size $\widetilde{O}(2^{n/4})$. The solution will only be found for the correct guess of the intermediate value, requiring $2^{n/4}$ guesses overall, for a total cost of $\widetilde{O}(2^{n/2})$ time, but only $\widetilde{O}(2^{n/4})$ memory.

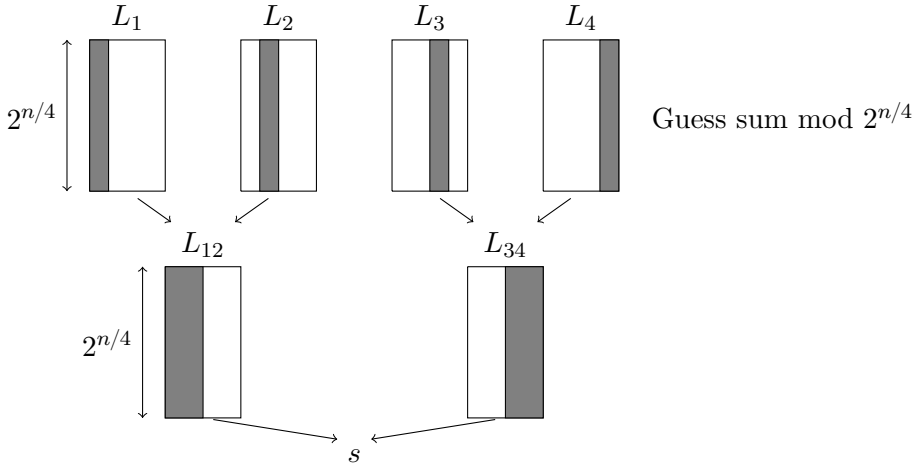


Figure 5.2: Schroeppel-Shamir merging. The subset of variables involved in each list is in grey

Finally, the merging in itself is the efficient generation of the intermediate lists (and of the complete solution) from the previous lists. As we want to produce a list of values with a constrained sum, we can sort the first list, and then check for each element of the second list if there is an element leading to a correct sum in the first list. The cost of the merging is the cost of sorting the input list and constructing the output list, here $\widetilde{O}(2^{n/4})$.

The algorithm from [BCJ11] uses similar techniques, but with a different splitting. Instead of considering a subset of the variables, they considered a partial sum with a smaller number of terms in it. They also allowed the ϵ_i in the intermediate step to lie in $\{-1, 0, 1\}$. With this approach, the merging has also to check for the consistency of the solutions, as the variables may overlap.

By splitting the sum in 8 and carefully choosing the size of the intermediate constraint and the ratio of -1, the authors of [BCJ11] obtained an asymptotic complexity of $\widetilde{O}(2^{0.291n})$ classical time and memory.

5.2.1.3 Polynomial-memory algorithms [BCJ11].

It is also possible to devise polynomial-memory algorithms for this problem. In [BCJ11], two polynomial-memory subset-sum algorithms are proposed, with a very similar time complexity.

The idea of the first algorithm is to rely on collision-finding. If we define

$$f_1 : \begin{cases} \{0,1\}^{\lfloor n/2 \rfloor} & \to & \{0,1\}^n \\ (\delta_1, \ldots, \delta_{\lfloor n/2 \rfloor}) & \mapsto & \sum_{i=1}^{\lfloor n/2 \rfloor} \delta_i x_i \end{cases},$$

$$f_2 : \begin{cases} \{0,1\}^{\lceil n/2 \rceil} & \to & \{0,1\}^n \\ (\delta_{\lfloor n/2 \rfloor+1}, \ldots, \delta_n) & \mapsto & s - \sum_{i=\lfloor n/2 \rfloor+1}^{n} \delta_i x_i \end{cases},$$

then a collision between f_1 and f_2 would lead to a subset-sum solution. If the two functions had the same domain and codomain size, we could apply a collision-finding algorithm to find the wanted values. Nevertheless, we can still apply the algorithm on truncated versions $f_1'(x) = f_1(x) \mod 2^{n/2}$ and $f_2'(x) = f_2(x) \mod 2^{n/2}$ to find (x, x') such that $f_1'(x) = f_2'(x')$ in time $O(2^{n/4})$. The pair (x, x') is a correct subset-sum solution modulo $2^{n/2}$. Hence, it is correct modulo 2^n with probability $2^{-n/2}$.

We can hence expect to find the correct solution in $2^{n/2}$ trials, by randomizing the f_1 and f_2 used. The complexity is then in $O(2^{3n/4})$ time, with only a polynomial memory.

The second algorithm uses the same method, but instead of splitting the variables, it splits between sums containing $n/4$ elements. This allows to slightly reduce the cost, to $O(2^{0.72n})$.

5.2.2 k-list algorithms

The k-list problem is a generalization of the subset-sum problem. It can be stated as:

Definition 5.2 (k-list in $\mathbb{Z}/(2^n)$)**.** Let $(L_1, \ldots, L_k) \in (\mathbb{Z}/(2^n))^N)^k$ be k lists of size N, $s \in \mathbb{Z}/(2^n)$.

Find $(x_1, \ldots x_k) \in [1; N]^k$ such that $\sum_{i=1}^{k} L[x_i] = s$.

The subset-sum problem is the particular case of lists of size 2. The generic problem has been introduced by Wagner [Wag02], who proposed a general algorithm based on 2-list merging. This has been extended with various works [MS12; NS15; Din18], in which many tradeoffs between time and memory are proposed, with various applications in cryptography.

Merging algorithms for subset-sum, like Figure 5.1 and 5.2 work if the constraints are strong enough to have only one solution, but also for weaker constraints, as they efficiently enumerate all the possible solutions. In that case, the output will be a list of solutions of the weak constraint. It has only an impact on the memory if the final number of solutions is too large.

5.2.2.1 Merging trees.

The general approach for k-list algorithm is to use recursively these algorithms to produce lists with increasingly constrained values, until we find the one we want. This recursive structure leads to a tree shape (as one list at a given level is produced from multiple lists of the previous level). Its parameters are for each level, the size of the lists and the used merging algorithm. The total constraint correspond to the sum of the number of constrained bits at each level.

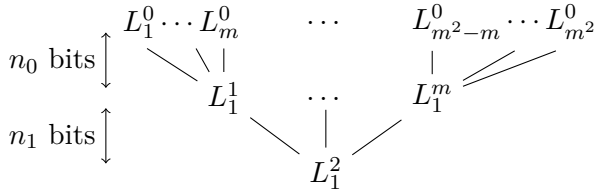


Figure 5.3: Merging tree structure. Each merging constrains n_i bits.

2-list merging. Wagner's algorithm [Wag02] uses 2-list merging. Hence, at each level, it merges two lists of size 2^{ℓ_i} into one list of size $2^{\ell_{i+1}}$, at a cost of 2^{ℓ_i} time and memory. At each level, the constraint is on $2\ell_i - \ell_{i+1}$ bits. The final list can be of size 1. Hence, with q levels, one can constrain $\ell_0 + \sum_{i=0}^{q-1} \ell_i$ bits. The total number of lists is 2^q. The memory cost is $\sum_i 2^{\ell_i}$, as the lists have to be stored (we consider here that the initial lists are put in memory only when they are needed). The time cost is in $\mathcal{O}\left(\sum_i 2^{\ell_i} 2^{q-i}\right)$, as a merging at level i has to be done twice per list at level $i+1$.

Hence, to minimize the memory cost, the list size shall be fixed, and to minimize the time cost, the list size shall double between each level. All in all, if we begin by doubling the list sizes from 2^{ℓ_0} until we reach a maximal size of 2^m, in e steps, the time cost will be $2^t = 2^{\ell_0+q}$, and we can constrain n bits, with

$$t - q - \frac{1}{2}(t - m - q)^2 + mq = n.$$

4-list merging. The previous approach can be adapted if we see 2 levels of 2-list merging as one level of 4-list merging. In that case, we can use the Schroeppel-Shamir algorithm. The time cost is squared, the memory cost and the number of lists is not affected, and the constraint is now on $4\ell_i - \ell_{i+1}$ instead of $2\ell_i + \ell_{i+1} - \ell_{i+2}$.

Hence, with the same approach as before (that is, doubling the list size at each step), we obtain

$$\frac{t-q}{3} - \frac{1}{2}\left(\frac{t}{2} - m - \frac{q}{2}\right)^2 + mq = \frac{2n}{3}$$

5.3 The easy instances

There are two cases where the hidden shift problem is especially simple, and can be solved in polynomial time.

5.3.1 Case of $(\mathbb{Z}/(2))^n$

If the group is $\mathbb{Z}/(2)$, then the hidden shift is in fact a hidden period, and Simon's algorithm can be used. Indeed, we can consider the function

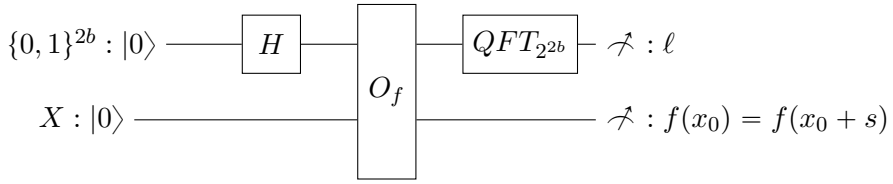
$$F(b, x) = \begin{cases} f(x) \text{ if } b = 0 \\ g(x) \text{ if } b = 1 \end{cases}.$$

This function satisfies $F(0,x) = f(x) = g(x \oplus s) = F(1, x \oplus s)$ and $F(1,x) = g(x) = f(x \oplus s) = F(0, x \oplus s)$. Hence, F is periodic, of period $(1,s)$, and Simon's algorithm can be applied.

5.3.2 Case $f = g$

If the two functions are the same, then this is also a hidden period problem. This case is tackled with Shor's algorithm [Sho94], which is extremely similar to Simon's algorithm.

We suppose to have access to a function $f : \mathbb{Z} \to X$, such that $f(x+s) = f(x)$. Moreover, we suppose that we know a bound on s, that is, $s \leqslant 2^b$. The period can be easily found with Circuit 5.1.



Circuit 5.1: Shor's circuit

This circuit corresponds to Algorithm 5.1. The amplitude of each value is variable. With a high probability, we will measure an ℓ such that $\ell \frac{s}{2^n}$ is very close to an integer, as shown in Lemma 5.1. Assuming we have measured such an ℓ, we can obtain the value of s with a good probability, from Proposition 5.1.

Lemma 5.1 (Shor's measurement, from [Sho94])**.** *A measurement of a state of the form*

$$\frac{1}{\sqrt{2^n \alpha}} \sum_{\ell=0}^{2^n-1} \sum_{r=0}^{\alpha-1} \exp\left(2i\pi r \ell \theta\right) |\ell\rangle$$

with $\alpha \leqslant 2^n$ has an outcome ℓ_0 such that $|\theta \ell_0 \mod 1| \leqslant \frac{1}{2\alpha}$ with probability greater than $\frac{4\alpha}{\pi^2 2^n}$, assuming such ℓ_0 exists.

Proof. The probability of an outcome ℓ_0 is

$$\left| \frac{1}{\sqrt{2^n \alpha}} \sum_{r=0}^{\alpha-1} \exp\left(2i\pi r \ell \theta\right) \right|^2$$

As this is a geometric sum, it reduces to

$$\left| \frac{1}{\sqrt{2^n \alpha}} \frac{1 - \exp\left(2i\pi\alpha\ell\theta\right)}{1 - \exp\left(2i\pi\ell\theta\right)} \right|^2 = \left| \frac{1}{\sqrt{2^n \alpha}} \frac{\sin\left(\pi\alpha\ell\theta\right)}{\sin\left(\pi\ell\theta\right)} \right|^2$$

The expression is decreasing while $0 \leqslant \alpha\ell\theta \leqslant \frac{1}{2}$, from $\frac{\alpha}{2^n}$ at the limit of θ at 0 to $\frac{1}{2^n \alpha \sin\left(\frac{\pi}{2\alpha}\right)^2}$ when $|\alpha\theta\ell_0| = \frac{1}{2}$. As $\sin x \leqslant x$ when $x \geqslant 0$, this latter term is greater than $\frac{1}{2^n \alpha \left(\frac{\pi}{2\alpha}\right)^2}$, which gives the expected bound. The situation is symmetric for $\frac{-1}{2} \leqslant \alpha\ell\theta \leqslant 0$, hence the lemma holds. □

Algorithm 5.1 Quantum state evolution in Circuit 5.1

Input: $O_F : |x\rangle|0\rangle \mapsto |x\rangle|f(x)\rangle$ with $f(x) = f(x+s)$, b
Output: ℓ

1: Initialize two registers : $|0\rangle|0\rangle$
2: Apply H gates on the first register, to compute

$$\frac{1}{\sqrt{2^{2b}}} \sum_{x=0}^{2^{2b}-1} |x\rangle|0\rangle$$

3: Apply O_f, to compute

$$\frac{1}{\sqrt{2^{2b}}} \sum_{x=0}^{2^{2b}-1} |x\rangle|f(x)\rangle$$

4: Measure an $f(x_0)$ in the last register, to get a

$$\frac{1}{\sqrt{\alpha}} \sum_{j=0}^{\alpha-1} |x_0 + js\rangle \quad \alpha = \left\lfloor \frac{2^{2b} - x_0}{s} \right\rfloor$$

5: Apply a QFT on the first register, to compute

$$\frac{1}{\sqrt{\alpha 2^{2b}}} \sum_{j=0}^{\alpha-1} \sum_{\ell=0}^{2^{2b}-1} \exp\left(2i\pi \frac{(x+js)\ell}{2^{2b}}\right) |\ell\rangle$$

6: The state is

$$\frac{1}{\sqrt{\alpha 2^{2b}}} \sum_{y=0}^{2^{2b}-1} \exp\left(2i\pi \frac{xy}{2^{2b}}\right) \sum_{\ell=0}^{\alpha-1} \exp\left(2i\pi \frac{js\ell}{2^{2b}}\right) |\ell\rangle$$

7: Measure the state, return ℓ.

Proposition 5.1 (Shor's period finding [Sho94])**.** *The knowledge of an ℓ such that $|\frac{\ell s}{2^{2b}} \bmod 1| \leqslant \frac{1}{2\alpha}$ allows to recover s with probability $\Omega(1/\log\log s)$.*

Proof. There exist an integer c such that

$$\left|\frac{\ell s}{2^{2b}} - c\right| \leqslant \frac{1}{2\alpha}$$
$$\Leftrightarrow \left|\frac{\ell}{2^{2b}} - \frac{c}{s}\right| \leqslant \frac{1}{2\alpha s}$$

We have that $\alpha \geqslant \frac{2^{2b}-s}{s} - 1$, hence $\alpha s > 2^{2b} - 2s > (2^b - 1)^2$. Hence, there is at most one fraction of the form $\frac{c}{s}$ with $s < 2^b$ that can satisfy the constraint. Such fraction can be found by computing the continued fraction expansion of $\frac{\ell}{2^{2b}}$. This will allow to recover s if $c \wedge s = 1$. This occurs with a probability in $\Omega\left(1/\log\log s\right)$. □

Theorem 5.1 (Period finding success probability)**.** *Circuit 5.1 allows to recover s with probability $\Omega\left(1/\log\log s\right)$.*

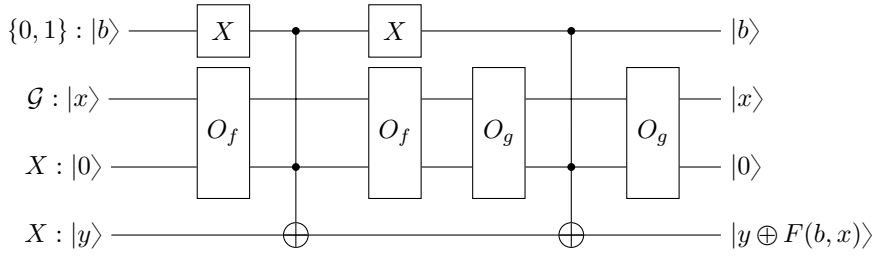
Proof. From Lemma 5.1, we measure a given ℓ such that $\left|\frac{\ell s}{2^{2b}} \mod 1\right| \leqslant \frac{1}{2\alpha}$ with a probability greater than $\frac{4\alpha}{\pi^2 2^{2b}}$. There are 2^{2b} values for ℓ, among which one over α will satisfy the condition. Hence, there are $\frac{2^{2b}}{\alpha}$ compatible ℓ. Overall, the probability to measure any ℓ that satisfy the condition is greater than $\frac{4}{\pi^2}$. From such an ℓ, we can recover s with a probability in $\Omega\left(1/\log\log s\right)$, with Proposition 5.1. □

Proposition 5.2 (General case [ME98])**.** *Let $f : \mathbb{Z}^k \to X$ be a function such that $f(x) = f(x+s)$ for $s \in \mathcal{L}$. A basis of the lattice $\mathcal{L}$ can be recovered in polynomial time.*

Remark 5.2. This algorithm allows to recover the group structure of the domain of a function, which can for example be used for a hidden shift problem in which the exact domain of the shifted functions is not given.

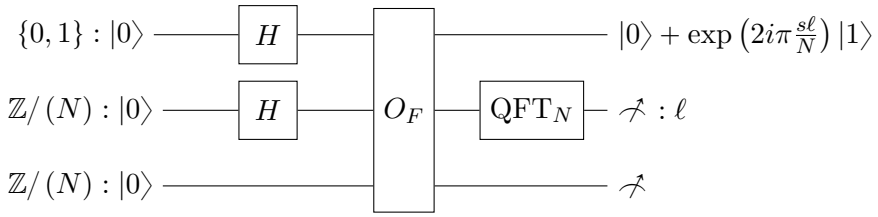
5.4 The generation algorithm

The hidden shift algorithms all work with qubits of a specific form, produced from the oracle by the generation algorithm (Circuit 5.3). It uses the function $F : \{0,1\} \times \mathcal{G} \to X$, defined as $F(0,x) = f(x)$, and $F(1,x) = g(x)$. From oracles to f and g, we can construct an oracle to F, as presented in Circuit 5.2. For clarity, we detail the algorithm in the case $\mathcal{G} = \mathbb{Z}/(N)$, but it can be naturally generalized to a product of cyclic groups, and therefore to any finite abelian group.



Circuit 5.2: Quantum oracle to F from oracles to f and g

The generation circuit (Circuit 5.3) is very close to Simon's circuit (Circuit 4.1), the two differences are that a Quantum Fourier Transform is used instead of a Hadamard gate after the oracle, and that the output is not a tuple of classical values but a qubit plus some classical values. The different steps are detailed in Algorithm 5.2.



Circuit 5.3: Generation circuit

Algorithm 5.2 Quantum state evolution in Circuit 5.3

Input: N, $O_F : |b\rangle|x\rangle|0\rangle \mapsto |b\rangle|x\rangle|F(b,x)\rangle$ with $F(0,x) = F(1,x+s)$
Output: $\ell, |\psi_\ell\rangle = |0\rangle + \exp\left(2i\pi\frac{s\ell}{N}\right)|1\rangle$

1: Initialize a 1-bit register and two n-bit registers : $|0\rangle|0\rangle|0\rangle$
2: Apply H gates on the two first registers, to compute $\sum_{b=0}^{1}\sum_{x=0}^{2^n-1}|b\rangle|x\rangle|0\rangle$
3: Apply O_F, to compute $\sum_{b=0}^{1}\sum_{x=0}^{2^n-1}|b\rangle|x\rangle|F(b,x)\rangle$
4: Measure an $F(0,x_0) = F(1,x_1)$ in the last register, to get $|0\rangle|x_0\rangle + |1\rangle|x_0+s\rangle$
5: Apply a QFT on the second register, to compute

$$\sum_{\ell=0}^{N-1}\exp\left(2i\pi\frac{x_0\ell}{N}\right)|0\rangle|\ell\rangle + \sum_{\ell=0}^{N-1}\exp\left(2i\pi\frac{(x_0+s)\ell}{N}\right)|1\rangle|\ell\rangle$$

6: The state is

$$\sum_{\ell=0}^{N-1}\exp\left(2i\pi\frac{x_0\ell}{N}\right)\left(|0\rangle + \exp\left(2i\pi\frac{s\ell}{N}\right)|1\rangle\right)|\ell\rangle$$

7: Measure ℓ in the second register, return it and the qubit $|0\rangle + \exp\left(2i\pi\frac{s\ell}{N}\right)|1\rangle$.

At step 6, the amplitude of a given ℓ is

$$\left||0\rangle + \exp\left(2i\pi\frac{s\ell}{N}\right)|1\rangle\right| \sqrt{\sum_{y=0}^{N-1}\left||0\rangle + \exp\left(2i\pi\frac{s\ell}{N}\right)|1\rangle\right|^2}^{-1} = \frac{1}{\sqrt{N}}.$$

Hence, the measured ℓ is a uniformly random integer between 0 and $N-1$.

The circuit of Circuit 5.3 produces a *phase element* $|\psi_\ell\rangle = |0\rangle + \exp\left(2i\pi\frac{s\ell}{N}\right)|1\rangle$, with a classically known, uniformly random label ℓ.

Complexity. As the generation algorithm contains only simple operations beyond the quantum oracle to F, we consider that its cost is the cost of one query to F. The memory is at least $2\log_2(N)+1$ qubits, but it can be more depending on the implementation of the quantum oracle.

5.5 Quantum query complexity

One of the firsts results on hidden shift problems is due to Ettinger and Høyer [EH99]. It states that, in the cyclic case, its quantum query complexity is in $\mathcal{O}(n)$.

Theorem 5.2 (Quantum query complexity)**.** *The quantum query complexity of the cyclic hidden shift is in* $\mathcal{O}(n)$.

Lemma 5.2 (Hoeffding's inequality [Hoe63])**.** *Let $X_1, \dots, X_n$ be n independent random variables and a, b, such that, for all i, $Pr[a \leqslant X_i \leqslant b] = 1$. Let $S = \sum_{i=1}^{n} X_i$. Then*

$$Pr[S - E[S] \geqslant k] \leqslant \exp\left(\frac{-2k^2}{n(b-a)^2}\right)$$

Proof of Theorem 5.2, derived from [EH99]. We consider the qubits produced by the circuit of Circuit 5.3, $|0\rangle + \omega^{s\ell} |1\rangle$. If we apply a Hadamard gate on them, we obtain

$$\omega^{\frac{s\ell}{2}} \cos\left(i\pi\frac{s\ell}{N}\right)|0\rangle - \omega^{-\frac{s\ell}{2}} \sin\left(i\pi\frac{s\ell}{N}\right)|1\rangle$$

Hence, given a particular ℓ, the probability of measuring a 0 is $\cos^2\left(i\pi\frac{s\ell}{N}\right)$, and 1 is measured with probability $\sin^2\left(i\pi\frac{s\ell}{N}\right)$.

This means we can sample a tuple (b, ℓ) according to the distribution X

$$\Pr\left[X = (b,\ell)\right] = \begin{cases} \frac{1}{N}\cos^2\left(i\pi\frac{s\ell}{N}\right) & \text{if } b = 0 \\ \frac{1}{N}\sin^2\left(i\pi\frac{s\ell}{N}\right) & \text{if } b = 1 \end{cases}$$

Now, let

$$f_t(b,x) = \begin{cases} \cos\left(2i\pi\frac{xt}{N}\right) & \text{if } b = 0 \\ -\cos\left(2i\pi\frac{xt}{N}\right) & \text{if } b = 1 \end{cases}$$

Then,

$$E[f_t(X)] = \begin{cases} 1 & \text{if } t = s = 0 \\ 0.5 & \text{if } t = s \text{ or } t = N - s \\ 0 & \text{otherwise} \end{cases}$$

We can now estimate the probability that $\sum_{i=1}^{n} f_t(X_i) \geqslant \frac{n}{4}$. From Lemma 5.2, if $t \notin \{0, s, N-s\}$, this probability is lower than $\exp(-m/32)$.

Hence, we can distinguish f_s and f_{N-s} from the other f_t, and identify s (or $N - s$) by computing f_t for all the possible $t \in [0; N/2]$.

In order to distinguish, we need that all the sums satisfy the inequality. By the union bound, this happens with a probability greater than $1 - N/2\exp(-m/32)$. The algorithm succeeds with probability greater than $1 - \epsilon$ if $m \geqslant 32\log(N/(2\epsilon))$. □

Remark 5.3 (Tighter bound). The explicit bound is made tigher in [BCD06], with a number of queries in $\log_2(N) + \mathcal{O}(1)$ and a matching lower bound.

These results show that fairly few queries are required to be able to uniquely identify the shift. Unfortunately, the corresponding algorithms to effectively recover the shift are not efficient. The next sections present algorithms which need more queries, but cost much less time.

5.6 Hidden shift modulo a power of 2

The simplest case to describe the hidden shift algorithms is when the group is $\mathbb{Z}/(2^n)$. This section presents the three known approaches applicable in this case.

5.6.1 Recovering the shift

A simple approach to recover the shift is proposed in [Kup05], which uses the fact that the phase element with label 2^{n-1} is

$$|\psi_{2^{n-1}}\rangle = |0\rangle + (-1)^s |1\rangle$$

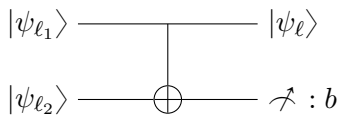
Hence, the least significant bit of s, s_0, can be recovered from this element if we measure it in the basis $\{|+\rangle, |-\rangle\}$. Once s_0 is known, we can reapply the same algorithm on a smaller instance: if we had $f(x) = g(x+s)$ in $\mathbb{Z}/(2^n)$, we can consider $f'(x) = f(2x)$ and $g'(x) = g(2x + s_0)$. These two functions fulfill $f'(x) = g'(x + s')$ in $\mathbb{Z}/\left(2^{n-1}\right)$, with $s = 2s' + s_0$, and allow to recover s_1. We can recover the complete shift s by solving smaller and smaller instances of the problem. The algorithms of this section offer different approaches to construct $|\psi_{2^{n-1}}\rangle$ from some random $|\psi_\ell\rangle$.

5.6.2 Kuperberg's first algorithm: $\pm$

5.6.2.1 Combination

Kuperberg's first algorithm [Kup05] uses the very simple combination circuit of Circuit 5.4. This circuit takes two phase elements $|\psi_{\ell_1}\rangle, |\psi_{\ell_2}\rangle$ and produces

$|\psi_\ell\rangle$ and b, with $\ell = \ell_1 \pm \ell_2$, that is, the output phase element label is either the sum or the difference of the input labels. Both outcomes are equiprobable.



Circuit 5.4: First combination circuit

5.6.2.2 Algorithm

This ability to produce either the sum or the difference of two chosen labels can be leveraged to obtain a subexponential algorithm, by converging to a given value. The idea is that given a large amount L_u of labels that are multiples of 2^u, we can find many pairs of labels such that either their sum or difference will satisfy a constraint on $\log_2(L_u)$ bits: it will be a multiple $2^{u+\lfloor \log_2(L_u) \rfloor}$). If we apply recursively this idea to converge to $|\psi_{2^{n-1}}\rangle$, we obtain Algorithm 5.3.

5.6.2.3 Complexity analysis

Theorem 5.3 (Complexity of Algorithm 5.3 [Kup05, Theorem 5.1])**.** *Algorithm 5.3 can recover s in $\widetilde{O}\left(2^{\sqrt{2\log_2(3)n}}\right)$ quantum queries, classical and quantum time, and classical and quantum memory.*

Algorithm 5.3 A first hidden shift algorithm, from [Kup05, Proof of Theorem 5.1]

Generate $\widetilde{O}\left(2^{\sqrt{2\log_2(3)n}}\right)$ phase elements in L ▷ Queries
$e \leftarrow \sqrt{2n/\log_2(3)}$
$k \leftarrow \sqrt{2n\log_2(3)}$
For $i := 1$ to $\lceil e \rceil$ **do**
    $L' = \varnothing$
    While L contains two phase elements $|\psi_a\rangle, |\psi_b\rangle$ such that $2^{\lfloor k \rfloor} | (a-b)$ **do**
        Pop $(|\psi_a\rangle, |\psi_b\rangle)$ from L
        Combine $|\psi_a\rangle, |\psi_b\rangle$ into $|\psi_c\rangle$ ▷ Combination
        If $c = a - b$ **then**
            Insert $|\psi_c\rangle$ into L'
        Else
            Insert $|\psi_c\rangle$ back into L ▷ $c = a + b$
    $L \leftarrow L'$
    $k \leftarrow k + \sqrt{2n\log_2(3)} - i\log_2(3)$
If $|\psi_{2^{n-1}}\rangle \in L$ **then**
    Perform a measurement on $|\psi_{2^{n-1}}\rangle$
    Return s_0
Else
    Return Failure

Lemma 5.3 (Iterative combination)**.** *Let L_u be a set of $e2^k$ phase elements whose labels are all multiples of 2^u. By using the combination of Circuit 5.4, one can construct a set L_{u+k} of phase elements of size $(e-1)2^k/3$ whose labels are all multiples of 2^{u+k}.*

Proof. We consider the subsets L_i of L_u, with $L_i = \{x \in L_u | x \mod 2^{u+k} = i2^u\}$. Any two elements $(x, y) \subset L_i$ are such that $x - y = 0 \mod 2^{u+k}$. Hence, we can obtain an element of L_{u+k} by taking any two element that belong to the same L_i, and combine them, with probability one half. If the combination failed, we obtain a new phase element that belongs to L, as $2^a | x$ and $2^a | y$ implies $2^a | (x+y)$.

We can combine any two elements if they belong to the same L_i, which means that at most 2^k elements cannot be combined, and will be lost.

Hence, from $e2^k$ elements in L, we can obtain $(e2^k - 2^k)/4$ elements in L' and $(e2^k - 2^k)/4 + 2^k$ elements still in L. The total number of elements in L' is then $\sum_i (e2^k - 2^k)/4^i \simeq (e-1)2^k/3$. □

Proof of Theorem 5.3. Let L be a set of size $(e+1)2^{e\log_2(3)}$ phase elements. By Lemma 5.3, we can obtain a set of size $e2^{(e-1)\log_2(3)}$ that are multiples of $2^{\lfloor e\log_2(3) \rfloor}$.

Hence, by induction, we can obtain multiples of 2^{n-1} if we have $\sum_{i=1}^{e} \lfloor i\log_2(3) \rfloor \geqslant n-1$.

This implies $e \simeq \sqrt{\frac{2n}{\log_2(3)}}$. Hence, the query cost is in $\widetilde{O}\left(2^{\sqrt{2\log_2(3)n}}\right)$. The quantum time cost consists only in the combination, which reduces the total number of phase elements by 1. Hence, it is also in $\widetilde{O}\left(2^{\sqrt{2\log_2(3)n}}\right)$. The classical part of the algorithm

have to split the labels in subsets according to their value modulo a power of 2, which can be done in classical time and memory linear in the number of elements to split, which are also in $\widetilde{O}\left(2^{\sqrt{2\log_2(3)n}}\right)$. □

5.6.3 Regev's subset-sum variant

The previous algorithm had the notable drawback that it needs a large amount of quantum memory, and all its algorithmic efficiency comes from the fact that we can choose interesting pairs of values among the ones we have. Regev [Reg04] has proposed to use a different combination mechanism to use only a polynomial amount of phase element at a time. The method was later generalized by Childs, Jao and Soukharev [CJS14], and we were able to improve the algorithm in [Bon19b].

This algorithm is slightly slower than the previous one. For its complexity estimates, we define $L(\alpha, c)$ and $L(c)$ as

$$L(\alpha, c) = 2^{(c+o(1))n^{\alpha}\log(n)^{1-\alpha}} \quad L(c) = L(1/2, c).$$

5.6.3.1 Combination routine

The idea is to take a certain amount of phase elements (k), and use them to produce one better phase element (here, a multiple of a larger power of 2). Recall that the elements are of the form $|0\rangle + \exp\left(2i\pi\frac{s\ell_i}{N}\right)|1\rangle$. If we tensor them, we obtain

$$\bigotimes_i |\psi_{\ell_i}\rangle = \sum_{j\in\{0,1\}^k} \exp\left(2i\pi\frac{j\cdot(\ell_1,\ldots,\ell_k)}{2^n}\right)|j\rangle.$$

From this superposition, one can compute the function

$$|j\rangle|0\rangle \mapsto |j\rangle|j\cdot(\ell_1,\ldots,\ell_k) \mod 2^r\rangle$$

and measure the second register. By definition, the remaining j have a phase identical modulo 2^r. Hence, if we can retain exactly two different values j_1 and j_2, we have a superposition with a phase difference which will be a multiple of 2^r. Next, the only remaining step is to change the superposition of $|j_1\rangle$ and $|j_2\rangle$ to a superposition of $|0\rangle$ and $|1\rangle$. The complete routine is Algorithm 5.4.

The projection (Algorithm 5.4, step 5) can be implemented by Algorithm 5.5, which either projects successfully on a superposition of two given values j_1 and j_2, or projects on the superposition of all the other possible values. With an even number of possible values, this method is guaranteed to succeed, while with an odd number we may fail to obtain a pair.

Success probability. In order to succeed, we need that at least two j are solutions of the equation $j\cdot(\ell_1,\ldots,\ell_k) \mod 2^{r+a} = V$ and that the projection succeeded. As there are 2^k possible values for J and 2^r possible values for V, there are at most 2^r values of j that correspond to a unique solution. When there are at least 2 solutions,

Algorithm 5.4 Regev's combination routine [Reg04]

Input: $(|\psi_{\ell_1}\rangle, \ldots, |\psi_{\ell_k}\rangle) : \forall i, 2^a | \ell_i,\ r$
Output: $|\psi_{\ell'}\rangle,\ 2^{r+a} | \ell'$

1: Tensor $\bigotimes_i |\psi_{\ell_i}\rangle = \sum_{j\in\{0,1\}^k} \exp\left(2i\pi \frac{j\cdot(\ell_1,\ldots,\ell_k)}{2^n}\right)|j\rangle$
2: Add an ancilla register, apply $|x\rangle|0\rangle \mapsto |x\rangle|x\cdot(\ell_1,\ldots,\ell_k) \mod 2^{r+a}\rangle$
3: Measure the ancilla register, leaving with

$$V \text{ and } \sum_{j\cdot(\ell_1,\ldots,\ell_k) \atop \mod 2^{r+a}=V} \exp\left(2i\pi \frac{j\cdot(\ell_1,\ldots,\ell_k)}{2^n}\right)|j\rangle$$

4: Compute the corresponding j
5: Pair them, project to a pair (j_1, j_2).
The register is now $\exp\left(2i\pi \frac{j_1\cdot(\ell_1,\ldots,\ell_k)}{2^n}\right)|j_1\rangle + \exp\left(2i\pi \frac{j_2\cdot(\ell_1,\ldots,\ell_k)}{2^n}\right)|j_2\rangle$
6: Map $|j_1\rangle$ to $|0\rangle$, $|j_2\rangle$ to $|1\rangle$
7: Return $|0\rangle + \exp\left(2i\pi \frac{(j_2-j_1)\cdot(\ell_1,\ldots,\ell_k)}{2^n}\right)|1\rangle$

Algorithm 5.5 Projection routine

Input: $\sum_{x\in J}\phi(x)|x\rangle$, $(j_1, j_2) \subset J$.
Output: $\phi(j_1)|j_1\rangle + \phi(j_2)|j_2\rangle$ or $\sum_{x\in J\setminus\{j_1,j_2\}}\phi(x)|x\rangle$
1: Add an ancilla qubit: $\sum_{x\in J}\phi(x)|x\rangle|0\rangle$
2: Apply the operator $|x\rangle|0\rangle \mapsto |x\rangle|x = j_1 \vee x = j_2\rangle$
3: Measure the ancilla qubit

the projection fails if there is an odd number of solutions and we fail to project on a pair. This occurs with probability at most 1/3. Overall the success probability of the combination routine is at least of $1 - \frac{2}{3}\left(1 - 2^{r-k}\right)$.

Remark 5.4. This success probability depends on the values of r and k. In [Reg04], $k = r + 4$ is used. In [CJS14], $k = r + 1$ is used.

5.6.3.2 Complete algorithm

The complete algorithm uses Algorithm 5.4 multiple times, with a pipeline structure: one routine takes as input a given number of phase elements, and produces one element, which will be given as input to the next routine, as in Figure 5.4. This approach has the advantage of allowing an on-the fly computation: if at any point a routine has access to its inputs, it can process them and produce a refined phase element. This allows to never have more than n phase elements at any given point in the algorithm.

Childs-Jao-Soukharev algorithm [CJS14, Appendix A]. If we want each routine in the pipeline to be the same, we will have m routines, and we need $mr \simeq n$ in order to succeed. If each routine combines k elements and succeed with probability p, the total cost is then of $(k/p)^m$ queries, km qubits (excluding the quantum oracle overhead),

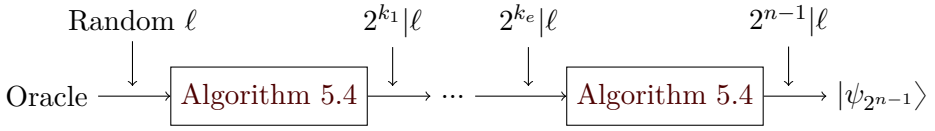


Figure 5.4: Pipeline of combinations

a classical time in $\widetilde{O}(\sum_{i<m}(k/p)^i 2^k) = \widetilde{O}((k/p)^m 2^k)$ and a polynomial classical memory. If $k = \alpha\sqrt{n\log_2(n)}$, the query cost is $L\left(1/(2\alpha)\right)$, and the time cost is $L(1/(2\alpha) + \alpha)$. The classical cost is minimized for $\alpha = 1/\sqrt{2}$, which implies $k = \sqrt{n\log_2(n)/(2)}$, which leads to a quantum query and time cost of $L(1/\sqrt{2})$ and a classical time cost of $L(\sqrt{2})$.

Remark 5.5. The quantum query exponent of [CJS14, Theorem 5.2] corresponds to the quantum time exponent, and is not tight.

We now present the multiple tradeoffs we proposed in [Bon19b].

A better tradeoff. Each combination has the same cost in the previous algorithm. This means that most of the cost is concentrated in the first level, as this is the level we're iterating the most. We can improve the algorithm by increasing k for the later levels. The level i is performed k/p times more than the level $i+1$, which means we can afford to have the level $i+1$ to cost k/p times more than the level i. As $k = \mathcal{O}(\sqrt{n\log(n)})$, we can increase k by $\log(n)/2$ at each level without changing the time exponent.

Theorem 5.4 (Classical/quantum tradeoff)**.** *The hidden shift problem can be solved in $L(\sqrt{\beta^2+1})$ classical time and $L(-\beta+\sqrt{\beta^2+1})$ quantum queries and time for any $\beta \geqslant 0$.*

Proof. We can take $k_i = \beta\sqrt{n\log(n)} + i\log(n)/2$. In that case, we have

$$n = \sum_{i=0}^{m-1} k_i \simeq m\beta\sqrt{n\log(n)} + \frac{m^2}{4}\log(n)$$

$$\Rightarrow m \simeq 2\left(-\beta+\sqrt{\beta^2+1}\right)\sqrt{n/\log(n)}$$

The classical time cost correspond to the cost of the last level, as each level has the same cost. Its logarithm is in $k_m \simeq \sqrt{\beta^2+1}\sqrt{n\log(n)}$. The query cost corresponds to the number of time we're doing the first level. As each level has the same cost, its logarithm is $k_{m-1} - k_0 \simeq m\log(n)/2 \simeq \left(-\beta+\sqrt{\beta^2+1}\right)\sqrt{n\log(n)}$.

□

Example 5.1. The minimum cost is achieved with $\beta = 0$, which leads to a cost in $L(1)$ quantum queries, quantum and classical time. If we want to have a quadratic gap between the classical and quantum time, we can take $\beta = \frac{1}{\sqrt{3}}$, and obtain a classical time cost in $L\left(\frac{2}{\sqrt{3}}\right)$ and a quantum query and time cost in $L\left(\frac{1}{\sqrt{3}}\right)$.

Tradeoffs for lower quantum cost. Lower quantum cost can be achieved if we reduce the number of steps. In that case, the gains obtained by increasing the size for later steps become negligible.

Theorem 5.5 (Classical/quantum tradeoff with low quantum cost)**.** *The hidden shift problem can be solved in $L(\alpha, \beta)$ classical time and $L(1-\alpha, \alpha/\beta)$ quantum queries and time for any $\alpha \in (1/2; 1]$).*

Proof. We can take $k = \beta n^\alpha \log(n)^{1-\alpha}$. In this case, we have $m \simeq 1/\beta n^{1-\alpha} \log(n)^{\alpha-1}$, and $m \log k \simeq \alpha/\beta n^{1-\alpha} \log(n)^\alpha$. This leads to a quantum query and time cost in $L(1-\alpha, \alpha/\beta)$ and a classical time cost in $L(\alpha, \beta)$. □

At the limit, if we consider a fixed number of combination steps, then the number of queries required is polynomial, while the processing time has to be exponential.

Theorem 5.6 (Classical/quantum tradeoff with polynomial quantum cost)**.** *The hidden shift problem can be solved in $\mathcal{O}\left(2^{n/\alpha}\right)$ classical time and $\mathcal{O}(n^\alpha)$ quantum queries and time for any $\alpha \geqslant 1$.*

Proof. Use α combination steps, each of size n/α. □

5.6.3.3 Subset-sum algorithms

The combination routine requires (Algorithm 5.4, step 4) to compute efficiently all the solutions of the equation $j \cdot (\ell_1, \ldots, \ell_k) = V$. This corresponds to a subset-sum problem, and algorithms better than brute-force can be used. A summary of these algorithms is presented in Table 5.1.

All these algorithms have a cost in $\mathcal{O}(2^{cn})$, with $c \leqslant 1$. We can use any of them to replace the exhaustive search in Algorithm 5.4, step 4. This leads to a direct improvement of all the previous complexities, but the most time-efficient algorithm add a cost in memory. In particular, memory-heavy algorithm for the subset-sum are less likely to be relevant here, as Kuperberg's algorithms have a better asymptotic complexity. Moreover, the improvement of Theorem 5.4 induces a larger memory cost than the original tradeoff, as the final instances are bigger. The tradeoffs are summarized in Table 5.2.

5.6.4 Kuperberg's second algorithm: k-list

Kuperberg's second algorithm [Kup13] is the most time-efficient algorithm to date. It can be seen as a generalization of Regev's variant, with a less strict definition of what we consider to be elements.

5.6.4.1 Phase vectors

The previous algorithms used elements of the form

$$|\psi_\ell\rangle = |0\rangle + \exp\left(2i\pi\frac{s\ell}{N}\right)|1\rangle.$$

Table 5.2: Summary of the tradeoffs for Regev's variant. c is the exponent of the subset-sum algorithm used as a subroutine.

Classical Time	Classical Memory	Quantum query/time	Quantum Memory	subset-sum	Algorithm
$L(\sqrt{2})$	$\mathcal{O}(n)$	$L(1/\sqrt{2})$	$\mathcal{O}(n)$	Ex. search	[CJS14]
$L(\sqrt{\beta^2+c})$	Variable	$L(-\beta+\sqrt{\beta^2+c})$	Variable	Variable	Theorem 5.4
$L(\alpha, c\beta)$	Variable	$L(1-\alpha, \alpha/\beta)$	Variable	Variable	Theorem 5.5
$\mathcal{O}\left(2^{cn/\alpha}\right)$	Variable	$\mathcal{O}(n^\alpha)$	Variable	Variable	Theorem 5.6
$L\left(1\right)$	$\mathcal{O}(n)$	$L\left(1\right)$	$\mathcal{O}(n)$	Ex. search	Theorem 5.4
$L\left(2/\sqrt{3}\right)$	$\mathcal{O}(n)$	$L\left(1/\sqrt{3}\right)$	$\mathcal{O}(n)$	Ex. search	Theorem 5.4
$L(0.763)$	$L(0.381)$	$L(0.381)$	$\mathcal{O}(n)$	[BCJ11]	[CJS14]
$L(0.539)$	$L(0.539)$	$L(0.539)$	$\mathcal{O}(n)$	[BCJ11]	Theorem 5.4
$L(0.312)$	$L(0.623)$	$L(0.623)$	$\mathcal{O}(n)$	[BCJ11]	Theorem 5.4
$L(0.849)$	$L(0.849)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	[BCJ11]	Theorem 5.4
$L(0.490)$	$L(0.980)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	[BCJ11]	Theorem 5.4

We can see them as a particular case of

$$\left|\psi_{(\ell_0,\ldots,\ell_M)}\right\rangle = \sum_{i=0}^{M} \exp\left(2i\pi\frac{s\ell_i}{N}\right)|i\rangle,$$

which contains $\lceil\log(M)\rceil$ qubits, and is indexed by the M values $(\ell_0,\ldots,\ell_M)$.

In that case, the vector is defined up to the termwise addition of any value, as only the phase difference is meaningful (in other words, we can impose that $\ell_0 = 0$).

5.6.4.2 Combination routine

Regev's combination routine can be naturally adapted to phase vectors. From

$$\left|\psi_{(\ell_0,\ldots,\ell_M)}\right\rangle\left|\psi_{(\ell'_0,\ldots,\ell'_M)}\right\rangle = \sum_{i,j=0}^{M} \exp\left(2i\pi\frac{s(\ell_i+\ell'_j)}{N}\right)|i\rangle|j\rangle,$$

we can compute

$$\sum_{i,j=1}^{M} \exp\left(2i\pi\frac{s(\ell_i+\ell'_j)}{N}\right)|i\rangle|j\rangle\left|\ell_i+\ell'_j \mod 2^k\right\rangle,$$

and, as before, measure the sum to get a value V. We will obtain a state containining the qubits $|i\rangle|j\rangle$ for $\ell_i+\ell'_j \mod 2^k = V$. In practice, we take $k \simeq \log(M)$, to obtain a vector whose termwise differences will always be a multiple of 2^k, and which will contain roughly M elements. This is called *collimation*. Now, if we want to have a compact phase vector that uses only $\log(M)$ qubits, we need to relabel the $|i\rangle|j\rangle$, that is, choose a mapping $f : \left\{(i,j)|\ell_i+\ell'_j \mod 2^k = V\right\} \to [0;M]$ and transform the state

$$\sum_{i,j:\ell_i+\ell'_j=V} \exp\left(2i\pi\frac{s(\ell_i+\ell'_j)}{N}\right)|i\rangle|j\rangle$$

into

$$\sum_{i,j:\ell_i+\ell'_j=V} \exp\left(2i\pi\frac{s(\ell_i+\ell'_j)}{N}\right)|f(i,j)\rangle\,.$$

This state is also a phase vector, associated to the vector containing the value $(\ell_i+\ell'_j)$ at index $f(i,j)$. Moreover, the difference between all the values will always be a multiple of 2^k.

Algorithm 5.6 Kuperberg's second combination routine [Kup13]

Input: $\left(\left|\psi_{(\ell_0,\ldots,\ell_M)}\right\rangle, \left|\psi_{(\ell'_0,\ldots,\ell'_M)}\right\rangle\right) : \forall i, 2^a|\ell_i, 2^a|\ell'_i,\ r$

Output: $\left|\psi_{(v_0,\ldots,v_{M'})}\right\rangle : \forall i, 2^{r+a}|v_i$

1: Tensor $\left|\psi_{(\ell_0,\ldots,\ell_M)}\right\rangle\left|\psi_{(\ell'_0,\ldots,\ell'_M)}\right\rangle = \sum_{i,j=0}^{M}\exp\left(2i\pi\frac{s(\ell_i+\ell'_j)}{N}\right)|i\rangle|j\rangle$

2: Add an ancilla register, apply $|i\rangle|j\rangle|0\rangle \mapsto |i\rangle|j\rangle\left|\ell_i+\ell'_j \mod 2^{r+a}\right\rangle$

3: Measure the ancilla register, leaving with

$$V \text{ and } \sum_{i,j:\ell_i+\ell'_j \mod 2^{r+a}=V} \exp\left(2i\pi\frac{s(\ell_i+\ell'_j)}{N}\right)|i\rangle|j\rangle$$

4: Compute the $M'+1$ corresponding i,j

5: Apply to the state a transformation f from (i,j) to $[0;M']$.

6: Return the state and the vector v with $v_{f(i,j)} = \ell_i + \ell'_j$.

Combination cost. We consider here that $M' \simeq M$. The classical time cost comes from the computation of the labels i,j such that $\ell_i + \ell'_j \mod 2^{r+a} = V$. This can be done efficiently by sorting the two lists, hence the cost is in $\mathcal{O}(M\log(M))$. The classical memory cost is only the storage cost of the lists, in $\mathcal{O}(M)$. The quantum time cost is the cost of computing the sum in superposition, and then relabeling. The sum can be computed in $\mathcal{O}(M)$ operations, by adding each classically-known vector component, one at a time. The relabeling can be done by computing, for each (i,j), $f(i,j)$, and then, for each $f(i,j)$, erasing the corresponding (i,j). Overall, this is in $\mathcal{O}(M)$.

While the previous algorithm used the number of phase elements to obtain a better element, it is here the size of the vectors which allows to obtain a better vector. Moreover, we have an additional quantum time cost which was not in the previous approach.

5.6.4.3 Complete algorithm

The algorithm, as Regev's variant, applies recursively the combination routine. The combination of a vector of 2^a elements allows to gain b bits, and produces a vector of size 2^{a-b}. As with Regev's variant, the combination routine for level $i+1$ can cost more than the combination routine for level i. As each combination takes 2 elements and produces 1, it can cost twice more. However, this induces a higher memory cost, as

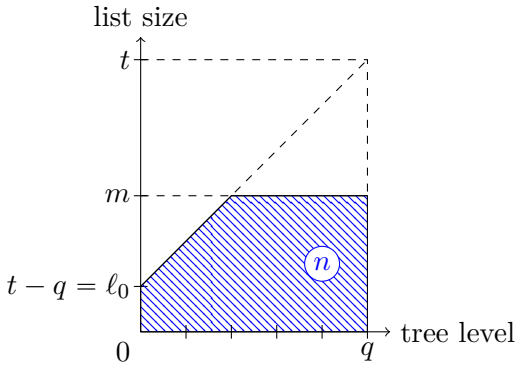


Figure 5.5: Size of the lists in function of the tree level, in $\log_2$ scale, annotated with the different parameters.

larger elements require to have a larger list in classical memory. Finally, in the end, we only need to have a vector of size 2. Hence, the algorithm will have 3 parts: while some classical memory is available, increase the length of the vectors, once it is saturated, use a fixed length, and in the end, project the vector on a pair. This final step allows to gain $b - 1$ bits if the vector is of size 2^b.

Initial vectors. We can take directly the phase elements as phase vectors of size two to begin. However, we can also take b phase elements, and produce a phase vector of size 2^b, simply by tensoring them. Indeed,

$$\bigotimes_i |\psi_{\ell_i}\rangle = \sum_{j\in\{0,1\}^k} \exp\left(2i\pi \frac{j\cdot(\ell_1,\ldots,\ell_k)}{2^n}\right)|j\rangle\,,$$

and the index j has the label $j \cdot (\ell_1, \ldots, \ell_k)$.

Complexity. We consider that the algorithm has e levels, and the initial phase vectors are of size 2^b. The time cost will be in $\widetilde{O}\left(2^{b+e}\right)$, as the first step is performed 2^e times, the second 2^{e-1} times, and so on.

If we omit polynomial factors, we can denote the classical and quantum time as 2^t, the available memory as 2^m memory and the number of quantum queries as 2^q. With these notations, the list size for each level is distributed as in Figure 5.5. We have q equal the number of levels, and t equal the number of levels plus ℓ_0. As each level constrains as many bits as the log of its list size, the total amount of bits constrained by the algorithm corresponds to the hatched area.

Hence, with $\max(m, q) \leqslant t \leqslant m + q$, we can solve the hidden shift problem with

$$-\frac{1}{2}(t - m - q)^2 + mq = n$$

Kuperberg's algorithm [Kup13]. If we consider $t = m = q$, we directly obtain the cost of $\widetilde{O}\left(2^{\sqrt{2n}}\right)$ from [Kup13].

Table 5.3: Summary of the tradeoffs for Kuperberg's second algorithm

Classical Time	Classical Memory	Quantum query	Quantum Time	Note
$\mathcal{O}\left(2^t\right)$	$\mathcal{O}\left(2^m\right)$	$\mathcal{O}\left(2^q\right)$	$\mathcal{O}\left(2^t\right)$	$-\frac{1}{2}(t-m-q)^2+mq=n$
$\mathcal{O}\left(2^{\sqrt{2n}}\right)$	$\mathcal{O}\left(2^{\sqrt{2n}}\right)$	$\mathcal{O}\left(2^{\sqrt{2n}}\right)$	$\mathcal{O}\left(2^{\sqrt{2n}}\right)$	
$\mathcal{O}\left(2^{2\sqrt{n}}\right)$	$\mathcal{O}\left(2^{\sqrt{n}}\right)$	$\mathcal{O}\left(2^{\sqrt{n}}\right)$	$\mathcal{O}\left(2^{2\sqrt{n}}\right)$	
$\mathcal{O}\left(2^{2t}\right)$	$\mathcal{O}\left(2^m\right)$	$\mathcal{O}\left(2^q\right)$	$\mathcal{O}\left(2^t\right)$	$-\frac{1}{2}(t-m-q)^2+mq=\frac{2n}{3}$
$\mathcal{O}\left(2^{4\sqrt{\frac{n}{3}}}\right)$	$\mathcal{O}\left(2^{2\sqrt{\frac{n}{3}}}\right)$	$\mathcal{O}\left(2^{2\sqrt{\frac{n}{3}}}\right)$	$\mathcal{O}\left(2^{2\sqrt{\frac{n}{3}}}\right)$	
$\mathcal{O}\left(2^{4\sqrt{\frac{2n}{3}}}\right)$	$\mathcal{O}\left(2^{\sqrt{\frac{2n}{3}}}\right)$	$\mathcal{O}\left(2^{\sqrt{\frac{2n}{3}}}\right)$	$\mathcal{O}\left(2^{2\sqrt{\frac{2n}{3}}}\right)$	

Classical/Quantum Tradeoffs. The previous approach had the inconvenient of using equal classical and quantum times, up to polynomial factors. In practice, we can expect to be allowed more classical operations than quantum gates. We can obtain different tradeoffs by reusing the previous 2-list merging tree, and seeing it as a 2^k-list merging tree. That is, we see k levels as one, and merge the 2^k lists at once. This allows to use the Schroeppel-Shamir algorithm for merging, with a classical time of $2^{2^k/2}$ and a classical memory of $2^{2^k/4}$. This operation is purely classical, as we are computing lists of labels, and it does not impact the quantum cost. Moreover, while we used to have a constraint on $\log(k)m$ bits, we now have a constraint on $(k-1)m$ bits.

For $k=2$, omitting polynomial factors, with a classical time of 2^{2t} and quantum time of 2^t, a memory of 2^m, a number of quantum queries of 2^q and $\max(m,q)\leqslant t\leqslant m+q$, we can solve the hidden shift problem with

$$-\frac{1}{2}(t-m-q)^2+mq=2n/3 \quad .$$

The general tradeoffs for $k=1$ and $k=2$ are presented in Table 5.3.

5.7 General hidden shift algorithms

This section shows how to generalize the previous algorithms in different cases. The optimization of Algorithm 5.3, its simulation, the variant for $(\mathbb{Z}/(2^p))^w$, the discussion on the variants of the promise and the extension to nonabelian groups have been proposed in [BN18]. The simulation of Algorithm 5.8 for $\mathbb{Z}/(N)$ have been proposed in [BS18].

5.7.1 Optimizing Algorithm 5.3

It is possible to relax the constraints on the set of labels we want, as we can associate, in this case, a bit of s to a given label that allows to recover that bit.

For example, the phase element $|\psi_{2^{n-1}}\rangle=|0\rangle+(-1)^s|1\rangle$ can be associated with s_0, the parity bit of s, as a measure of $|\psi_{2^{n-1}}\rangle$ in the Hadamard basis has outcome s_0. The

phase element $|\psi_{2^{n-2}}\rangle = |0\rangle + (-1)^{(s-s_0)/2} \exp\left(i\pi\frac{s_0}{2}\right)|1\rangle$ can be associated with s_1, as if s_0 is known the term $\exp\left(i\pi\frac{s_0}{2}\right)$ can be erased by applying a phase shift gate of angle $-\pi\frac{s_0}{2}$. We have the same property with $|\psi_{2^{n-1}+2^{n-2}}\rangle$, which is associated with s_1, and has to be corrected by a phase shift of $-\pi\frac{3s_0}{2}$.

In general, the phase element $|\psi_{\alpha 2^{n-i+1}+2^{n-i}}\rangle$ can be associated with s_{i-1}, and requires a phase correction of angle $-\pi\frac{(2\alpha+1)s}{2^{i-1}}$, which only depends on the bits $s_0 \dots s_{i-2}$ of s. Hence, we do not need to have a specific set of n labels to recover s, but only one odd label, one label multiple of 2, one multiple of 4, and so on.

Algorithm 5.3 wastes many phase elements at each step, that we cannot combine, and only seeks for specific collisions. A simpler and slightly more efficient approach is, given a set of elements multiple of 2^a, to look for the pair that may lead to a multiple of the largest possible multiple of 2. This opportunistic variant of the algorithm corresponds to Algorithm 5.7.

Algorithm 5.7 Opportunistic variant, in base 2 [Kup05, Algorithm 3]

Generate a sufficiently large number N of phase elements ▷ Queries
Separate them in pools P_i of elements whose label is divisible by 2^i and not 2^{i+1}
For $i := 0$ to $n-2$ **do**
    While $|P_i| \geqslant 2$ **do**
        Pop two elements $(|\psi_a\rangle, |\psi_b\rangle)$ of P_i such that $a+b$ or $a-b$ has the highest possible divisibility by 2 (and is not 0)
        Combine $|\psi_a\rangle, |\psi_b\rangle$ ▷ Combination
        Insert the resulting $|\psi_c\rangle$ in the corresponding pool P_j
        If $P_{n-1} \neq \varnothing$ **then** ▷ Found $|\psi_{2^{n-1}}\rangle$?
            Perform a measurement on $|\psi_{2^{n-1}}\rangle$
            Return s_0
Return Failure

The previous algorithm focuses on s_0, but it turns out that any phase element in P_i can be associated with the bit s_{n-i-1}. Hence, as we've shown in [BN18], we can recover all the bits of s if we ensure that all the P_i are non-empty. This is done in Algorithm 5.8.

Other variants. The two other hidden shift algorithms compute on-the-fly, and do not waste anything after producing $|\psi_{2^{n-1}}\rangle$, which makes this optimization less appealing for them.

5.7.1.1 Concrete cost estimates of Algorithm 5.8

Algorithm 5.8 is a quantum algorithm that can easily be simulated classically, as all the logic is classical computations on the labels. As the initial phase elements have a label uniform in $\mathbb{Z}/(2^n)$, we can replace them with a uniformly random value in $\mathbb{Z}/(2^n)$, and as the outcome of the combination routine is either the sum or the difference of the two input labels, the combination can be replaced with an unbiaised coin flip.

Algorithm 5.8 Variant to obtain all the needed elements at once [BN18]

Generate a sufficiently large number N of phase elements ▷ Queries
Separate them in pools P_i of elements whose label is divisible by 2^i and not 2^{i+1}
For $i := 0$ to $n-2$ **do**
  While $|P_i| \geqslant 3$ **do**
    Pop two elements $(|\psi_a\rangle, |\psi_b\rangle)$ of P_i such that $a+b$ or $a-b$ has the highest possible divisibility by 2 (and is not 0)
    Combine $|\psi_a\rangle, |\psi_b\rangle$ ▷ Combination
    Insert the resulting $|\psi_c\rangle$ in the corresponding pool P_j
    If $\forall i \in [0, n-1], P_i \neq \varnothing$ **then**
      Perform a measurement on a qubit in each pool
      Return s
Return Failure

Table 5.4: Simulation results for Algorithm 5.8 for 90% success probability [BN18]

n	queries	$\log_2$(queries)	$1.8\sqrt{n} - 0.5$	trials
16	118	6.9	6.7	10^6
32	826	9.7	9.7	10^6
64	14975	13.9	13.9	5×10^5
80	49200	15.6	15.6	10^5
128	9.8×10^5	19.9	19.9	5×10^4

We used a radix tree structure to efficiently find the elements such that their sum or differences have the highest possible divisibility by 2. Table 5.4 summarizes the results of our simulations. We obtained that Algorithm 5.8 requires roughly $2^{1.8\sqrt{n}-0.5}$ queries. As $\sqrt{2\log_2(3)} \simeq 1.8$, this matches the asymptotic complexity estimation of Theorem 5.3, with a small constant instead of a polynomial overhead.

5.7.2 Hidden shift in $\mathbb{Z}/(N)$

If we are not working modulo a power of 2, we need slightly more work to recover the shift. The general idea is the same, that is, recovering qubits whose label is a power of 2.

Proposition 5.3. *Let $n = \lceil \log_2(N) \rceil$. From the phase elements labeled 2^j, for $0 \leqslant j < n$, it is possible to recover s with a probability greater than $\frac{4}{\pi^2}$.*

Proof. Let's consider the tensored state of the phase elements

$$\frac{1}{2^{n/2}} \bigotimes_{j=0}^{n-1} |0\rangle + \exp\left(2i\pi \frac{2^i s}{N}\right) |1\rangle = \frac{1}{2^{n/2}} \sum_{x=0}^{2^n-1} \exp\left(2i\pi \frac{xs}{N}\right) |x\rangle$$

If $N = 2^n$, this is exactly the QFT of s, hence applying the inverse QFT produces s with probability 1.

Table 5.5: Simulation results for Algorithm 5.8 modulo N, for 90% success

$\log_2(N)$	$\log_2(Q)$	$1.8\sqrt{\log_2(N)} + 2.3$
20	10.1	10.3
32	12.4	12.5
50	15.1	15.0
64	16.7	16.7
80	18.4	18.4
100	20.3	20.3

Otherwise, if we apply an inverse QFT modulo 2^n on this state, we obtain

$$\frac{1}{2^n} \sum_{y=0}^{2^n-1} \sum_{x=0}^{2^n-1} \exp\left(2i\pi x \left(\frac{s}{N} - \frac{y}{2^n}\right)\right) |y\rangle$$

From Lemma 5.1, a measurement of this state will produce a value y such that $2^n \left(\frac{s}{N} - \frac{y}{2^n}\right) < \frac{1}{2}$ with a probability greater than $\frac{4}{\pi^2}$. Such a y always exists and uniquely defines s if $n > \log_2(N)$. □

Hence, we still want to produce labels corresponding to powers of 2. However, there is no longer an incentive to produce multiples of powers of 2, as this does not correspond to a subgroup. There are two approaches to produce a given label:

- **Focusing on the label 1 [Kup05].** Using the same combination techniques, be can seek smaller values instead of multiples of a power of 2. For Kuperberg's first algorithm, the combination circuit is the same. For Regev's variant and Kuperberg's second algorithm, we only need to measure $\lfloor j \cdot (\ell_1, \ldots, \ell_k)/2^r \rceil$ instead of $j \cdot (\ell_1, \ldots, \ell_k) \mod 2^r$.

- **Applying the previous method [BS18].** If we apply the previous method, we will obtain a label $\ell = 2^{n-1} \mod 2^n$. There is a chance that the equality holds in $\mathbb{Z}$, that is, $\ell = 2^{n-1}$. In that case, the produced label can be used to recover s. We have simulated this approach for Algorithm 5.8, the results are presented in Table 5.5.

Recovering the other powers of 2 [Kup05]. The approach to get a small label allows in fact to obtain any power of 2. Indeed, if we want to recover the value v for v invertible, we can multiply all the labels by v^{-1} and compute the label 1, which will correspond to v. If $N = 2^e N'$, then we need to mix the two approaches: using the first approach to produce labels multiples of 2^e, and then the second approach to produce any given multiple of 2^e.

5.7.3 Hidden shift in abelian groups

The general method for abelians group is the same, except that the addition used in the computation is the one of the group. If the group is not cyclic, then there are multiple independent components, but the same approaches can be used, at a cost roughly corresponding to the cost of the algorithm for a cyclic group of the same size.

5.7.3.1 Case of $(\mathbb{Z}/(2^w))^p$

As we've shown in [BN18], the case of parallel additions modulo a power of 2 is in fact easier. We can see it as an intermediate problem between Simon's problem ($w = 1$) and the cyclic hidden shift problem ($p = 1$).

Target labels. The elements whose label satisfies $\ell_j \in \{0, 2^{w-1}\}$ are of the form $|\psi_{\ell_1,\dots,\ell_p}\rangle = |0\rangle + \exp(i\pi \sum s_j \ell_j)|1\rangle$, so measuring them in the $\{|-\rangle, |+\rangle\}$ basis will give us the parity of $\sum s_j \ell_j$, that is, a linear equation in the parity bits of the s_j. In the case $w = 1$, we get a variant of Simon's algorithm for hidden shifts.

Recovering the complete shift. Once all the parity bits are known, we can recover the next bits if we have a set of p elements whose labels are multiple of 2^{w-2} and that form a full-rank system of equations, and so on.

New approach. We can use the parallel structure of the hidden subgroup: given $p+1$ random labels, we can find a subset whose sum (or difference) will always be even on all the components: if we look at the parity vector of the elements, this corresponds to a linearly dependent subset of the vectors. This approach can be useful if p is big with respect to the size of the pools: with on average $p/2+1$ vectors, we can zero p bits. We can then iterate the technique to set to zero the next row of bits, and so on. This is described in Algorithm 5.9.

Lemma 5.4 (Equation cost)**.** *An iteration of the outer for loop of Algorithm 5.9 produces one element with p zeroed bits using on average $(p/2+1)$ elements, and needs p qubits.*

Proof. A step of Algorithm 5.9 uses random equations to produce a zeroed element. If we have p elements that form a basis of $\mathbb{Z}/(2)^p$, any other element is a linear combination of $p/2$ elements, on average, in this basis. If we have a basis, we can hence get an equation that has, on average $p/2+1$ elements, and that sums to zero on the p bits. We can then construct such a basis by choosing p random elements : if they form a free family, we have a basis, if not, we then have some elements that sum to zero. □

Theorem 5.7. *Algorithm 5.9 has a complexity in quantum queries and time of around $2(p/2+1)^w$, and a classical time in $\mathcal{O}((p/2+1)^{w+3})$. It needs $2p(w-1)$ quantum memory, plus the oracle cost.*

Algorithm 5.9 Algorithm for parallel additions [BN18]

```
System = [∅; w]
While System[w − 1] has not full rank do
    Query |ψ_{ℓ_1...,ℓ_p}⟩.
    v ← (ℓ_1 ..., ℓ_p)
    i ← 0
    While v ≠ (0, ..., 0) and i < w − 1 do
        If v is in the span of System[i] then
            There exists S ⊂ System[i] such that v = ⊕_{w∈S} w
            Apply Circuit 5.4 on v and the elements of S
            Compute the corresponding label (ℓ'_1 ..., ℓ'_p)
            v ← (ℓ'_1 ..., ℓ'_p)
        Else
            Add v to System[i]
            v ← (0, ..., 0)
From the values in System[w − 1], recover the parity bits of each component of the shift.
```

Proof. We want p elements in $\texttt{System}[w-1]$ (with only p elements, as they would be random, the success probability is only of $1/e$, but we can get arbitrarily close to 1 with a fixed overhead). We can obtain 1 element in $\texttt{System}[i+1]$ from $(p/2+1)$ elements in $\texttt{System}[i]$. The total cost is then of

$$p(p/2+1)^{w-1} + p(p/2+1)^{w-2} + \cdots + p,$$

which reduces to $2(p/2+1)^w$. The total cost in quantum memory is then $p(w-1)$ qubits, for the $w-1$ steps. One combination requires to solve a linear system of equations in dimension p, which costs $\mathcal{O}(p^3) = \mathcal{O}((p/2+1)^3)$ classical time. □

Comparison with the generic approaches. This algorithm can be seen as a particular case of Regev's variant. Indeed, the subset-sum problem modulo 2 is easy, and corresponds to solving a system of linear equations. Its scaling in function of p and w is different, as the cost of the generic approach depends on $\sqrt{pw}$ while the cost of this algorithm depends on $w\log(p)$.

Remark 5.6 (Case of interest). Algorithm 5.9 is more efficient than Kuperberg's second algorithm when $w < \frac{p}{2\log(p/2+1)^2}$.

5.7.4 Combining the different algorithms

All the hidden shift algorithms presented in this chapter work with phase elements, and the time-efficient algorithms combine them to obtain new elements with a more interesting label. This means we can mix the different approaches in the same algorithm. In particular, Algorithm 5.9 can be used to reduce the cost of solving the hidden shift in $(\mathbb{Z}/(2^w))^p$ even for large w. Even if this does not allow to reduce the best

asymptotic exponents, the different approaches have a different classical and quantum time/memory/query tradeoffs, and a combined algorithm may be more interesting, depending on the set of resources available.

5.7.5 Variants on the promise

This section studies different situations, when the promise of the hidden shift is not exactly fullfilled, as we did with Simon's algorithm in Section 4.2.

Lemma 5.5 (Unwanted collisions)**.** *Let $f : \mathbb{Z}/(N) \to \mathbb{Z}/(N)$ be a random function, $s \in \mathbb{Z}/(N)$, g such that $g(x) = f(x+s)$. Given a quantum oracle access to f and g, we can retrieve s in Q quantum queries if we can solve the hidden shift problem in $\mathbb{Z}/(N)$ with a permutation using Q/e quantum queries.*

Proof. We need to decompose the steps of the generation algorithm (Algorithm 5.2) . The measurement of the third register of

$$\sum_x |0\rangle |x\rangle |f(x)\rangle + |1\rangle |x\rangle |g(x)\rangle$$

produces

$$|0\rangle \sum_{j=1}^{c} |x_j\rangle + |1\rangle \sum_{j=1}^{c} |x_j + s\rangle$$

and the measurement yields $f(x_j)$ with probability $c/2^n$. After the QFT, the measurement will give us a label ℓ and a qubit

$$\left(\sum_{j=1}^{c} \exp\left(2i\pi \frac{x_j \ell}{2^n}\right)\right)\left(|0\rangle + \exp\left(2i\pi \frac{s\ell}{2^n}\right)|1\rangle\right)$$

As a qubit is invariant by a global phase shift, we still get a valid element. However, it is not uniformly sampled, and the probability of getting a given ℓ is

$$p = \frac{1}{c2^n}\left|\sum_{j=1}^{c} \exp\left(2i\pi \frac{x_j \ell}{2^n}\right)\right|^2.$$

Notably, the case $\ell = 0$, which is useless for us, is the most probable.

It is known [FO90] that for a random function, the expected number of images with r preimages is $2^n/e/r!$. The first measurement samples on the images, uniformly if it is a bijection, and proportionally to the number of preimages in the general case. That means we'll have a probability of $r/e/r! = 1/e/(r-1)!$ of getting an image with r preimages. We'll get a unique preimage with probability $1/e$, so that means with e times the number of samples, we'll get enough elements with only one preimage. This is a very rough approximation, as the multiple preimages induces only a bias on the generated elements. □

Remark 5.7. Alternatively, we can consider the function $F(x) = (f(x), f(x+1), \dots)$, that has the same shifts as f, but has a smaller probability of unwanted collisions, at the cost of having to query f multiple times for one query of F.

Multiple shifts. If we have multiple shifts, that is, $f(x) = g(x+s) = g(x+t)$, then $t-s$ is a hidden period of f (and g), which can be recovered with Shor's algorithm.

The following lemma addresses the problem of functions which respect the shift promise only for a subset of their inputs, and shows this is still resolvable if the number of wrong inputs is small enough.

Lemma 5.6 (Partial shift)**.** *Let f, g two permutations of $\mathbb{Z}/(N)$, $s \in \mathbb{Z}/(N)$, $X \subset \mathbb{Z}/(N)$ such that, for all $x \in X$, $f(x) = g(x+s)$. Then if the hidden subgroup problem in $\mathbb{Z}/(N)$ costs Q queries, we can retrieve s given quantum oracle access to f and g in Q queries, with probability $(|X|/N)^Q$.*

Proof. If we measure an $f(x)$ whose x is in X, then we have a valid element. This happens with probability $|X|/N$. If this is not the case, we get a malformed qubit. We can expect the algorithm to succeed only if all the Q queried elements are valid, which happens with probability $(|X|/N)^Q$. □

The following lemma is used to attack Poly1305 with quantum queries in Section 7.7.3.

Lemma 5.7 (Input restriction)**.** *Let f, g be two permutations of $\mathbb{Z}/(N)$, $s \in \mathbb{Z}/(N)$ such that, for all x, $f(x) = g(x+s)$. Given a quantum oracle access to f and g restricted to the inputs $0 \leqslant x < 2^n$, if $0 \leqslant s < 2^{n-1}$ and the hidden subgroup problem in $\mathbb{Z}/\left(2^{n-1}\right)$ can be solved in Q queries, s can be retrieved in eQ^2 queries.*

Proof. We are only given access to the interval $[0; 2^n)$. We cannot see the hidden shift in $\mathbb{Z}/(N)$ as a hidden shift in $\mathbb{Z}/(2^n)$. However, if s is small enough, we have an instance of a partial hidden shift, the valid elements being the ones such that $0 \leqslant x < 2^n$ and $0 \leqslant x + s < 2^n$. The probability to get a bad element is less than $s/2^n$ in this case. If we need Q queries, and $s/2^n \simeq 1/Q$, then the success probability will be greater than $(1-1/Q)^Q \simeq 1/e$. This fails for greater s.

However, we can query a subinterval of $[0; 2^n)$ for f and g. For $A \in [0; 2^{n-1})$, if we query $[0; 2^{n-1})$ to $f(x)$ and $g(x+A)$, we will retrieve s with probability $1/e$ if $0 \leqslant s - A < 2^{n-1}/Q'$, if we need Q' queries to solve the hidden subgroup problem in $\mathbb{Z}/\left(2^{n-1}\right)$.

To retrieve s, we can sequentially test for all A multiples of $2^{n-1}/Q'$, until we reach 2^{n-1}. We then have Q' intervals to test, and each test costs Q' queries. Moreover, the algorithm will succeed if the test with the right guess of A succeeds, and can be verified with a few classical queries. As the right guess has a success probability greater than $1/e$, we expect to find the shift in eQ'^2 queries. □

5.7.6 Hidden shift in nonabelian groups

The algorithms presented in this chapter can't be used with non-abelian groups. However, these groups contain some abelian subgroups (as for example the iterated powers of an element). We can apply the algorithm on such a subgroup, and it will succeed if the hidden shift lies in this subgroup. The idea is then to look for this situation. It can

be done by considering the right cosets of the abelian subgroup $\mathcal{A}$. Indeed, all group elements can be uniquely written as ar, with $a \in \mathcal{A}$ and r a fixed representative of a right coset. The hidden shift can be decomposed as $s = s_a s_r$, and with $f(xs) = g(x)$, the relation $f(xs_a s_r) = g(x)$ can be seen as $f'(xs_a) = g(x)$, which is an instance of the hidden subgroup problem in $\mathcal{A}$. The complete algorithm is then to do a Grover search on the right coset, and then try to solve the problem in $\mathcal{A}$, as presented in Algorithm 5.10.

As the hidden shift is a joint property of the two functions, we cannot do a collision search as in the generic case, and need an exhaustive search. This procedure can also be used to solve the hidden period problem, as this is the case $f = g$. Hence, we can upper bound the hardness of the generic hidden shift problem in function of the size of the group (around 2^n) and the size of its maximal abelian subgroup (around 2^a), which would be around $2^{(n-a)/2+\sqrt{2a}}$. Such bound is not always interesting, as the generic quantum collision in $2^{n/3}$ can also be applied.

Algorithm 5.10 Generic resolution of a hidden shift problem in a nonabelian group

$\mathcal{A} \leftarrow$ A commutative subgroup of $\mathcal{G}$
$\mathcal{C} \leftarrow$ {A representative of each right coset}
procedure GroverSearch($c \in \mathcal{C}$) ▷ c is assigned in quantum superposition
  $s \leftarrow$ Kuperberg's algorithm result in $\mathcal{A}$ for $f(xc)$ and $g(x)$
  If $f(xsc) = g(x)$ for a few random x **then**
    Mark c
    ▷ c is now the representative of the coset of the shift
$s \leftarrow$ Kuperberg's algorithm result in $\mathcal{A}$ for $f(xc)$ and $g(x)$
Return sc

Examples. $GL_2(q)$ contains $(q^2-q)(q^2-1)$ elements, some of which of order q^2-1. If we consider the group generated by such an element, we'll have $\mathcal{A}$ of size around $q^2 \simeq 2^{n/2}$. This means the complexity of the HSP in $GL_2(q)$ is at most around $q2^{2\sqrt{\log_2(q)}} = 2^{n/4+\sqrt{n}}$. For $\mathcal{S}_n$, we can consider the subgroup generated by the 2-cycles $\{(2i-1, 2i) | 1 \leqslant i \leqslant n/2\}$. As all the cycles are disjoint, this is an abelian group of size $2^{n/2}$, isomorphic to $(\mathbb{Z}/(2))^{n/2}$. This allows the use of Simon's algorithm to find the hidden shift, with a total cost of around $n2^{-n/4}\sqrt{n!}$, for a group of size $n!$. This is however asymptotically worse than $\sqrt[3]{n!}$.

Chapter 6
Searching for a Hidden Structure

This chapter is devoted to quantum algorithms for problems more applied than mere hidden subgroup problems, where there exists a function that fulfills the property among many possible functions, and we want to identify both the function and the subgroup. This kind of problems are recurrent in cryptanalysis. A first algorithm that combined Grover and Simon's algorithm was proposed by Leander and May [LM17], with an application to attack the FX construction with superposition queries. With Akinori Hosoyamada, María Naya-Plasencia, Yu Sasaki and André Schrottenloher [Bon+19], we proposed a new version of the algorithm that only needs a polynomial number of queries (while the original algorithm was exponential in query), and showed that in some cases, it can be applied even if we only have access to a classical oracle. Our algorithm is the building block for the first example of quantum attacks that use Simon's algorithm but do not require a quantum oracle. In that sense, it also provides a partial answer to a question in quantum computing: "Is Simon's algorithm useful?". It can be seen either as an improved collision algorithm, or as a hidden period algorithm that does not need superposition queries. This chapter focuses on a search with Simon's algorithm (Chapter 4), but it can be applied with any quantum algorithm that needs to call a function on a fixed quantum superposition. Some applications of these algorithms are presented in Chapter 7 and in Chapter 9.

Contents

The general problem we want to tackle is the following:

Problem 6.1 (Search for a periodic function)**.** *Let $f \in \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^\ell$ be a function, such that there exists a unique $i_0 \in \{0,1\}^k$ such that $f(i_0, \cdot)$ fulfills the promise of Simon's algorithm. Given oracle access to f, find i_0 and the period of $f(i_0, \cdot)$.*

We will focus on algorithms that retrieve i_0, as once i_0 is known, the period of $f(i_0, \cdot)$ can be found in one application of Simon's algorithm.

6.1 Combining Grover's and Simon's algorithms

The algorithm proposed in [LM17] to solve Problem 6.1 is quite simple: do a quantum search on i with the test function "is $f(i,\cdot)$ periodic?". In order to do so, we need a measurement-free version of Simon's algorithm. This is not an issue, but it requires to solve a linear system of equations in quantum superposition. The only thing we need to define is the quantum circuit used in the amplitude amplification, and we need to estimate its success probability.

To simplify the analysis, we suppose that $f(i_0,x)$ is a 2-to-1 function, and that $f(i,x)$ for $i \neq i_0$ is a permutation. The analysis can be extended to any function using the methods of Section 4.2.2, that is, we only need to multiply the number of queries used by Simon's algorithm by a small constant.

Simon's algorithm as a test function. What we really want is to be able to distinguish between a periodic function and a random function. Simon's algorithm samples values orthogonal to the period. Hence, we first need to assume that the period is not 0. This is generally not an issue, as we can test separately this case, or simply dismiss it, as this edge case should occur with a negligible probability.

Algorithm 6.1 Grover-meets-Simon algorithm [LM17]

Input: An oracle to $f(i,x)$, such that $f(i_0,\cdot)$ has a period $s \neq 0$.
Output: i_0

Filter i
  Apply Simon's algorithm to $f(i,\cdot)$.
  Get the period s.
  If $s = 0$: **Abort**
End Filter

Theorem 6.1 (Cost of Algorithm 6.1)**.** *Let $f \in \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^\ell$ be a function, $i_0 \in \{0,1\}^k$ such that:*

- *There exists a unique $s \neq 0$ such that $[f(i_0,x) = f(i_0,y) \Leftrightarrow x \oplus y \in \{0,s\}]$.*
- *For all $i \neq i_0$, $f(i_0,\cdot)$ is injective.*

Algorithm 6.1 finds i_0 with probability greater than $1-2^{-\alpha}-2^{-k}-2^{\alpha-k}$ using at most $\frac{\pi}{2}2^{k/2}$ applications of Simon's algorithm if each application uses $n+\alpha+k$ superposition queries to f.

Proof. We use Simon's algorithm to sample values orthogonal to the period. If the period exists, the rank of these values will be at most $n-1$, and if there is no period, we are sampling random values, which can be of rank n. This is our distinguisher between the two cases. The probability that a set m of random values is of rank less than n is lower than 2^{n-m} (Proposition 4.1). Hence, each of the 2^k-1 wrong values passes our test with probability 2^{n-m}, while the value we want passes it with probability

1. Amplitude amplification will produce the superposition of all the values that pass the test. Hence, we will measure the value we want with probability greater than $\frac{1}{1+(2^k-1)2^{n-m}} \geqslant 1 - 2^{k+n-m}$. The more false positives we have, the fewer amplification amplitude iterations we need to do, hence we have at most $\frac{\pi}{2}2^{k/2}$ iterations.

The test circuit of amplitude amplification has a success probability of $2^{-k} \times 1 + (2^k - 1)2^{-k} \times 2^{n-m} \simeq 2^{-k} + 2^{n-m}$. Hence, it succeeds with probability greater than $1 - 2^{-k} - 2^{n-m}$ in less than $\frac{\pi}{2}2^{k/2}$ applications of Simon's algorithm.

Overall, we obtain the value we want with a probability greater than

$$\left(1 - 2^{-k} - 2^{n-m}\right)\left(1 - 2^{k+n-m}\right) \geqslant 1 - 2^{-k} - 2^{n-m} - 2^{k+n-m}. \qquad \square$$

Remark 6.1 (Non-injective functions.). For non-injective functions, we can bound the number of queries using Proposition 4.5, and taking the maximal p_0 over all the $f(i, \cdot)$, which will only multiply the number of queries by a constant that depends on f.

Using an external test function. In some cases, we may be able to check directly if the index and the period are correct. This would allow us to check if the value found to have a period is correct or not. It allows to very efficiently remove the problem of false positives, and reduces the necessary number of queries in Simon's algorithm. Moreover, it does not require a special treatment of the case $s = 0$. We suppose that the cost of the test is comparable to a call to f, which allows to neglect it compared to the cost of Simon's algorithm.

Algorithm 6.2 Grover-meets-Simon with external test

Input: An oracle to $f(i, x)$, such that $f(i_0, \cdot)$ has a period $s \neq 0$, a test function for (i, s).
Output: i_0

Filter i
  Apply Simon's algorithm to $f(i, \cdot)$.
  Get the period s. ▷ If the rank is $n - 1$, take the non-zero period
  If (i, s) passes the external test **then**
    Return i
End Filter

The algorithm is the same as before, except that:

- We need to have a maximal rank for the correct guess (or to check all the values in the vector space),
- We can suppress any false positive with the external test.

In practice, this allows to directly use Simon's algorithm (and in particular Proposition 4.1 and 4.5) within an amplitude amplification (Theorem 3.2), as the probability that Simon's algorithm, given a uniformly random i, passes the test will be $2^{-k}(1 - 2^{-\alpha})$. As we do not have any false positive, we only need to amplify slightly more than in the

case of a deterministic test ($\sqrt{1-2^{-\alpha}}$ times more), to obtain the correct i with a probability greater than $1-2^{-k}$. Hence, the number of queries we need is unaffected by k. Overall, the total number of queries is $(n+\alpha)\frac{\pi}{2}2^{k/2}\frac{1}{\sqrt{1-2^{-\alpha}}}$.

Theorem 6.2 (Optimal number of queries for Algorithm 6.2)**.** *The total number of queries is minimized when* $\alpha \simeq \log\left(\frac{\ln(2)}{2}(n+\log(n))\right)$*, with a total number of queries around* $(n+\log(n))\frac{\pi}{2}2^{k/2}$*.*

Proof. The total number of queries is minimized when $\frac{n+\alpha}{\sqrt{1-2^{-\alpha}}}$ is minimized. Deriving it, we obtain that the cost is minimal if $2^{\alpha}-1=\frac{\ln(2)}{2}(n+\alpha)$. □

6.2 The offline Simon's algorithm

We devised the following algorithms, which are presented in [Bon+19].

6.2.1 A more structured problem

In cryptographic applications, the family of functions $f(i,\cdot)$ is often fairly structured: it depends on a unique secret keyed function E_k, that we query, and some additional computations. Moreover, the input space on which we query E_k is generally fixed: it is the uniform superposition of its inputs. Hence, the problem is more structured: we do not have a black-box access to a family of functions, we have a black-box access to a unique function, from which we construct the family of functions. This section shows how to reduce drastically the number of queries if we have a family of functions that can all be computed given the same superposition of inputs, that is, given $\sum_{i,x}|i\rangle|x\rangle|E_k(x)\rangle$, we can compute $\sum_{i,x}|i\rangle|x\rangle|f(i,x)\rangle$.

Problem 6.2 (Constructing and Finding a Hidden Period)**.** *Let* $E_k:\{0,1\}^n\to\{0,1\}^\ell$ *be a function,* $f:\{0,1\}^n\times\{0,1\}^k\to\{0,1\}^\ell$ *be a family of functions such that for all* i*, there exists a reversible mapping between the set* $\{(x,E_k(x))|x\in\{0,1\}^\ell\}$ *and* $\{(x,f(i,x))|x\in\{0,1\}^\ell\}$*. Assume that there exists a unique* $i_0\in\{0,1\}^k$ *such that* $f(i_0,\cdot)$ *satisfies Simon's promise. Given oracle access to* E_k*, find* i_0 *and the period of* $f(i_0,\cdot)$*.*

This notion of reversible mappings corresponds to any superposition computable reversibly given the superposition $\sum_x|x\rangle|E_k(x)\rangle$, which means we need to be able to compute reversibly, from $i,x,E_k(x)$, and $i,x',f(i,x')$, with a bijection between x and x'.

This problem catches a notion of constructibility of a family of functions given a *fixed* superposition of values. In practice, we can often state the instance we want to solve as a simpler problem:

Problem 6.3 (Asymmetric Search of a Period)**.** *Let* $F:\{0,1\}^m\times\{0,1\}^n\to\{0,1\}^\ell$ *and* $E_k:\{0,1\}^n\to\{0,1\}^\ell$ *be two functions. We consider* F *as a family of functions indexed by* $\{0,1\}^m$ *and write* $F(i,\cdot)=f_i(\cdot)$*. Assume that there exists a unique* $i_0\in\{0,1\}^m$ *such that* $f_{i_0}\oplus E_k$ *satisfies Simon's promise. Given oracle access to* F *and* E_k*, find* i_0*.*

This simpler version makes more explicit the kind of functions we are interested in: each function can be separated in two parts, E_k, which is fixed, and expected to be costly, and f_i, which depends on a parameter and cheaper to compute.

6.2.2 The offline Simon's algorithm

The idea to solve Problem 6.2 is to see Simon's algorithm slightly differently than usual. Instead of querying an oracle to a function, we suppose that the algorithm is given as input a *database* of E_k, a set of superpositions $\sum_x |x\rangle |E_k(x)\rangle$. It then computes the periodic function from this set, and finally extracts the period. We note $\left|\psi_{E_k}^m\right\rangle = \bigotimes_{j=1}^m \sum_x |x\rangle |E_k(x)\rangle$, a state which contains m copies of the superpositions of input/outputs of E_k.

Hence, the algorithm is very similar to Algorithm 6.1, but the test function fetches $\left|\psi_{E_k}^m\right\rangle$, uses it to check if the function is periodic, and finally uncomputes everything, to get back $\left|\psi_{E_k}^m\right\rangle$.

We want a circuit that tests if $i = i_0$. If the test function computed exactly that, as we would have a result independent from the database, we could get back exactly the original superposition, and apply the amplitude amplification theorems. This is unfortunately not the case, and each iteration will degrade the state $\left|\psi_{E_k}^m\right\rangle$. In order to make it work, we need to bound the error at each iteration.

As before, we suppose that $f(i_0, x)$ is a 2-to-1 function, and that $f(i, x)$ for $i \neq i_0$ is a permutation in the analysis.

Algorithm 6.3 The offline Simon's algorithm

Input: $|i\rangle$, a database $\left|\psi_{E_k}^m\right\rangle$
Output: $|b\rangle$ telling if $f(i, \cdot)$ is periodic, a quantum state close to the database

1: For each superposition in the database, compute $\sum_x |x\rangle |f(i, x)\rangle$.
2: Apply a Hadamard gate on the first register of each superposition.
3: Compute in superposition the rank r of the values in each first register.
4: $b \leftarrow r \neq n$
5: Uncompute everything but the value of b.
6: Return b and the database.

Proposition 6.1 (Precision of Algorithm 6.3)**.** *Algorithm 6.3 tests if $i = i_0$ and adds a noise of amplitude smaller than $2^{(n-m)/2+1}$.*

Proof. The ideal circuit we want takes $\sum_i |i\rangle \left|\psi_{E_k}^m\right\rangle |0\rangle$ as input, and produces the output $\sum_i |i\rangle \left|\psi_{E_k}^m\right\rangle |i = i_0\rangle$. This is however not exactly what we have. If $i = i_0$, then the answer is correct with probability 1, as the periodicity reduces the maximal rank, and it behaves like the ideal circuit.

For $i \neq i_0$, the circuit first constructs

$$|i\rangle \left(\bigotimes_{j=1}^m \sum_{x_j} |x_j\rangle |f(i, x_j)\rangle \right) |0\rangle \quad .$$

It then applies a Hadamard gate on $|x\rangle$, to obtain

$$|i\rangle \left(\bigotimes_{j=1}^{m} \sum_{x_j} \sum_{y_j} (-1)^{x_j \cdot y_j} |y_j\rangle |f(i,x_j)\rangle \right) |0\rangle \quad .$$

Once this is done, we can compute the rank of the m values y_j, and obtain

$$|i\rangle \left(\bigotimes_{j=1}^{m} \sum_{\substack{(y_1,\dots,y_m) \\ \text{of maximal rank}}} \sum_{x_j} (-1)^{x_j \cdot y_j} |y_j\rangle |f(i,x_j)\rangle \right) |0\rangle$$
$$+ |i\rangle \left(\bigotimes_{j=1}^{m} \sum_{\substack{(y_1,\dots,y_m) \\ \text{of lower rank}}} \sum_{x_j} (-1)^{x_j \cdot y_j} |y_j\rangle |f(i,x_j)\rangle \right) |1\rangle \quad .$$

The second term is the annoying one. The state deviates from the state we want by 2 times the amplitude of the second term, as the terms with the correct results are missing, and the terms with the wrong result have been added. Hence, the norm of the noise is 2 times the amplitude of this term. As the probability for m words of n bits to be of a lower rank than n is bounded by 2^{n-m} (Proposition 4.1), the amplitude of the second term is bounded by $2^{(n-m)/2}$. With the factor 2, we obtain $2^{(n-m)/2+1}$. □

We can directly plug this test algorithm into a quantum search, as in Algorithm 6.4.

Algorithm 6.4 Quantum search with offline Simon

Input: m, an oracle to E_k.
Output: i_0

1: Call m times the oracle on the uniform superposition, to make $\left|\psi_{E_k}^m\right\rangle$
2: **Filter** i
3: Call Algorithm 6.3 on $|i\rangle \left|\psi_{E_k}^m\right\rangle$
4: **If** a period is found **then**
5: **Return** i
6: **End Filter**

Theorem 6.3 (Complexity of Algorithm 6.4)**.** *Algorithm 6.4 finds i_0 in $\frac{\pi}{4}2^{k/2}$ applications of Algorithm 6.3, with probability greater than $1 - 2^{-k} - 2^{(k+n-m)/2+2}$.*

Proof. From Proposition 6.1, the amplitude of the noise is smaller than $2^{(n-m)/2+1}$. We need to perform $\frac{\pi}{4}2^{k/2}$ iterations of the test function, hence, by Theorem 3.3, the final result will be correct with a probability greater than

$$\left(1 - 2^{-k}\right) \left(1 - 2^{k/2} 2^{(n-m)/2+1}\right)^2 \geqslant 1 - 2^{-k} - 2^{(k+n-m)/2+2}. \qquad \square$$

With an external test function Previously, if we had access to an external test function, we could suppress the dependency in k in the required number of queries. With the offline variant, this is no longer the case, as a noise is added at each iteration. Hence, this alternative approach does not gain much here, and has essentially the same cost as Algorithm 6.4.

Non-injective functions. As with the direct application of Simon's algorithm, we can apply the same algorithm to distinguish between periodic and non-periodic functions, even if such functions are not respectively 2-to-1 and injective. We only need to multiply the number of queries by a small constant that depends on the functions. Details on how to estimate the constant are presented in Section 4.2.2.

6.3 Simon's algorithm with classical queries

The previous section showed how to reduce the query cost of a quantum search with Simon's algorithm by an exponential factor. If this reduction can offer significant gains when the oracle costs more than Simon's algorithm, it does not change the nature of the attack, which only works if a quantum oracle to the primitive is accessible. This section shows how to extend the attack when the oracle is only accessible classically. The core technique is quite simple: the database $\left|\psi_{E_k}^m\right\rangle$ can be constructed from 2^n classical queries and $m2^n$ quantum operations. With this approach, we can solve the search for a period when restricted to classical queries, and this allows in some cases to solve Simon's problem with classical queries.

Generating the database The database $\left|\psi_{E_k}^m\right\rangle$ is generated sequentially, one query at a time, as described in Algorithm 6.5. Interestingly, this algorithm can use each query on-the-fly, and does not need to store anything classically, as all the information is encoded in the database.

Algorithm 6.5 Generation of $\left|\psi_{E_k}^m\right\rangle$ from classical queries

Input: A classical oracle to E_k, m
Output: $\left|\psi_{E_k}^m\right\rangle$

1: $|\phi\rangle \leftarrow \bigotimes_m \sum_x |x\rangle|0\rangle$
2: **For** $0 \leqslant i < 2^n$ **do**
3: Query $E_k(i)$
4: Apply to each register in $|\phi\rangle$ the operator

$$|x\rangle|y\rangle \mapsto \begin{cases} |x\rangle|y \oplus E_k(i)\rangle & \text{if } x = i \\ |x\rangle|y\rangle & \text{otherwise} \end{cases}$$

5: **Return** $|\phi\rangle$

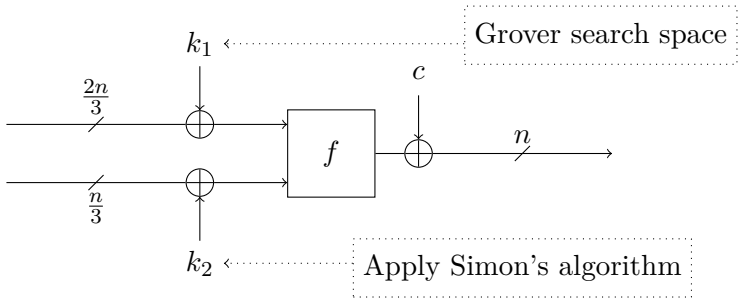


Figure 6.1: Decomposition of E_k for period domain reduction

Restrictions. The use of classical queries adds some restrictions to the instances we can solve. Notably, to construct the database, we need to query 2^n times the same function, and we lose the ability to use Simon's algorithm with a function that changes at each query.

Search for a period with classical queries. We can directly plug the database generation (Algorithm 6.5) into the offline search (Algorithm 6.4), to obtain our algorithm.

Theorem 6.4 (Complexity of Algorithm 6.4 with classical queries)**.** *Algorithm 6.4 finds i_0 in $\frac{\pi}{4}2^{k/2}$ iterations of Algorithm 6.3, with probability greater than $1 - 2^{-k} - 2^{(k+n-m)/2+2}$. It uses 2^n classical queries and $m2^n$ simple quantum operations to produce $\left|\psi_{E_k}^m\right\rangle$.*

Domain Reduction. In cases where we have a small search space (and in the extreme case of Simon's problem), Theorem 6.4 is less than ideal, as it imposes to query classically all the entries to the secret function. We can overcome this limitation by artificially reducing the size of the periodic function. Indeed, Consider the case $E_k(x) = f(x \oplus k) \oplus c$. The function $E_k(x) \oplus f(x)$ has the period k. Furthermore, we can restrict the size of the period by considering the family of functions

$$\begin{aligned} F : \{0,1\}^{2n/3} \times \{0,1\}^{n/3} &\to \{0,1\}^n \\ i, x \quad &\mapsto f(i||x) \end{aligned}$$

If we split k into k_1 and k_2 as in Figure 6.1, then we have that the function $E_k(x) \oplus F(k_1, x)$ has the period k_2. We can also apply this approach when we need to search for a periodic function. Such domain reduction allows to reduce the amount of classical queries, at the expense of a larger search space.

Theorem 6.5 (Complexity of Algorithm 6.4 with domain reduction)**.** *Let $E_{k_1,k_2} : \{0,1\}^n \to \{0,1\}^n$ be a function of the form $E_{k_1,k_2}(x) = f(k_1, x \oplus k_2) \oplus c$, with f a publicly computable function and k_1 a k-bit key. For all $u \in [0; n]$, Algorithm 6.4 can find k_2 in $\frac{\pi}{4}2^{(n-u+k)/2}$ applications of Algorithm 6.3, $m2^{u/3}$ simple quantum operations*

and $2^{u/3}$ *classical queries with a probability greater than* $1-2^{-(n-u+k)/2}-2^{(k+n-u-m)/2+2}$. *It uses* $\mathcal{O}(nm)$ *quantum memory and* $\mathcal{O}(n)$ *classical memory.*

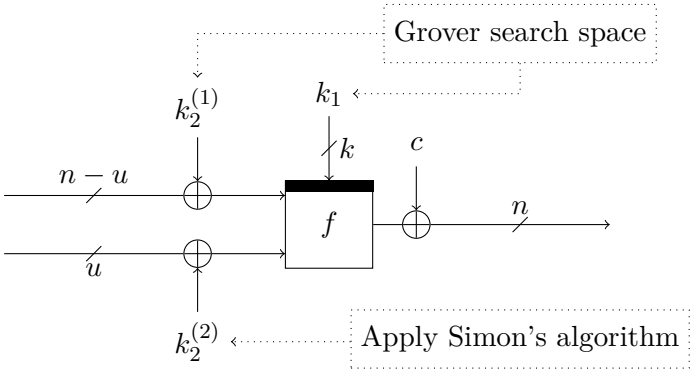


Figure 6.2: Domain reduction with an initial search space.

Proof. Define the function

$$\begin{aligned} F : \{0,1\}^k \times \{0,1\}^{n-u} \times \{0,1\}^u &\to \{0,1\}^n \\ i, j, x &\mapsto f(i, j||x) \end{aligned}$$

and k_1, $k_2 = k_2^{(1)}||k_2^{(2)}$ as in Figure 6.2. As $E_{k_1,k_2}(x) \oplus F(k_1, k_2^{(1)}, x)$ has period $k_2^{(2)}$, we can directly apply Theorem 6.4 to obtain the wanted result. □

Corollary 6.1 (Simon's algorithm with classical queries)**.** *Let* $E_k : \{0,1\}^n \to \{0,1\}^n$ *be a function of the form* $E_k(x) = f(x \oplus k) \oplus c$, *with* f *a publicly computable function. Algorithm 6.4 finds* k *in* $\frac{\pi}{4}2^{n/3}$ *applications of Algorithm 6.3,* $m2^{n/3}$ *simple quantum operations and* $2^{n/3}$ *classical queries with a probability greater than* $1-2^{-n/3}-2^{n/3-m/2+2}$. *It uses* $\mathcal{O}(nm)$ *quantum memory and* $\mathcal{O}(n)$ *classical memory.*

When domain reduction is applicable, the best we can achieve is a time and query cost of $2^{n/3}$. This happens to match the query cost of the Brassard-Høyer-Tapp algorithm for collision (Section 3.3.2). The two problems are slightly different, as on the one hand we need a period on n bits, which is a stronger constraint than a collision, and on the other hand, we can perform an additional search on k bits and balance the costs, which is weaker than having a simple function on which we look for a collision. Moreover, this algorithm is memoryless, while BHT needs a huge amount of quantum RAM.

6.4 Implications

This section shows that even with only a classical access to a secret function, Simon's algorithm can be useful. This is, to our knowledge, the first utilization of Simon's

algorithm which does not require a quantum access to a secret function. This also makes a bridge between quantum algorithms that require a quantum oracle, like the various quantum algorithms for the hidden subgroup problem, and the ones that do not, like the search algorithms. In terms of improvements, the offline algorithm proposes 3 gains: a gain in queries with quantum oracles, a broader scope of applications, as it is applicable with classical queries, and a gain in memory if we compare with the Brassard-Høyer-Tapp algorithm for collisions from Section 3.3.2.

Part II
Quantum Cryptanalysis

Chapters

Chapter 7
Hidden Structures in Symmetric Cryptography

The previous part proposed multiple quantum algorithms to solve problems of the form "given f and g such that $f(x) = g(x+s)$, find s". This kind of structure appears often in symmetric cryptography. The first cryptanalysis of this kind was proposed by Kuwakado and Morii in 2010. It allowed to distinguish between 3 rounds of a Feistel cipher and a random function [KM10], and was later improved to 4 rounds when a decryption oracle is available [Ito+19]. They also proposed in 2012 an attack on the widespread and classically proven secure Even-Mansour construction [KM12]. In 2013, Roetteler and Steinwandt proposed a model of quantum related-key attacks, and showed that we cannot have secure constructions in this model [RS15]. These early results have been largely extended by Marc Kaplan, Gaëtan Leurent, Anthony Leverrier and María Naya-Plasencia [Kap+16], who attacked multiple concrete ciphers and modes of operations, and introduced the *quantum slide attacks*, which are presented in Chapter 9. Leander and May then proposed to attack the FX construction by combining a quantum search with Simon's algorithm [LM17].

All the aforementioned attacks require quantum queries. With Akinori Hosoyamada, María Naya-Plasencia, Yu Sasaki and André Schrottenloher, we showed that some of the attacks can also be applied when only classical queries are available [Bon+19], using the algorithms of Chapter 6.

This chapter proposes a survey of the currently known quantum distinguishers and quantum hidden structure attacks in symmetric cryptography. They come from [KM10; KM12; RS15; Kap+16; SS17; LM17; BN18; DLW19; ND19; II19; Ito+19; Bon+19]. We also present the classical-query variant when applicable. We developed some more advanced attacks that rely on the same principles. They are presented in their own chapters: a cryptanalysis of AEZ [Bon17] is proposed in Chapter 8 and quantum slide attacks [Kap+16; BNS19a] are developed in Chapter 9.

Contents

7.1 Claims in symmetric cryptography

Until recently, most of the primitives in symmetric cryptography did not consider these models, and did not make any claim against quantum attackers. This is starting to change, as for example the instances of the Farfalle construction Kravatte [Ber+17] and Xoofff [Dae+18] have a security claim against quantum attacks, but explicitly restrict to classical queries, and the lightweight authenticated scheme Saturnin [Can+19], which aims to be secure even if quantum queries are accessible. Finally, Bernstein and Lange have proposed in Nature [BL17, Table 1] some "conjectured security levels" against a quantum attacker for multiple symmetric primitives, including Poly1305.

A quantum analysis of Kravatte and Xoofff (which do not violate their security claims) is presented in Section 7.5, and two quantum superposition attacks against Poly1305 are presented in Section 7.7.3.

7.2 General method

The idea underlying all the attacks presented here is to construct a periodic function (to be used with Simon's algorithm), or for 2 functions that are identical up to a shift of their input (to be used with Simon's or Kuperberg's algorithm, depending on the shift). For key-recovery attacks, we also want the period or shift to provide some information on the sought key material.

A typical pattern for the functions is $g(f(x \oplus a) \oplus f(x' \oplus b))$, which, if we impose that $x = x'$, has a period $a \oplus b$. Another typical pattern is $g(f(x) \oplus y)$, from which we can compute the function

$$F : b, y \mapsto \begin{cases} g(f(x_0) \oplus y) & \text{if } b = 0, \\ g(f(x_1) \oplus y) & \text{if } b = 1. \end{cases}$$

The function F has the period $(1||f(x_0) \oplus f(x_1))$.

In general, the functions we will consider won't be 2-to-1 periodic functions. As shown in Proposition 4.4 and Lemma 5.5, this is not an issue.

Targets. All the attacks presented here target constructions that contain some secret material. Indeed, to obtain an attack, we would need to have a public function that embeds a secret or unknown shift, which is more to be expected in asymmetric constructions. In practice, no quantum attacks that target hash functions have been proposed beside some generic exponential-time algorithms, and the Davies-Meyer and Merkle-Damgård constructions, used in particular by SHA-2 [SHA2], are proven in a quantum setting [HY18].

7.3 Quantum distinguishers

This section presents 3 quantum distinguishers on cryptographic constructs, that is, quantum algorithms that take as input a quantum oracle to either the stated construction or a random function, and have to decide which one it is. These distinguishers do not have a classical equivalent, and show that access to a quantum oracle is very powerful.

7.3.1 One-time pad

The one-time pad is a very simple cipher which has been proven *unconditionally* secure by Shannon in 1949 [Sha49]. It uses a key taken uniformly at random of the same size than the message, and simply xors them. A given key shall only be used once. As the confidentiality of the plaintext is information-theoretical, it is also impossible from a quantum computer to recover it from a ciphertext. However, the classical security is stronger than this: given a plaintext and its associated ciphertext, it is not possible to state if the ciphertext has been produced with a one-time-pad. This is what becomes possible with a quantum oracle: it is possible, in one query, to distinguish between a one-time pad and a random function, by using the possibility to query a superposition of inputs.

The idea of Algorithm 7.1 is that for a one-time pad, $P(x) \oplus x$ does not depend on its input, while it does for a random function. Hence, at step 4, the two registers are entangled only if P is not a one-time pad. This allows for a very efficient distinguisher, as entanglement can be observed:

Proposition 7.1 (Quantum distinguisher for the one-time pad)**.** *Algorithm 7.1 uses one quantum query to P, and returns:*

- *"P is a one-time pad" with probability 1 if P is a one-time pad,*
- *"P is a random function" with probability $1 - \frac{1}{2^{n-1}}$ if P is a random function.*

Proof. If P is a one-time pad, then the state before step 4 is

$$\frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle |k\rangle \quad .$$

Hence, after step 4, the state is $|0\rangle |k\rangle$.

Algorithm 7.1 Quantum distinguisher of the one-time pad

Input: An oracle to the n-bit function P

Output: Either "P is a one-time pad" or "P is a random function"

1: Construct with Hadamard gates the state

$$\frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle |0\rangle$$

2: Call the oracle, to produce

$$\frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle |P(x)\rangle$$

3: Add the input to the output, to produce

$$\frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle |P(x) \oplus x\rangle$$

4: Apply Hadamard gates on the first register, to produce

$$\frac{1}{2^n} \sum_{y=0}^{2^n-1} \sum_{x=0}^{2^n-1} (-1)^{x \cdot y} |y\rangle |P(x) \oplus x\rangle$$

5: Measure the value y_0 in the first register

6: **If** $y_0 = 0$ **then**

7: **Return** "P is a one-time pad"

8: **Else**

9: **Return** "P is a random function"

If P is a random function, then the state after step 4 is

$$\frac{1}{2^n} \sum_{y=0}^{2^n-1} \sum_{x=0}^{2^n-1} (-1)^{x \cdot y} |y\rangle |P(x) \oplus x\rangle .$$

The amplitude of $|0\rangle$ in the first register depends on the number of preimages of $P(x) \oplus x$. Indeed, each value of $P(x) \oplus x$ adds a contribution of $\left(\frac{1}{2^n} \sum_{x:P(x)\oplus x=\alpha} (-1)^{x \cdot y} \right)^2$ to the amplitude squared of each y, and in particular, for $y = 0$, the contribution is $\left(\frac{r}{2^n}\right)^2$ if $P(x) \oplus x$ has r preimages. For a random function, the fraction of image points with r preimages is $\frac{1}{r!e}$ [FO90]. Hence, the expected amplitude squared of $y = 0$ is

$$\sum_{r=0}^{\infty} \frac{2^n}{r!e} \left(\frac{r}{2^n}\right)^2 = \frac{1}{2^n e} \sum_{r=0}^{\infty} \frac{r^2}{r!} = \frac{1}{2^n e} \left(\sum_{r=0}^{\infty} \frac{r(r-1)}{r!} + \sum_{r=0}^{\infty} \frac{r}{r!} \right) = \frac{2}{2^n}$$

□

Other group laws. This distinguisher does not require a key xoring to be efficient, it only needs to be able to transform $P(x)$ into a value independent from the input. In particular, this is achievable if the key is inserted using a group law. This fails if the relation between the plaintext and the ciphertext is more complex, which is typically what a block cipher aims to achieve.

Applications. This distinguisher may look frightening, as for example stream ciphers are very close to a one-time-pad, and can be distinguished from a random permutation. It requires one chosen-plaintext (superposition) query. Classically, the xored key can be recovered from one known-plaintext query. The main difference is that the distinguisher proves that a constant value has indeed been xored to the plaintext. Classically, the same could be achieved from two known queries with the same key. Hence, this distinguisher is only relevant for functions with some randomness that change for each query (typically, a nonce). For example, it can be used to attack Poly1305, as presented in Section 7.7.3. It also shows that some classical proof techniques, which rely on the indistinguishability of some functions from a random function may be hard to translate to a quantum setting in some cases.

7.3.2 Feistel networks

Feistel networks are a popular construction of block ciphers, their most notable representative is DES [DES], which was the standard block cipher before being obsoleted by AES [AES].

Feistel networks iterate on an n-bit state a round function of the form $(L, R) \mapsto (R \oplus F(L), L)$. Half of the state is passed through a round function and added to the other half and then the two halves are swapped, as in Figure 7.1.

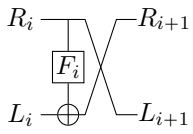


Figure 7.1: Feistel round function

Classical security. The security of Feistel networks has been studied by Luby and Rackoff [LR88]. They showed that if the round functions are pseudorandom functions, then with access to an encryption oracle, one needs $\Omega\left(2^{n/4}\right)$ queries to distinguish between 3 rounds of a Feistel network and a random permutation. If a decryption oracle is also available, then 3 rounds is not enough, and you need 4 rounds to have a safe construction, that still requires $\Omega\left(2^{n/4}\right)$ queries to be distinguished from a random permutation. This proof is optimal, as some attacks matching the bound have been proposed [Pat92].

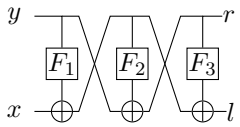


Figure 7.2: 3-round Feistel network $P(x, y) = (l, r)$

Generalizations. Feistel networks can be naturally generalized by increasing their number of branches This leaves more freedom in the branch addition patterns. 3 constructions are generally considered, called type-1, type-2 and type-3 Feistel networks [ZMI90], which differ on the number of branches added to their neighbour at each round.

7.3.2.1 Quantum Distinguisher on 3 rounds

The quantum distinguisher of Kuwakado and Morii [KM10] is the first known application of Simon's algorithm in cryptography. Its aim is to distinguish between a 3-round Feistel cipher and a random permutation, that is, given a quantum query access to an unknown permutation, identify with a good probability whether or not it is a 3-round Feistel network. This is achieved by constructing from the unknown permutation P a function that will always be periodic if it is a Feistel cipher, and, with an overwhelming probability, won't be for a random function.

From Figure 7.2, we can see that $r = y \oplus F_2(x \oplus F_1(y))$. Hence, if we choose two distinct values α and β, we can define

$$f : \begin{array}{ccc} \{0,1\} \times \{0,1\}^{n/2} & \to & \{0,1\}^{n/2} \\ b, x & \mapsto & \begin{cases} Trunc_R(P(x,\alpha)) \oplus \alpha & \text{if } b = 0, \\ Trunc_R(P(x,\beta)) \oplus \beta & \text{if } b = 1. \end{cases} \end{array}$$

We have that for a Feistel cipher, $f(0, x) = F_2(x \oplus F_1(\alpha))$, and $f(1, x) = F_2(x \oplus F_1(\beta))$. Hence, in that case, for all x, $f(0, x) = f(1, x \oplus F_1(\beta) \oplus F_1(\alpha))$.

Algorithm 7.2 Quantum distinguisher of a 3-round Feistel

Input: m, A quantum oracle to the n-bit function P
Output: Either "P is 3-round Feistel" or "P is a random function"
1: Call m times Simon's routine on the function f.
2: **If** The m values have a maximal rank **then**
3: **Return** "P is a random function"
4: **Else**
5: **Return** "P is a 3-round Feistel"

Theorem 7.1 (Quantum distinguisher for 3-round Feistels [KM10])**.** *With $m = O(n)$ queries to P, Algorithm 7.2 returns:*

- *"P is a 3-round Feistel" with probability 1 if P is a 3-round Feistel,*

- *"P is a random permutation" with probability in* $1 - \mathcal{O}\left(2^{-n}\right)$ *if* P *is a random permutation.*

Proof. If P is a 3-Round Feistel, then f is periodic. If P is a random permutation, f is a random function, then the $n/2$-bit values sampled by Simon's routine are random and from Section 4.2.2, with $c \times n/2$ queries, with c a small constant, we obtain a set of values of maximal rank. □

Generalizations. A very similar distinguisher can be applied against a $3d-3$-round d-branches Type-1 generalized Feistel network [ND19; II19] and a $d+1$-round d-branches Type-2 generalized Feistel network [DLW19]. With access to the decryption function only, the same approach can distinguish a d^2-d+1-round d-branches Type-1 generalized Feistel network [II19].

Safe constructions. One could ask if this attack can be performed on a higher number of rounds, and what is the minimal number of round to obtain a secure primitive. This question has been studied by Akinori Hosoyamada and Tetsu Iwata [HI19]. They showed that 4 rounds are enough, that is, if only P is accessible one needs $\Omega\left(2^{n/12}\right)$ quantum queries to distinguish between a 4-round Feistel and a random permutation. They also proposed a distinguisher that requires $\mathcal{O}\left(2^{n/6}\right)$ queries, which leaves a gap between the current proof and the best known distinguisher to date. The case where P^{-1} is also accessible is treated in the next section.

7.3.2.2 Quantum Distinguisher on 4 rounds

The previous distinguisher identifies a 3-round Feistel network using only a quantum access to the function P. This section presents the quantum distinguisher on a 4-round Feistel proposed by Gembu Ito, Akinori Hosoyamada, Ryutaroh Matsumoto, Yu Sasaki, and Tetsu Iwata [Ito+19], which requires quantum access to both P and P^{-1}.

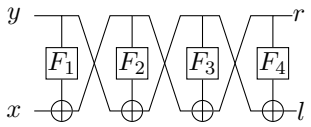


Figure 7.3: 4 rounds Feistel network $P(x, y) = (l, r)$

From Figure 7.3, we can see that $l = y \oplus F_2(x \oplus F_1(y)) \oplus F_3(y \oplus F_2(x \oplus F_1(y))) \oplus F_4(r)$. We can remark that every x involved in this equality has the value $F_1(y)$ added to it. We however do not have directly a shift between to functions if we take two distinct values for y, due to the term $F_3(y \oplus F_2(x \oplus F_1(y)))$, which changes when y change. However, we can add a difference to the value fed into F_3 if we encrypt, add a constant to l, and then decrypt. If we fix two values for y, this allows to construct two functions whose components in F_3 will be equal. For example, we can construct the function f of Figure 7.4.

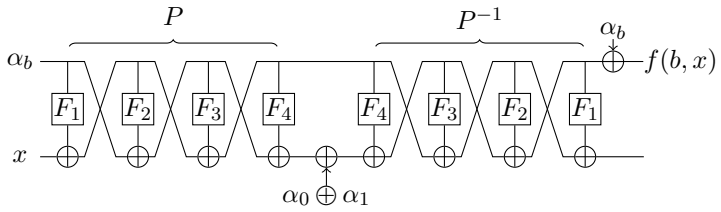


Figure 7.4: Periodic function for the 4-round Feistel quantum distinguisher

We have, if we note $F'_b(x) = F_3(\alpha_b \oplus F_2(x))$

$$\begin{aligned} f(b,x) &= \alpha_b \oplus F_2(x \oplus F_1(\alpha_b)) \oplus \alpha_0 \oplus \alpha_1 \oplus F_2(x \oplus F_1(\alpha_b) \oplus \\ &\qquad F_3(\alpha_b \oplus F_2(x \oplus F_1(\alpha_b))) \oplus F_3(\alpha_b \oplus \alpha_0 \oplus \alpha_1 \oplus F_2(x \oplus F_1(\alpha_b)))) \oplus \alpha_b \\ &= F_2(x \oplus F_1(\alpha_b)) \oplus \alpha_0 \oplus \alpha_1 \oplus F_2(x \oplus F_1(\alpha_b) \oplus F'_b(x \oplus F_1(\alpha_b)) \oplus F'_{1-b}(x \oplus F_1(\alpha_b))) \end{aligned}$$

Hence, we can see that $f(b,x)$ is periodic, of period $(1, F(\alpha_0) \oplus F(\alpha_1))$.

The distinguisher used is exactly the same as Algorithm 7.2, but with a slightly different function f, and has the same efficiency.

Safe constructions. While the 4-round distinguisher cannot be naturally extended to 5 rounds, the quantum security of 5-round Feistels is not currently known.

7.4 The case of quantum-related key attacks

Related-keys attacks [Bih94; BDK08] are a class of attacks on symmetric ciphers where the attacker is not only allowed to query some messages encrypted with an unknown fixed key, but also to add an offset on the secret key. If this kind of attack can lead to a better understanding of the primitive, it is generally considered less realistic. In most practical applications, we either have a cipher with a unique fixed key, or ciphers with multiple independent keys, a notable exception being hash functions constructed from a block cipher.

Moreover, Roetteler and Steinwandt have shown that this model can be degenerate with superposition queries [RS15]. Using Simon's algorithm, one can break any cipher with a polynomial amount of quantum queries and computations if, given a block cipher E_k, we have access to a quantum oracle able to compute $|\delta\rangle|X\rangle|0\rangle \mapsto |\delta\rangle|X\rangle|E_{k\oplus\delta}(X)\rangle$. We can remark that this function is the same as the publicly available function E_δ, with a fixed offset in the key. There are many periodic functions constructible from this. For example, it is possible construct a circuit that computes, for any given plaintext P,

$$f_P(\delta) = E_{k\oplus\delta}(P) \oplus E_\delta(P).$$

The function f_P is periodic, of period k. However, the input is in the key space, of size 2^k, and the output is in the message space, of size 2^n. This can be an issue if k

is larger than n, as Simon's algorithm works best with 2-to-1 functions. This is easily solved if we consider multiple f_X with distinct X, with for example

$$F(x) = f_{P_1}(x), \ldots, f_{P_r}(x)$$

The function F also has the period k, and its output length can be made as large as we want.

As this attack is completely generic, we can interpret it as a warning on models of quantum attacks, as with some of them, any construction can be easily broken. A different example, with a different model of related-key, is presented in Section 9.2.1.

7.4.1 With classical queries

The offline Simon's algorithm leads to a more interesting result: the function f_P fits into the canvas of Problem 6.3. In order to have a relevant attack, we need to reduce the domain of the period, and to increase the (previously empty) domain of the search space. We can consider the function

$$\begin{array}{rccc} f'_P : & \{0,1\}^{2n/3} \times \{0,1\}^{n/3} & \to & \{0,1\}^n \\ & (i,x) & \mapsto & E_{k\oplus(0||x)}(P) \oplus E_{(i||x)}(P) \end{array}$$

If we note $k = k_1||k_2$, with $|k_1| = 2n/3$, then $f'_P(k_1, \cdot)$ is periodic of period k_2. Hence, we can apply Theorem 6.4, to obtain a key recovery in $\mathcal{O}\left(2^{n/3}\right)$ classical queries and quantum time.

Application. This new generic attack uses a model of quantum attacks with classically related keys that was not considered before. As it provides a bound in this new model, it may be used to extend the security claims of the NIST lightweight competition candidate SATURNIN, which aims to be secure against quantum queries, and whose variant SATURNIN_{16} is claimed to be secure against *classical* related-keys.

7.5 Even-Mansour

The Even-Mansour construction [EM97], presented in Figure 7.5, is often referred as the minimal block cipher. It uses two n-bit keys k_{in}, k_{out}, and a publicly known random permutation P. It was later shown that the cipher can be made even simpler without a security loss by having $k_{in} = k_{out}$ [DKS12]. This construction is often used as a building block in more complex designs.

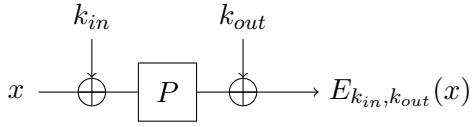


Figure 7.5: The Even-Mansour construction.

Classical security. The original paper proved that any attack that uses D queries to E_k and T queries to P must satisfy $DT = \Omega(2^n)$. This proof is information-theoretical, that is, only the queries to either $E_{k_{in},k_{out}}$ or P allow to recover some information. Hence, one needs $\mathcal{O}\left(2^{n/2}\right)$ queries to E_k or P to break it, and multiple matching attacks have been proposed [Dae93; BW00; DKS12].

Quantum security. As shown by Kuwakado and Morii [KM12], it turns out that this classically proven construction is completely broken if one is allowed an access in quantum superposition to $E_{k_{in},k_{out}}$ and P. Indeed, the function $E_{k_{in},k_{out}}(x) \oplus P(x)$ has the period k_{in}. Hence, roughly n queries are enough to recover k_{in}. Once k_{in} is known, k_{out} is trivial to recover. This result might appear quite surprising, as the proof was *information-theoretical*. The issue is that the proof assumed that a query corresponds to a pair $x, f(x)$, with f equal to $E_{k_{in},k_{out}}$ or P. This is no longer the case if we allow a quantum access.

With classical queries. If we restrict to classical access to both P and $E_{k_{in},k_{out}}(x)$, then the proof holds, and quantum computing offers no advantage. However, P is a public permutation. Hence, an insteresting scenario is to study what happens if we are restricted to classical queries to $E_{k_{in},k_{out}}(x)$, and allow quantum queries to P, which is something we can always do if a circuit of P is publicly given. In that case, Theorem 6.5 is applicable, and k_{in} can be recovered in $\mathcal{O}\left(2^{n/3}\right)$ classical queries to $E_{k_{in},k_{out}}$, quantum queries to P, classical and quantum time.

Concrete instances. The Farfalle construction [Ber+17] is an Even-Mansour construction if the input message is only 1 block long, hence with quantum queries, the construction is broken. With a restriction to classical queries, then it depends on the instance, as the state keys are derived from a smaller secret key. The Kravatte instance [Ber+17] has a state size of 1600 bits, and a key size between 256 and 320 bits, which leads to a key recovery at a whopping cost of 2^{533} data and time, while the direct key search would cost at most 2^{160}. Xoofff [Dae+18] has a state size of 384 bits and a key size between 192 and 384 bits. Our attack needs 2^{128} data, which is exactly the data limit of Xoofff, and 2^{128} time. Hence, it performs better than the exhaustive key search if the key is longer than 256 bits. This is however not enough to contradict its security claim against quantum adversaries, who are always restricted to a search of at most 192 secret bits.

7.6 FX Construction

The FX construction [KR96] is a simple way to increase the key length of a block cipher. It is similar to the Even-Mansour construction, except that the public permutation is replaced with a keyed block cipher, as presented in Figure 7.6.

Quantum security. As it is very similar to the Even-Mansour construction, it has the same issues if a quantum oracle is available. As shown in [LM17], this construction

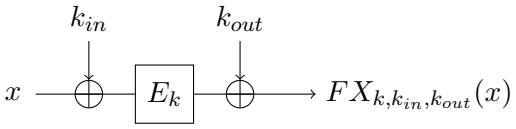


Figure 7.6: The FX construction.

can be attacked at roughly the cost of an exhaustive search on k, by doing a search for a periodic function (Algorithm 6.1), as the function $FX_{k,k_{in},k_{out}}(x) \oplus E_k(x)$ has the period k_{in}. We improved this result by showing that the number of quantum queries to $FX_{k,k_{in},k_{out}}(x)$ can be restricted to a polynomial amount with the offline Simon's algorithm (Algorithm 6.4).

With classical queries. The FX construction fits into the canvas of Theorem 6.5, hence we can directly apply it. If E_k uses an m-bit key and $m \leqslant 2n$, then the best we can achieve is a cost of $\mathcal{O}\left(2^{(n+m)/3}\right)$, classical queries and quantum time, with a period of size $(n+m)/3$ and a search space of size $(2n-m)/3+m$.

Concrete instances. Multiple instances of the FX construction have been proposed. The original proposal is an improvement on DES called DESX, which adds two 64-bit whitening keys to the 56-bit DES key. With quantum access to DESX, the keys can be recovered with around 120 quantum queries and 2^{28} computations of Simon's algorithm. If only classical queries to DESX are allowed, then this increases to 2^{40} classical queries and 2^{40} computations of Simon's algorithm. In both cases, only a small amount of classical and quantum memory is used.

PRINCE [Bor+12] and PRIDE [Alb+14] are two ciphers using the FX construction with a 64-bit state, a 64-bit inner key and two 64-bit whitening keys. Hence, the keys can be recovered with around 128 quantum queries and 2^{32} computations of Simon's algorithm if quantum queries are allowed, and 2^{42} classical queries and computations of Simon's algorithm if only classical queries are allowed.

7.6.1 Multiple-FX

In some constructions, we do not have access to only one FX construction, but multiple ones, with a fixed block cipher key k, but varying whitening keys k_{in}, k_{out}. In that case, we can construct a periodic function computable from quantum queries without the knowledge of k. Indeed, from $E_k(x \oplus k^1_{in}) \oplus k^1_{out}$ and $E_k(x \oplus k^2_{in}) \oplus k^2_{out}$, we can construct

$$f(x) = E_k(x \oplus k^1_{in}) \oplus k^1_{out} \oplus E_k(x \oplus k^2_{in}) \oplus k^2_{out}$$

which has the period $k^1_{in} \oplus k^2_{in}$. This is weaker than a complete key recovery, but as shown in [Kap+16], in many cases, the knowledge of this value either allows to recover a key or allows to break the integrity of a scheme.

With classical queries. The function f does not have a publicly computable part, hence this attack does not appear to be applicable with classical queries, and we need to fall back to the attack on the single FX construction.

Concrete instances. The LRW construction [LRW02] of tweakable block ciphers and the disk encryption mode XTS [Mar10] can be seen as instances of the Multiple-FX construction. PMAC [Rog04] contains internally the same block cipher with different offsets, and as such is broken with quantum queries. A variant of PMAC is also used in AEZ [HKR15], which is presented in Chapter 8.

7.7 MACs

This section presents the attacks on various MAC constructions that were first introduced in [Kap+16; SS17; BN18]. Most of these attacks require quantum queries, with the notable exception of Chaskey [Mou+14].

7.7.1 CBC-MAC

CBC-MAC [BKR00; BR00; IK03] is an early MAC construction inspired by the CBC block cipher mode of operation. The message m is split into fixed-size blocks m_i, and iteratively injected into a state, as in Figure 7.7.

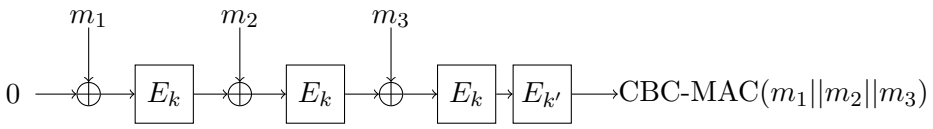


Figure 7.7: CBC-MAC, from [BKR00]

CBC-MAC can be attacked if we consider two distinct values α_0, α_1 and the function

$$f(b, x) = \text{CBC-MAC}(\alpha_b||x)$$

This function has the period $(1, E_k(\alpha_0) \oplus E_k(\alpha_1))$. Hence, Simon's algorithm retrieves $\Delta = E_k(\alpha_0) \oplus E_k(\alpha_1)$. This allows to forge some messages, as the messages $\alpha_0||x||m$ and $\alpha_1||x \oplus \Delta||m$ will have the same tag.

7.7.2 Chaskey

Chaskey [Mou+14] is a lightweight MAC that can be seen as a combination of Even-Mansour and CBC-MAC (Figure 7.8). Hence, it is vulnerable to the same attack as CBC-MAC. However, the specific structure of Chaskey (in particular, the lack of π between the xoring of the last message block and first xoring of K_1) allows to attack it with classical queries. Indeed, with a message that vary only in its last block, Chaskey is of the form $\pi(x \oplus K_1 \oplus c) \oplus K_1$, which is an Even-Mansour construction.

The Chaskey round function is applied on 128 bits. It contains 16 rounds with 4 modular additions on 32 bits, 4 XORs on 32 bits and some rotations. With a data limit of 2^{48} blocks, as advocated in the specification, the attack can make use of 2^{48} queries for Simon's algorithm. Hence, we can attack Chaskey with 2^{48} classical queries, and roughly $2^{(128-48)/2} = 2^{40}$ computations of Simon's algorithm. Interestingly, as π is extremely cheap to compute, we can expect that the dominant cost of the Simon computation will be the resolution of the 48-dimensional linear system.

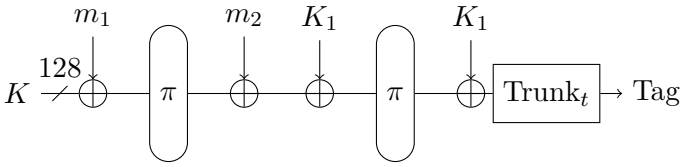


Figure 7.8: Two-block Chaskey example.

7.7.3 Poly1305

Poly1305 is a widely used MAC designed by Bernstein [Ber05]. Poly1305 associated with Chacha20 has been standardized for TLS 1.2 [Lan+16], is in one of the recommended cipher suites of TLS 1.3 [Res18], and is notably supported by OpenSSH, Firefox and Chrome.

7.7.3.1 Description

We describe here the original construction Poly1305-AES from [Ber05], which uses AES as a block cipher in the Wegman-Carter-Shoup construction [WC81; Sho96]. Our analysis focuses only on the hash part, hence it can also be applied to different instances with different block ciphers. Concretely, Poly1305-AES uses two 128-bit keys (r, k) and a 128-bit nonce n, takes as input a variable-length message m considered as an array of 128-bit blocks, and outputs a 128-bit tag. For efficiency purposes, 22 bits of r are fixed to 0, which means it can only take 2^{106} different values. The function is

$$\text{Poly1305-AES}_{(r,k,n)}((m_i)_{i\leqslant q}) = \left(\sum_{i=1}^{q}(m_{q-i+1} + 2^{128})r^i \mod 2^{130} - 5\right) + \text{AES}_k(n).$$

Security. Poly1305-AES has a classical security proof that, if an adversary knows up to 2^{64} valid message/tag tuples and do not leverage a weakness of AES, the success probability of a forgery is less than 2^{-106}. The best known classical attack is a universal forgery by Leurent and Sibleyras [LS18] that costs around 2^{83} time and queries. The designer described in [BL17] a conjectured classical and quantum security of Poly1305 of 128 bits.

Quantum attacks For our quantum attacks, we suppose that we have access to a quantum oracle that implements

$$\text{Poly1305} : |m_1\rangle |m_2\rangle |0\rangle \mapsto |m_1\rangle |m_2\rangle |\text{Poly1305-AES}_{(r,k,n)}(m_1, m_2)\ \rangle ,$$

with two unknown fixed classical keys r, k and a classical nonce n that will differ for each call.

We proposed two superposition attacks in [BN18]. The first attack uses the fact that this MAC can be split as the sum of two independent functions. The second attack relies on the polynomial structure of the hash function.

7.7.3.2 Distinguisher-based attack

It is possible to see Poly1305-AES as a one-time pad, with the message $\sum_{i=1}^{q}(m_{q-i+1} + 2^{128})r^i \mod 2^{130} - 5$ and the key $\text{AES}_k(n)$. Hence, we can guess the value of r, and then check if what we obtain corresponds to a one-time-pad, or to a random function, using the distinguisher of Section 7.3.1.

Concretely, we consider

$$A(x, s) = \text{Poly1305}(x) - \left((x + 2^{128})s \mod 2^{130} - 5\right) \mod 2^{128},$$

where $\text{Poly1305}(x)$ is the tag of the one-block message x, with an unspecified nonce. If $s = r$, then $A(x, r) = \text{AES}_k(n)$ (which is a fixed value for each query), and if not, it will be $A(x, s) = \left((x + 2^{128})r \mod 2^{130} - 5\right) - \left((x + 2^{128})s \mod 2^{130} - 5\right) + \text{AES}_k(n) \mod 2^{128}$. As the function $x \mapsto xs$ in $\mathbb{Z}/\left(2^{130} - 5\right)$ is, with an overwhelming probability, a permutation, we will measure a 0 for a wrong guess with a probability of around 2^{-126}.

This distinguisher can then be used in a Grover search on r that calls Poly1305: at each step, we compute $\sum_x |x\rangle |A(x, r)\rangle$, apply a Hadamard gate on the first register, and mark r if the first register is 0. As the test function is not perfect, the success probability of the algorithm will be smaller than 1, but the error of the test function is small enough to have a negligible final error.

There is still one problem: to perform a Grover search, we need to uncompute our computations. And as our oracle generates a fresh nonce at each query, we cannot uncompute the function. This is however not a problem, as a nonce difference in the uncomputation will produce a fixed difference in the output $(\text{AES}_k(n) - \text{AES}_k(n'))$. This means that the uncomputation would leave the register in a non-zero but non-entangled state, which allows to safely erase and reset it. We can hence apply a Grover search on r. As it has 106 variable bits, it would cost around 2^{53+1} queries and time to retrieve r.

Remark 7.1. This attack leverages the fact that Poly1305 can be written as the sum of two functions that act on independent variables. This simple relation between the two functions allows us to attack only one of them.

7.7.3.3 Hidden Shift Attack

We proposed a second attack that uses the polynomial structure of the universal hash function to obtain the key r. The commutative algebra $\mathbb{Z}/\left(2^{130} - 5\right)[X]$ contains many

possible shift structures, both in $\mathbb{Z}/\left(2^{130}-5\right)$ (with addition) and in $\mathbb{Z}/\left(2^{130}-6\right)$ (with multiplication). For example, one can consider the two functions $f(x) = xr + r^2 + 2^{128}(r + r^2)$ and $g(x) = xr + 2^{128}(r + r^2)$, that satisfy $f(x) = g(x + r)$. There is no way to access them directly, but we can call $F(x) = \text{Poly1305-AES}_{(r,k,n)}(1, x)$ and $G(x) = \text{Poly1305-AES}_{(r,k,n)}(0, x)$, which also satisfy $F(x) = G(x+r)$ if the nonce is the same.

There are two issues that do not allow the direct application of a hidden shift algorithm on F and G. First, the nonce changes at each query, which means that in order to have $F(x) = G(x + r)$, we must be able compute F and G in only one query to Poly1305. This can be circumvented by using the fact that both are of the form Poly1305$(a(x))$, with $a(x)$ a function of x: one can compute $a_F(x) = (1, x)$ and $a_G(x) = (0, x)$ in superposition in an auxiliary register, and then call the oracle to Poly1305 on it. Second, and more annoyingly, the inputs of Poly1305 are restrained to be between 0 and $2^{128} - 1$, which means we cannot sample all group elements.

This can still be solved by using Lemma 5.7, as we can query $[0; 2^{128})$. Indeed, if r is very small, then the functions

$$f : \begin{cases} \{0,1\}^{127} \to & \{0,1\}^{128} \\ x \mapsto & \text{Poly1305-AES}_{(r,k,n)}(0,x) \end{cases}, \qquad g : \begin{cases} \{0,1\}^{127} \to & \{0,1\}^{128} \\ x \mapsto & \text{Poly1305-AES}_{(r,k,n)}(1,x) \end{cases}$$

satisfy for most of their inputs that $f(x + r) = g(x)$. For a larger r, we would need to shift one of the inputs to have a smaller shift.

Solving the hidden shift in $\mathbb{Z}/\left(2^{127}\right)$, using Kuperberg's second algorithm will cost around 2^{16} queries and 2^{47} classical time and 2^{40} memory. We can thus set the interval size at 2^{110}. As the domain of the functions is $\{0,1\}^{127}$, r can be retrieved if it is below 2^{127}. This is the case, as the bit constraints on r implies $r < 2^{124}$, we need only to test 2^{14} intervals. The total cost is then $2^{16} \times 2^{14} = 2^{30}$ queries, 2^{61} classical time and 2^{40} classical memory, for a success probability close to 1. We can check if the found r is the right one by trying to forge some valid messages, or we can use the distinguisher presented above.

7.7.3.4 Grover acceleration.

As the previous attack involves an exhaustive search on the correct interval among the 2^{14} ones, one might want to use Grover's algorithm, in order to gain up to 2^7 on the attack. We automatically lose a factor 2 because of the uncomputation of the algorithm. Moreover, we would need to compute all the qubit choices quantumly, and we must have a success probability of the inner function very close to one. All these factors make the attack more efficient in queries (around 2^{31}), but require a lot of quantum memory.

7.8 Sponges

We recall the design of a sponge in Figure 7.9. A collision in the c bits of the inner part allows to obtain a collision on the complete state, as the other part is controlled by the input. If P is a random permutation, the sponge construction is classically indistinguishable from a random oracle up to $2^{c/2}$ queries to P [Ber+08], which makes the previous attack optimal. More recently, it has been shown that a similar result holds with quantum access to the sponge [CHS19], up to $2^{c/3}$ queries to P, which corresponds to the quantum query cost of collisions.

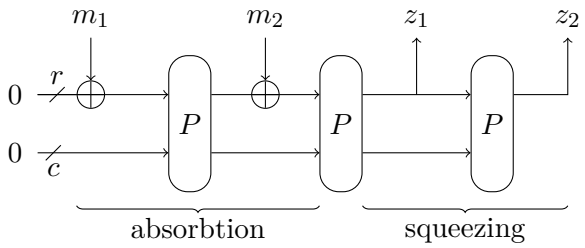


Figure 7.9: Sponge construction

Sponge Variants. Multiple variants of the sponge construction have been proposed. In particular, the *duplex* construction [Ber+12b] is quite popular for authenticated encryption, and is used by multiple candidates to the NIST lightweight competition [Aag+19; Dob+19; GMO19; Ber+19a; PM19; SMS19; Dae+19; Pen19]. Another change is the self-explanatory *full-state absorption* [Ber+12a], for keyed sponges. Both are described in Figure 7.10. Duplexing alternates between absorption and squeezing, which allows to encrypt, but also, as the state is modified by the messages, to authenticate with some additional squeezing. Duplex with full-state absorption is used by the NIST lightweight competition candidates SNEIK [Saa19] and Xoodyak [Dae+19]. Among the cited candidates, ACE, ASCON, Gimli and Xoodyak have been selected for the second round.

Quantum security. The inner part is especially annoying to mount a quantum attack on a sponge, as collision is still a hard problem with quantum computers. Moreover,

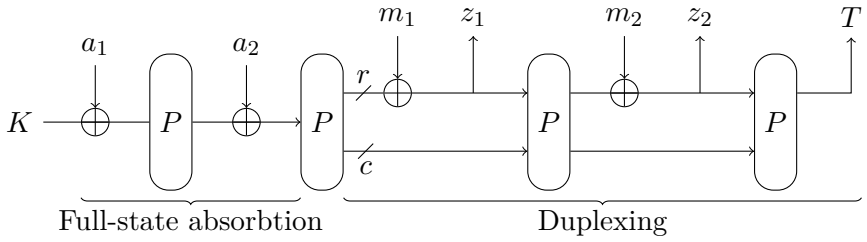


Figure 7.10: Duplex Sponge with full-state absorption

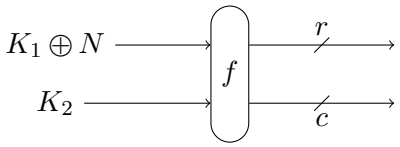


Figure 7.11: Beetle state initialization.

the period-based attacks would need to guess these c hidden bits of the state, which has a cost of $2^{c/2}$. As this is often the classical security claim, these constructions are often immune to this kind of attacks. However, this is no longer the case with full-state absorption, which is directly broken with quantum queries, as the function is of the form $f(x \oplus k)$. With classical queries, the cost of the attack is at least of $2^{(r+c)/3}$ if the whole state needs to be determined, but it can be lower if some parts are known. Moreover, we need to make all the queries to the same function, which can be prevented by a nonce change. However, in some cases, the nonce can be the variable of the function we query. We only need the encryptions of identical messages, with a set of nonces that fills an affine space. Nonce-respecting adversaries are generally allowed to choose the nonce, but here, the mere assumption that the nonce is incremented for each message (which is the standard way nonces are processed in practice) is sufficient: a set of 2^k consecutive values contains an affine space of $(\mathbb{Z}/(2))^{k-1}$.

Beetle. The Beetle sponge mode for lightweight authenticated encryption [Cha+18] is very similar to the duplex mode, with only a tweak in how the plaintexts are injected in the state of the sponge. It is used by some candidates to the NIST lightweight competition [Bao+19; Bei+19]. Its expected classical security is slightly below 2^c, which leaves some margin for a quantum attack. It has the initialization phase described in Figure 7.11. The nonce is directly injected on the key K_1, hence with quantum queries, we can recover K_1 and K_2 with around $2^{c/2}$ queries.

With classical queries, we can mount the same attack. Here, the nonce is directly added to the key K_1, but as the key has the same length as the state, the attack would still work if the nonce was added after. In Beetle[Light+], the rate is $r = 64$ bits and the capacity $c = 80$ bits. The rate is sufficiently large to embed 48 varying bits for the nonce; in that case, by making 2^{48} classical queries and 2^{48} Grover iterations, we can recover the secret $K_1||K_2$. In Beetle[Secure+], $r = c = 128$ bits. We can recover $K_1||K_2$ with 2^{85} messages and Grover iterations. This attack does not translate to the instances proposed in the NIST lightweight competition, which have a key smaller than the state, and whose initial state is $(N||K)$.

7.9 Protecting symmetric constructions

There are multiple approaches to thwart this kind of attacks.

Increasing the quantum cost. Simon's and Kuperberg's algorithm need the function to be close enough to a random function, and in particular, to not have some additional partial period that occurs with a high probability to be able to identify the secret. Hence, one way to prevent their application would be to make sure that such partial period exists. Unfortunately, this means that there exists a t such that, with a high probability $f(x \oplus t) = f(x)$ (or $f(x+t) = g(x)$) with f (and g) that depend on the primitive. This exhibits a weak differential property which may be used in a classical cryptanalysis.

Using a different group law. Another approach, which is proposed in [AR17], it to remark that the hidden shift attacks are subexponential at best. Hence, if the primitive has a large enough state, the attack will become impracticable. Their proposed patch is to change the group law involved in the hidden period. Using modular addition instead is the cheapest fix, but this will transform the corresponding hidden period problem into an abelian hidden shift problem, and the quantum algorithm would be in $2^{\sqrt{2n}}$, at best. Hence, if we only focus on the asymptotic exponent, we can obtain 64 bits of quantum security with a state of roughly 2048 bits, and 128 bits of quantum security with 8192 bits of state. This is much larger than the common state size in symmetric cryptography, and even larger than the 1600 bits of the state of SHA3 [SHA3]. Designing primitives with such a large state would be a work in itself, and may not be the most efficient approach.

In [AR17], it is also proposed to use a nonabelian group law in the primitive, and suggest the symmetric group, $\mathcal{S}_n$. This raises two issues: first, the cardinality of the group. Indeed, if the original design worked with a power of 2, a drop-in replacement would need to respect this cardinality, and this is not the case of the symmetric group. Second, such group laws are much more expensive to implement. If the operation is expensive to implement, one might ask if there is really a need to use a group law. The operation has only to be a key-dependent permutation, which is what a block cipher does. This leads to the final option to prevent this kind of attacks.

Using unaffected designs. If some classically safe designs become vulnerable with a quantum computer, this is not the case of all known constructions. In particular, well built iterated designs do not appear to be affected (except if their key schedules are weak, which is developed in Chapter 9), and multiple proven constructions can be used as-is. This is the approach chosen by the Saturnin team [Can+19], which is to date the only symmetric scheme designed with the aim to resist to this kind of attacks.

Chapter 8 Cryptanalysis of AEZ

AEZ [HKR15] is an authenticated encryption scheme designed by Hoang, Krovetz and Rogaway, and was a candidate of the CAESAR authenticated encryption competition. It aimed for fast and parallelizable encryption. Its security claims are in a very strong model: the cipher shall still be secure under nonce misuse and release of unverified plaintext, that is, if a fixed nonce is used or if the plaintext corresponding to an invalid tuple (ciphertext, tag) is known. AEZ has also an unusually low constraint on the amount of data to be processed with the same key: 2^{48} bytes.

This chapter presents AEZ [HKR15], the problem in AEZ version 4 we found with Patrick Derbez, Sébastien Duval, Jérémy Jean, Brice Minaud and Valentin Suder [Bon+17], a classical key-recovery cryptanalysis proposed by Colin Chaigneau and Henri Gilbert at FSE'17 [CG16], a generic quantum existential forgery attack proposed by Marc Kaplan, Gaëtan Leurent, Anthony Leverrier and María Naya-Plasencia [Kap+16] and the quantum key recovery that builds upon Chaigneau and Gilbert's attack we proposed at SAC'17 [Bon17]. Table 8.5 summarizes the known attacks on the different versions of AEZ.

Contents

8.1 Description of AEZ

AEZ [HKR15] is a tweakable block cipher for authenticated encryption, and its components have been tweaked in the different versions of the algorithm. It uses a master key K of 384 bits, decomposed in 3 subkeys (I,J,L) of 128 bits each. AEZ has at its core a tweakable function $E_K^{i,j}$ used in the intermediate function `AEZ-hash`. The user calls the external function Encrypt, that calls, depending on the message length, `AEZ-prf`, `AEZ-tiny` or `AEZ-core`. `AEZ-tiny` and `AEZ-core` are block ciphers, `AEZ-tiny` is used for messages of less than 32 bytes (one block), `AEZ-core` is used for longer messages. `AEZ-prf` is a pseudo-random function (PRF) called when the message is empty that takes some associated data and a length τ as arguments, and outputs a tag of the desired length that can be used to authentify the associated data. This is summarized in Figure 8.1.

Multiple versions of AEZ have been issued during the course of the CAESAR competitions, with some significant differences:

- AEZv2 changed the encryption mode to be faster and proposed the stronger mode AEZ10.
- AEZv3 introduced the definitive naming conventions, simplified the expression of some subkeys and changed the way the message was split for encryption.
  - AEZv3 is vulnerable to a collision attack by Fuhr, Leurent and Suder, at a cost beyond the security claim of 2^{55} blocks of data [FLS15].
- AEZv4 introduced the 384-bit key and the use of BLAKE2, added some offsets and changed their definitions to prevent the previous attack.
  - AEZv4 is vulnerable to a collision attack by Chaigneau and Gilbert, at a cost beyond the security claim of 2^{55} blocks of data [CG16].
  - An issue in the definition of the offsets allows for some trivial forgeries.
- AEZv5 patched the definition of the offsets, and overall simplified the design.

In the following part, we will describe AEZ versions 4 and 5. Our attacks will mainly use `AEZ-prf`, but we also need `AEZ-core` to retrieve one of the subkeys of AEZv4. Both are described below.

8.1.1 Associated data

The associated data is seen as a bidimensional vector of 128-bit blocks. An example for 7 blocks can be represented as:

$$\begin{matrix} A_1^1 \\ A_2^1 \\ A_3^1 A_3^2 A_3^3 \\ A_4^1 A_4^2 \end{matrix} \text{ that we note } (A_1^1, A_2^1, (A_3^1, A_3^2, A_3^3), (A_4^1, A_4^2)).$$

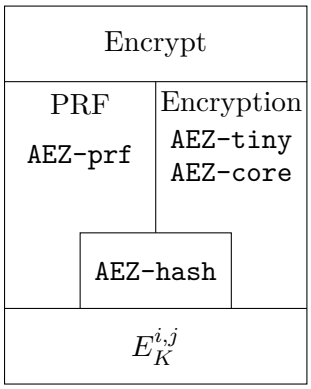


Figure 8.1: High-level view of the components of AEZ

Table 8.1: $E_K^{i,j}$ in AEZv4

i	j	$E_K^{i,j}(X)$
-1	$\mathbb{N}$	$\texttt{AES10}(X \oplus jJ)$
0	$\mathbb{N}$	$\texttt{AES4}(X \oplus jI)$
1	$\mathbb{N}$	$\texttt{AES4}(X \oplus \alpha_j I)$
2	$\mathbb{N}$	$\texttt{AES4}(X \oplus \alpha_j I)$ (This AES uses a different key schedule)
$\geqslant 3$	0	$\texttt{AES4}(X \oplus \beta_i L) \oplus \beta_i L$
$\geqslant 3$	$\geqslant 1$	$\texttt{AES4}(X \oplus \beta_i L \oplus \alpha_j J) \oplus \beta_i L \oplus \alpha_j J$

The associated data can contain any number of lines, and each line can have any length. In practice, we have two constraints. The first line A_1 contains the output length τ of the PRF, in bits. As we'll only have output lengths smaller than 2^{128} bits, the first line will only contain one block. The second line contains the nonce N. The specification recommends a nonce smaller than 128 bits, which also limits this line to one block.

8.1.2 Function $E_K^{i,j}$

The core of the algorithm is the function $E_K^{i,j}$, which is a permutation on 128 bits. It is concretely a tweaked version of 4 or 10 rounds of AES [DR02] (`AES4` and `AES10`). The exact function depends on the version of the algorithm and the values of i and j. These versions of AES don't use the normal key schedule but one of the subkeys (I, J, L) at each round.

Table 8.1 shows the value of $E_K^{i,j}$ in AEZv4, depending on the parameters i and j, with $\alpha_j = 2^{3+\lfloor (j-1)/8 \rfloor} \oplus ((j-1) \mod 8)$ and $\beta_i = 2^{i-3}$. The multiplication is done in the finite field $\mathbb{F}_{2^{128}}$, seen as $\mathbb{F}_2[X]/(X^{128} + X^7 + X^2 + X + 1)$.

The function is simpler in AEZv5:

- $E_K^{-1,j}(X) = \texttt{AES10}(X \oplus jL)$
- $E_K^{i,j}(X) = \texttt{AES4}(X \oplus iJ \oplus 2^{\lceil j/8 \rceil} I \oplus (j \mod 8)L)$

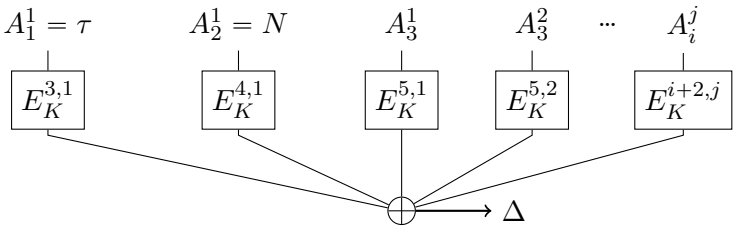


Figure 8.2: `AEZ-hash` scheme

For the variant AEZ10, the master key K has 128 bits and is directly used as an AES key to derive the subkeys.

$$I = \texttt{AES}_K(0), J = \texttt{AES}_K(1), \quad E_K^{i,j} = \texttt{AES}_K(X \oplus jI \oplus iJ)$$

8.1.3 `AEZ-hash`

This function takes as input the associated data A and the key K and outputs the 128-bit value Δ.

As presented in Figure 8.2, $\texttt{AEZ-hash}(K, A) = \Delta = \bigoplus_{i,j} E_K^{i+2,j}(A_i^j)$ in both v4 and v5.

8.1.4 `AEZ-prf`

This function is a pseudo-random function (PRF) of arbitrary output length which can be used to authentify the associated data. It takes as input an output length τ, some associated data A and the key K, and outputs τ bits.

It computes $\Delta = \texttt{AEZ-hash}(K, A)$, and outputs the first τ bits of the sequence $E_K^{-1,3}(\Delta)$, $E_K^{-1,3}(\Delta \oplus 1), E_K^{-1,3}(\Delta \oplus 2) \ldots$ The most interesting property of this function is that its value (for τ fixed) depends only on the value of `AEZ-hash`, and in particular, that a collision in `AEZ-hash` implies a collision in `AEZ-prf`.

8.1.5 `AEZ-core`

This function takes as input the hash Δ and a plaintext P of length greater than 256 bits. The plaintext is split in pairs of blocks of 128 bits $P||0^\tau = P_1||P_1'||P_2||P_2' ... P_u||P_v||P_x||P_y$. All the blocks are 128-bit long, except the second-to last pair P_u, P_v.

Then, a Feistel network that uses $E_K^{i,j}$ as a round function is applied. Each pair of plaintext blocks is encrypted almost independently: an intermediate value from each pair (X_i) is xored into the last pair's encryption intermediate state, then a mask S is computed from it, which is added to all the other intermediate states. Finally, another value (Y_i) is extracted from each encryption state and xored in the last pair. Following the notations of Figure 8.3, we have

$$X = \bigoplus_i X_i \quad Y = \bigoplus_i Y_i.$$

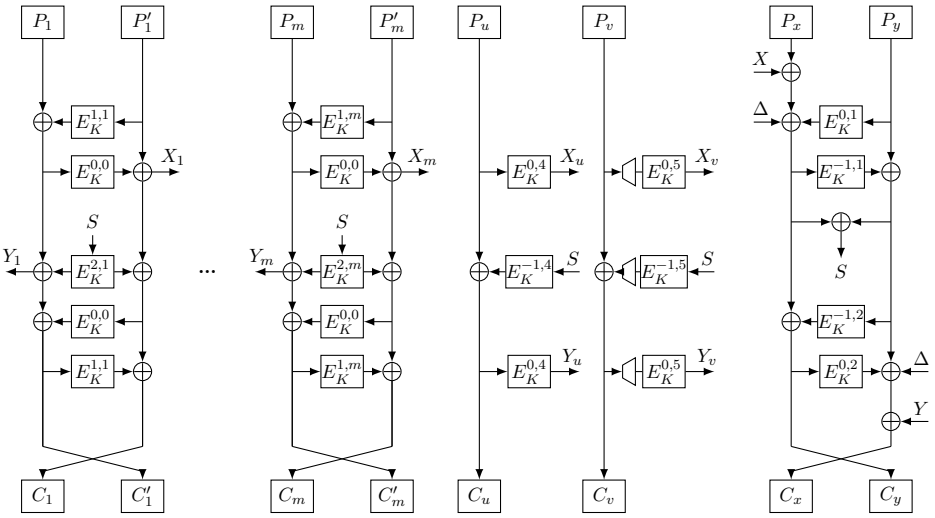


Figure 8.3: AEZ-core. Drawing based on the work of Colin Chaigneau [Jea16]

8.1.6 Encrypt

This function takes as input the key K, the associated data A and a variable-length message M. For empty messages, it is a direct call to $\texttt{AEZ-prf}(K, A, \tau)$. If $0 < |M| < 256 - \tau$, it calls `AEZ-tiny`. For longer messages, it calls $\texttt{AEZ-core}(\Delta, M||0^\tau)$, with $\Delta = \texttt{AEZ-hash}(K, A)$.

8.2 Classical cryptanalysis

This section presents the two works of (classical) cryptanalysis on AEZv4, namely a problem in the definition of its offsets, and the collision analysis proposed by Chaigneau and Gilbert.

8.2.1 The fault in AEZv4

The additional data is authenticated with the value $\Delta = \bigoplus_{i,j} E_K^{i+2,j}(A_i^j)$. Here, $E_K^{i,j}$ is the same primitive, with a block-dependent offset xored at the input and the output. In order to be secure, we need those offsets to be distinct, or else both positions would be equivalent: if we swap the input of two positions with the same offset, we would obtain the same Δ.

In AEZv4, the offset is $2^{i-3}L \oplus (2^{3+\lfloor (j-1)/8 \rfloor} \oplus ((j-1) \mod 8))J$. In particular, what interests us is the value $\alpha_j = 2^{3+\lfloor (j-1)/8 \rfloor} \oplus ((j-1) \mod 8)$. While j is small, $2^{3+\lfloor (j-1)/8 \rfloor}$ is a multiple of 8, hence there is no collision. The problem arises when $3 + \lfloor (j-1)/8 \rfloor$ reaches 128. In that case, the value resulting of the finite field modulo will not be a multiple of 8, and we can expect to use $(j-1) \mod 8$ to cancel a difference.

Table 8.2: Collision functions in [CG16]. lb() Outputs the last block of its input.

subkey	function	property
I	$f_I(x)$=lb(`AEZ-core`$(K,(\tau,N),(0,x,0,x,0,0)))$	$f_I(x)=f_I(x\oplus I)$
J	$f_J(x)$=`AEZ-prf`$(K,(\tau,N,(x,x)),\tau)$	$f_J(x)=f_J(x\oplus J)$
L	$f_L(x)$=`AEZ-prf`$(K,(\tau,x,x),\tau)$	$f_L(x)=f_L(x\oplus 6L)$

The first problematic offset is $j = 1001$, with $\alpha_{1001} = 2^7 + 4 + 2 + 1$. This turns out to be the same value as α_{40}. Hence, we have an existential forgery: one can swap on any string in the authenticated data the value of the blocks 40 and 1001 without altering Δ. A similar problem arises with higher values of j. The minimal cost for this attack is one query of roughly 1000 blocks.

The issue came from the fact that despite what the 3 might suggest, there is an overlap between the bits affected by $2^{3+\lfloor(j-1)/8\rfloor}$ and $(j-1) \mod 8$. This was easy to patch and led to AEZv5, which does not have such a problem.

8.2.2 The collision analysis of AEZv4

Chaigneau and Gilbert presented at FSE'17 a key-recovery attack on AEZv4 [CG16]. The attacker can query the functions of AEZ with a fixed unknown key, and chosen authenticated data and plaintexts. The attack is done in two parts: first, they apply 3 independent birthday sub-attacks that retrieve one of the 3 subkeys, and next they perform a differential attack that retrieves the 2 remaining subkeys once one is known. The first part needs a quantity of data at the birthday bound (2^{64} blocks), which is beyond the security claimed by AEZs designers, who limited the data to 2^{44} blocks for a given key. We'll describe here only that part, as it can be converted to an extremely efficient attack, while the differential attack does not gain as much, and becomes noncompetitive.

For each of the 3 attacks, they seek a collision in a specific function they construct from AEZ, and such a collision, with a high probability, will provide the value of a subkey if they xor the colliding inputs. The functions are described in table 8.2. The functions f_I and f_J need a fixed nonce N for each input, but not f_L, as for this function the nonce is set at the input value x.

For example, for f_L, the value of `AEZ-hash`$(K,(\tau,x,x))$ is $\Delta = E_K^{3,1}(\tau) \oplus E_K^{4,1}(x) \oplus E_K^{5,1}(x)$, which gives us, when we expand:

$$\Delta = E_K^{3,1}(\tau) \oplus \texttt{AES4}(x \oplus 2L \oplus 8J) \oplus \texttt{AES4}(x \oplus 4L \oplus 8J).$$

For $x = x' \oplus 6L$, we get

$$\Delta = E_K^{3,1}(\tau) \oplus \texttt{AES4}(x' \oplus 6L \oplus 2L \oplus 8J) \oplus \texttt{AES4}(x' \oplus 6L \oplus 4L \oplus 8J).$$

As we are in $\mathbb{F}_{2^{128}}$, it reduces to

$$\Delta = E_K^{3,1}(\tau) \oplus \texttt{AES4}(x' \oplus 4L \oplus 8J) \oplus \texttt{AES4}(x' \oplus 2L \oplus 8J).$$

Hence, we get the same Δ (which implies the same value of $f_L(x)$) if $x \oplus x' = 6L$, that is, $f_L(x) = f_L(x \oplus 6L)$.

The case of f_J is the same as f_L, but instead of considering two one-block strings, we consider one two-block string, and we have $f_J(x) = f_J(x \oplus J)$.

The case of f_I is slightly different, as it uses `AEZ-core`. The message $(0, x, 0, x, 0, 0)$ is split as $(P_1, P_1', P_2, P_2', P_x, P_y)$. Here, the only variable values are P_1' and P_2'.

The intermediate value X is

$$\texttt{AES4}(\texttt{AES4}(x \oplus 8I)) \oplus \texttt{AES4}(\texttt{AES4}(x \oplus 9I))$$

As before, if we find two values x, x' with $x = x' \oplus I$, then we obtain a collision on X.

It turns out that such a collision can be detected, as the value of the last block, C_y, depends only of Δ, X, P_x, P_y, and Δ, P_x, P_y are fixed here. Hence, we have $f_I(x) = f_I(x \oplus I)$.

Then, for f_L (and similarly for f_I and f_J), Chaigneau and Gilbert's attack is:

- Query $f_L(x)$ for 2^{64} different values of x.
- Search for a collision $f_L(x) = f_L(x')$
- With high probability, $x \oplus x' = 6L$.

All of these key recoveries have a cost at the birthday bound, which is above the security claim of the designers, who limited the total amount of bytes to be processed with the same key to 2^{48}.

8.3 Quantum cryptanalysis

The functions of Table 8.2 are of the form $f(x) = a \oplus g(x \oplus b) \oplus g(x \oplus b \oplus s)$, with g a xor of `AES4` with various inputs. Hence, Simon's algorithm (Algorithm 4.2) is applicable and can retrieve s, except if $s = 0$. In this case, f is a constant function, and Simon's routine (Algorithm 4.1) will always produce the value 0. Hence, it can easily be detected. Moreover, this is also easy to detect classically, as we can query classically these functions. It hence corresponds to a weak key. For the previous functions, it corresponds to the case where one subkey key is all 0. We can also craft some functions where s depends on multiple subkeys. An $s = 0$ would correspond to a case where two offsets are identical, which would also break the security of the scheme.

Simon's algorithm. As the block in AEZ are 128-bit long, we will only have to consider Simon's algorithm on functions on 128 bits. In order to obtain concrete values for the cost of Simon's algorithm, from Proposition 4.4 we need to bound the value

$$p_0 = \max_{t \notin \{0,s\}} \Pr_x[f(x \oplus t) = f(x)].$$

Here, f is based on 4 rounds of AES, and these additional periods correspond to a high differential on 4 rounds of AES. Hence, we can expect it to be fairly low. For

numeric applications, we considered that $p_0 < 1/8$. From Proposition 4.4, we obtain that after 192 queries, the failure probability is smaller than 2^{-30}. We based our numeric applications on this number of queries.

8.3.1 Quantum existential forgery

In [Kap+16], a quantum existential forgery is proposed: it simply consists in remarking that Δ is of the form

$$\Delta = \bigoplus_i E_K(A_i \oplus O_i).$$

Hence, if one queries quantumly $\texttt{AEZ-prf}(K, (\tau, N, x, x))$, the resulting function will have the period $p = O_3 \oplus O_4$, and can be retrieved with Simon's algorithm. From the value of p, one can easily make some forgeries, as the authenticated data (τ, N, A, B) and $(\tau, N, A \oplus p, B \oplus p)$ produces the same Δ.

8.3.2 Stronger quantum attacks

We presented the following attacks at SAC'17 [Bon17]. They build upon the classical analysis of Chaigneau and Gilbert to obtain a full key recovery on AEZv4 and AEZv5 and a universal forgery on AEZ10 at essentially the cost of the quantum existential forgery. Moreover, we adapted the classical attack to AEZv5 and the stronger variant AEZ10.

8.3.2.1 Key Recovery on AEZv4

We can directly use the functions of [CG16], described in table 8.2, in Simon's algorithm. There is however a slight difference for f_I, as the period is not on the full `AEZ-core` but only on the last block. This is not a problem, as we can compute truncated functions for free (see Section 2.4.3).

The complete attack is:

- For $k \in \{I, J, L\}$:
  - Query 192 times Simon's routine with f_k.
  - Solve classically the boolean equation system, get the period of f_k (I, J or $6L$).
  - If this period was a multiple of k, invert to retrieve k.

In the classical setting, $f_J(x) = \texttt{AEZ-prf}(K, (\tau, N, (x, x)), \tau)$ needed a nonce reuse. In the quantum setting, as the period is always J and does not depend on the nonce, Simon's algorithm can still be applied (see Section 4.2.3). The only constraint for the nonce is to be non-entangled with the input value. The situation is the same for f_I. For $f_L(x) = \texttt{AEZ-prf}(K, (\tau, x, x), \tau)$, the nonce was chosen by the attacker as part of the input. Hence, the quantum query would require a call with a nonce in superposition.

Table 8.3: Collision functions for AEZv5

subkey	function	Period
I	$f_I(x)$=AEZ-prf$(K,(\tau,N,(x,A,B,C,D,E,F,G,x),\tau)$	$6I$
J	$f_J(x)$=AEZ-prf$(K,(\tau,N,x,x),\tau)$	$3J$
L	$f_L(x)$=AEZ-prf$(K,(\tau,N,(x,x)),\tau)$	$3L$

If we want to only allow classical nonces, we can use f'_L = AEZ-prf$(K,(\tau,N,x,x),\tau)$, which satisfies $f'_L(x) = f'_L(x \oplus 12L)$, and processes one more block than f_L per query.

Overall this costs $576 = 2^{9.2}$ queries, for a total of $3456 = 2^{11.8}$ blocks.

But we can go even further, if we look at

$$f_{JL}(x) = \texttt{AEZ-prf}(K,(\tau,N,(x,x),(x,x)),\tau).$$

The associated Δ is

$$\begin{aligned} A \oplus \texttt{AES4}(x \oplus 4L \oplus 8J) \oplus \texttt{AES4}(x \oplus 4L \oplus 9J) \\ \oplus \texttt{AES4}(x \oplus 8L \oplus 8J) \oplus \texttt{AES4}(x \oplus 8L \oplus 9J)\end{aligned}.$$

This function has a hidden period of J and $12L$. Hence, from Proposition 4.2, we can apply Simon's Algorithm to retrieve the vector space $\langle J, 12L\rangle$. J and $12L$ need to be independent for the function to be non-constant.

In that case, we can retrieve the value of J and L with an exhaustive search, as there are only 3 non-zero values in a 2-dimensional vector space. Hence, there is only 6 possible values for J and L.

This diminishes even more the query complexity to $384 = 2^{8.6}$ and the number of blocks processed to $2880 = 2^{11.5}$.

8.3.2.2 Key Recovery on AEZv5

The functions in table 8.3 allow to perform the same attack on AEZv5, with a quantum query complexity of $2^{9.2}$, and a data complexity of $4224 = 2^{12.0}$ blocks.

We can even be more efficient in queries and recover the vector space $\langle 6I, 3J, 3L\rangle$ in one go, with the function

$$\begin{aligned} f_{IJL}(x) = \texttt{AEZ-prf}(K,(\tau,N,\ &(x,x,B,C,D,E,F,G,x,x),\\ &(x,x,B',C',D',E',F',G',x,x)\ ),\tau). \end{aligned}$$

Here, any non-x value in argument can be anything as long as it is not entangled with x. This f has the 3 periods of f_I, f_J and f_L, and allows us to recover the vector space $\langle 6I, 3J, 3L\rangle$ in 192 queries, and the same data complexity as before.

Once we know $\langle 6I, 3J, 3L\rangle$, we can simply perform an exhaustive search on all of its basis. There is only $7 \times 6 \times 4 = 168$ possible basis, and we can classically check them from a few classical queries.

Using the same principle, we can also define and use f_{IJ}, f_{JL} or f_{IL}, which all have comparable costs,

$$f_{IL}(x) = \texttt{AEZ-prf}(K,(\tau,N,(x,x,B,C,D,E,F,G,x,x),\tau)$$

and $f_J(x)$ giving the best data complexity of $3264 = 2^{11.7}$ blocks.

8.3.2.3 Universal forgery on AEZ10

The core function is even simpler in this variant: $E_K^{i,j}(X) = \texttt{AES}(X \oplus iJ \oplus jI)$. Hence, we can do the attack with the functions in table 8.4. With two functions, we can recover

Table 8.4: Collision functions for AEZ10

subkey	function	period
I	$f_I(x)$=$\texttt{AEZ-prf}(K, (\tau, N, (x, x)), \tau)$	$3I$
J	$f_J(x)$=$\texttt{AEZ-prf}(K, (\tau, N, x, x), \tau)$	$3J$
I, J	$f_{IJ}(x)$=$\texttt{AEZ-prf}(K, (\tau, N, (x, x), (x, x)), \tau)$	$3I, 3J, 3I \oplus 3J$

I and J in 384 quantum queries and 1920 blocks of quantum data. If we choose to get the vector space spawned by I and J, we only need 192 queries and $1344 = 2^{10.4}$ blocks of data. In this case, we don't get a full key recovery, but the knowledge of the tweaks I and J allows to make forgeries for any non-empty authenticated data.

Classical queries. As the previous attacks use Simon's algorithm, it would be interesting to use the methods of Chapter 6 to obtain an attack with classical queries, which would be closer to the security model considered by the designers. This however does not appear to be possible, as the periodic functions of AEZ do not have a publicly computable part.

8.3.2.4 Classical Versions of the attacks

All the attacks proposed in this section have a classical variant close to the attack from [CG16].

Their cost will be at the birthday bound (2^{64} queries). We can use the multiple periods to reduce the number of queries, but as the multi-periodic functions requires more blocks than the single-period functions, it doesn't change much on the overall complexity. For AEZv5, we need a total of 16 blocks per queries, hence the total data, time and memory cost is 2^{68}. We only need 6 blocks per queries for AEZ10, hence the total cost is $2^{66.6}$.

As remarked in [CG16], within the limits in data, the success probability of the classical attacks is around 2^{-45}. Hence, if we consider that a key is changed every 2^{44} blocks, we can expect to have one successful attack after a grand total of around 2^{90} block processed over all the users of AEZ.

8.4 Conclusion

This chapter presented the known attacks on AEZ from version 4, which are summarized in Table 8.5. If AEZv4 had a problem that made it trivially broken, the known attacks

Table 8.5: Summary of the cryptanalyses on AEZ since version 3. The cost corresponds to the time, memory and number of blocks of data.

Version	Cost	model	Type	Ref
AEZv3	$2^{66.6}$	Classical	Key Recovery	[FLS15]
AEZv4	$2^{66.6}$		Key Recovery	[CG16]
AEZv4	2^{10}		Existential Forgery	Section 8.2.1
AEZv5	2^{68}		Key Recovery	Section 8.3.2.4
AEZ10	$2^{66.6}$		Universal Forgery	Section 8.3.2.4
All	$\simeq 2^{9.6}$	Quantum query	Existential Forgery	[Kap+16]
AEZv4	$2^{11.5}$		Key Recovery	Section 8.3.2.1
AEZv5	$2^{11.7}$		Key Recovery	Section 8.3.2.2
AEZ10	$2^{10.4}$		Universal Forgery	Section 8.3.2.3

on AEZv5 and AEZ10 do not contradict their security claims. Indeed, either the data complexity is too high ($\simeq 2^{64}$ blocks, with a limit at 2^{44}), or it is extremely small, but with a different security model. The designers considered classical adversaries and queries, and the quantum attacks require a quantum oracle.

For quantum cryptanalysis, AEZ shows that some instances of Simon's problem have a hidden subgroup, and not only a hidden period. Moreover, AEZ demonstrates that some classical properties can be transferred quantumly. Indeed, as AEZ was not designed to be beyond-the-birthday-bound secure, some forgeries at the birthday bound were to be expected. It turned out that at the birthday bound, the cipher is totally broken, as there is a full key recovery. Something similar happened with the quantum attacks, with a simple existential forgery attack and a full key recovery that both use Simon's algorithm, and have a very similar cost, in polynomial time.

AEZ aimed at offering some extremely strong security guarantees (remaining secure even with the release of some unauthenticated plaintexts), while still being fast. This came with some drawbacks. First, AEZ has a complex design that has evolved with its multiple iterations. This has made it hard to analyze and has turned out to be error-prone, as seen with AEZv4. Second, the strong guarantees completely collapses if we reach the bounds of the security proof, at 2^{64} blocks. These two issues may explain why AEZ was not selected to be a finalist in the CAESAR competition.

Chapter 9 Quantum Slide Attacks

Slide attacks [BW99] are a cryptanalysis family proposed by Biryukov and Wagner aiming to analyze self-similar iterated block ciphers. Multiple classical variants aiming to analyze different constructions have been proposed since [BW00; BDK07; DKS15; Din+15; Bar+18]. These attacks have the peculiar property that their cost does not depend on the number of rounds of the iterated cipher. They showed the importance of having distinct rounds in a cipher (by using round constants or a key schedule, for example) and avoiding a periodic structure between the rounds of an iterated construction.

Quantum slide attacks are much more recent. The first one has been proposed by Marc Kaplan, Gaëtan Leurent, Anthony Leverrier and María Naya-Plasencia at CRYPTO 2016 [Kap+16], where a particular case of a construction considered in [BW99] is broken with Simon's algorithm. This was the first example of an exponential quantum speedup of a cryptanalysis technique.

The contents of this chapter mainly consist in the results I obtained with María Naya-Plasencia and André Schrottenloher [BNS19a], which were presented at SAC 2019. These quantum slide attacks are inspired from the classical ones from [BW00; DKS15; Din+15; Bar+18]. It also presents the original quantum slide attack from [Kap+16] and its generalization to arbitrary key-alternating ciphers in a related-key model proposed by Hosoyamada and Aoki [HA17]. Some of the results from [BNS19a] were also independently obtained by Dong, Dong and Wang [DDW18]. The classical and quantum attacks on some versions of MiMC and GMiMC described in Section 9.6 come from the note [Bon19a]. The attacks are summarized in Table 9.2.

All the attacks presented in this chapter require quantum queries, except for two attacks on self-similar Feistel ciphers in Section 9.3.2 which are generalized and applied to some versions of MiMC and GMiMC in Section 9.6.

Contents

9.1 Classical slide attacks

Slide attacks consider a cipher $E_k : \{0,1\}^n \to \{0,1\}^n$, constructed from a family of round functions $F_{k_i}(x) : \{0,1\}^n \to \{0,1\}^n$ applied r times. Each round uses a subkey $k_1, \dots k_r$ derived from the master key k of the cipher.

Assumption. We assume that F_k is a *weak* function, in the sense that given a a few equations of the form $F_k(x_i) = y_i$ for a given key k, it is computationally easy to retrieve k. This notion of weakness is broader in a quantum setting, as presented in Section 9.3.

Basic slide property. Suppose that all the round subkeys are equal: $k_1 = k, \dots k_r = k$; i.e the scheme is *one-round self-similar*. From the structure of the cipher, which is r similar applications of the same permutation F_k, we may write a simple equality, the *slide property*:

$$E_k(F_k(x)) = F_k(E_k(x)) \tag{9.1}$$

Basic slide attack. The goal of the attacker is to find two pairs x, y satisfying $F_k(x) = y$. The birthday paradox implies that, among $\mathcal{O}\left(2^{n/2}\right)$ plaintext-ciphertext couples P, C, there exists a *slid pair*: P_0, C_0 and P_1, C_1 such that $F(P_0, k) = P_1$. In that case, we also have: $F(C_0, k) = C_1$. If the round function is weak, the two equations are sufficient to retrieve k. In order to have an efficient attack, we also need to be able to identify the correct pair among the 2^n possible ones faster than exhaustive search.

Example of weak round function. One can consider a keyed permutation from a public permution Π: $F_k(x) = k \oplus \Pi(x)$. This is shown in Figure 9.1. The useless first application of Π is omitted, but we can consider $P'_i = \Pi(P_i)^{-1}$ to obtain a strict self-similar cipher. In that case, a slid pair $(P_0, C_0), (P_1, C_1)$ satisfies $P'_1 = F_k(P'_0)$, which corresponds to $P_1 = \Pi(k \oplus P_0)$. This is equivalent to $C_1 = k \oplus \Pi(C_0)$. Hence it suffices

to check if $\Pi(P_1)^{-1} \oplus C_1 = P_0 \oplus \Pi(C_0)$. The secret key can be trivially recovered from the slid pair, as $k = C_1 \oplus \Pi(C_0)$.

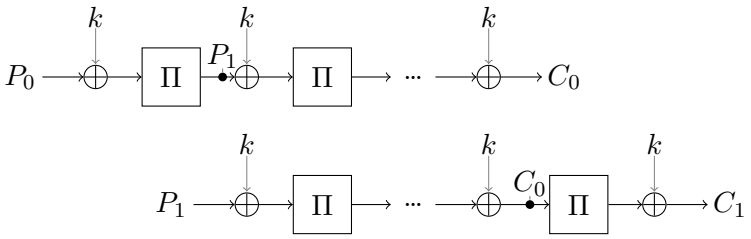


Figure 9.1: Example of slide attack on a cipher with a weak round function. The first application of Π is omitted.

In [BW99], slide attacks were applied to the TREYFER cipher, variants of DES and Blowfish, and some Feistel constructions. In all cases, these attacks have a cost at the square root of the exhaustive search of the recovered secret.

9.2 Slide-shift attacks

Classical slide attacks rely on a collision to break the cipher. It turns out that in many (but not all) cases, this collision can be written as a hidden shift, that we call the *slide-shift* property. This additional property can be leveraged with the corresponding quantum algorithm (from Chapter 4 or Chapter 5), with quantum queries to the primitive. This adds some constraints on the targets: the quantum attack depends on some parameters that were of little importance in the classical case, such as the way the key is added. In general, we describe the attack on the variant that uses modular additions, as XORs are a special case.

Cost estimates. Slide-shift attacks use a hidden shift algorithm to gain in efficiency. From Chapter 4, we estimate a cost of n queries and time for n-bit XOR-based slide-shift equations. For addition-based equations, many tradeoffs are available; we consider here a cost of $2^{\sqrt{2n}}$ quantum queries, quantum time and classical memory, for n-bit addition-based slide-shift equations, from Chapter 5. If we need a reversible variant of the algorithm, to be used by another quantum algorithm, we consider that the query cost is quadrupled (doubled to take into account the uncomputing, and doubled to have a failure probability exponentially small).

In this section, we first present the original quantum slide attack against 1-round self similar key-alternating cipher proposed in [Kap+16], and its multiple generalizations: to modular key additions, to arbitrary key schedules in a specific related-key model [HA17] and to 2-round self-similarity. Next, we present new advanced quantum slide attacks on Feistel networks, with slide attacks based on one-round self-similarity and advanced sliding techniques from [BW00], applied to Feistel variants.

9.2.1 Key-alternating cipher

The 1-round self-similar key-alternating cipher is presented in Figure 9.1, with a public permutation Π and a repeated secret key k. We can define the function G as in [Kap+16]:

$$G : \{0,1\} \times \{0,1\}^n \to \{0,1\}^n$$
$$b, x \quad \mapsto \begin{cases} g_0(x) = \Pi(E_k(x)) \oplus x & \text{if } b = 0, \\ g_1(x) = E_k(\Pi(x)) \oplus x & \text{if } b = 1. \end{cases}$$

As in the previous attack, we know that all x satisfy $\Pi(E_k(x)) \oplus k = E_k(\Pi(x \oplus k))$ because of the sliding property. Then we can see that G verifies the conditions of the hidden shift problem as $g_0(x) = g_1(x \oplus k)$:

$$G(0,x) = \Pi(E_k(x)) \oplus x = E_k(\Pi(x \oplus k)) \oplus k \oplus x = G(1, x \oplus k).$$

Hence, k can be recovered in n quantum queries and time.

With modular additions. Modular key additions do not change fundamentally the slide-shift property, and in that case, we have $\Pi(E_k(x)) + k = E_k(\Pi(x + k))$, which allows to define:

$$G : \{0,1\} \times \{0,1\}^n \to \{0,1\}^n$$
$$b, x \quad \mapsto \begin{cases} g_0(x) = \Pi(E_k(x)) - x & \text{if } b = 0, \\ g_1(x) = E_k(\Pi(x)) - x & \text{if } b = 1. \end{cases}$$

We can see that G verifies the conditions of the hidden shift problem as $g_0(x) = g_1(x+k)$:

$$G(0,x) = \Pi(E_k(x)) - x = E_k(\Pi(x + k)) - k - x = G(1, x + k).$$

In that case, k can be recovered in around $2^{\sqrt{2n}}$ quantum queries, time and classical memory. Alagic and Russel [AR17, Appendix B.2] have proposed this as an efficient counter-measure to the Simon's attack: as it is not polynomial-time, a primitive with large enough state will be hard to break. This is discussed in Section 7.9.

Arbitrary key schedule, in a related-key setting [HA17]. Hosoyamada and Aoki have remarked that the previous attack can also be applied to an arbitrary key-alternating cipher with quantum access to a pair of ciphers $E_{k_1,\dots,k_r}$ and $E_{k_2,\dots,k_{r+1}}$, as presented in Figure 9.2.

In that case, we have a very similar slide property:

$$\Pi(E_{k_1,\dots,k_r}(x)) \oplus k_{r+1} = E_{k_2,\dots,k_{r+1}}(\Pi(x \oplus k_1)) \quad .$$

Hence, we can define the function:

$$f(x) = \Pi(E_{k_1,\dots,k_r}(x)) \oplus E_{k_2,\dots,k_{r+1}}(\Pi(x)).$$

This function has the period k_1. From k_1, the slide property allows to compute $k_{r+1} = \Pi(E_{k_1,\dots,k_r}(x) \oplus E_{k_2,\dots,k_{r+1}}(\Pi(x \oplus k_1))$. The other keys can be recovered if the functions $E_{k_{1+i},\dots,k_{r+i}}$ are also accessible. In that case, it is not possible to have a secure key-alternating cipher.

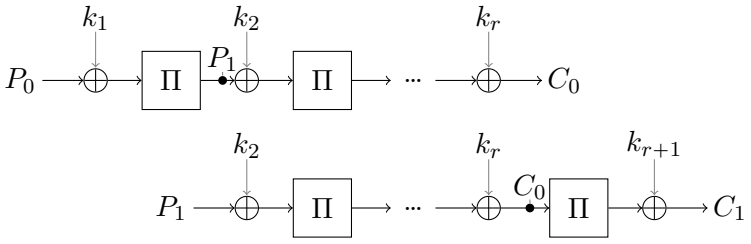


Figure 9.2: Quantum related-key slide attack [HA17].

2 rounds self-similarity. A similar attack can be made if only two keys k_1, k_2 are used alternatively. We can see this construction as a 1-round self-similar cipher with the permutation $\Pi'(x) = \Pi(k_2 \oplus \Pi(x))$. As this is not a public permutation, we need to combine the attack with a quantum search on k_2. We can use the offline Simon's algorithm to reduce the number of queries, and the total cost is then roughly $2n$ quantum queries and $n2^{n/2}$ quantum time.

9.2.2 Feistel schemes with one round self-similarity

We consider from now on Feistel schemes, like the one represented in Figure 9.3. We denote by E_k the encryption function, and by $Trunc_L$ and $Trunc_R$ the functions that truncate a Feistel state to its left or right part respectively. These functions are free to compute quantumly, as shown in Section 2.4.3.

For a Feistel construction, if we consider a slide attack over one round, the right part of the first plaintext R (left side in Figure 9.3) will be the same as the left part of the second plaintext, L'. Classical adversaries could use a fixed right part, and take random plaintexts for the left part. For our quantum attack, we fix a known value $R_0 = R = L'$. We can consider the variable $k' = f_k(R_0)$, represented on Figure 9.3, as an equivalent key, and this will be the value retrieved by the hidden shift algorithm.

We use the following function:

$$\begin{aligned} G : \{0,1\} \times \{0,1\}^{n/2} &\to \{0,1\}^{n/2} \\ b, x \quad &\mapsto \begin{cases} g_0(x) = Trunc_R(E_k(x, R_0)) & \text{if } b = 0, \\ g_1(x) = Trunc_L(E_k(R_0, x)) & \text{if } b = 1. \end{cases} \end{aligned}$$

From Figure 9.3 we can verify the slide shift equation $g_0(x) = g_1(x + k')$. By applying a hidden shift algorithm we will recover the value of $k' = f_k(R_0)$ for the value of R_0 that we fixed in the beginning. Since we know R_0, we can retrieve the actual value of k when f is weak. The cost of this attack with modular additions is $2^{1.2\sqrt{n}}$. With XORs, the analysis is quite similar, and the time complexity is reduced to n plus a constant. This basic attack is generalized to quantumly weak functions in Section 9.3.

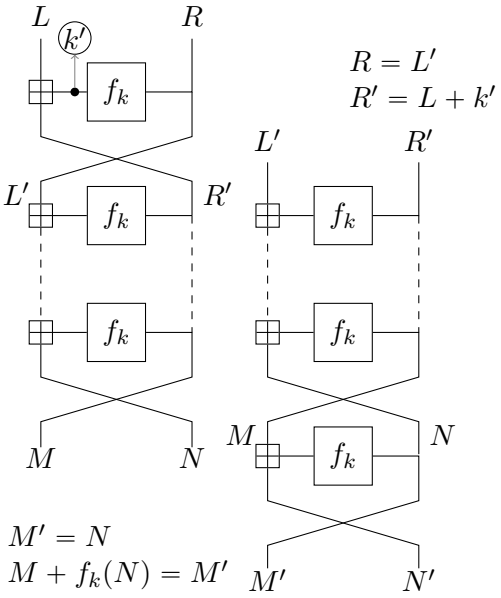


Figure 9.3: Slide attack on Feistel scheme with one round self-similarity

9.2.3 The quantum complementation slide attack

We illustrate the quantum *complementation slide attack*, that was originally proposed in a classical setting [BW00], on a Feistel cipher with 2-round self-similarity and a round function of the form $f(x+k)$. The main idea of this attack, described in Figure 9.4, is to slide by one round only. Though this implies that the round-keys of the middle rounds are not the same, this can be compensated by adding a relative difference between the complemented slide pairs. As can be seen on the figure, if we denote by $\Delta = k_0 - k_1$ and if the following equations implying a slid pair are verified: $R = L' - \Delta$ and $L + f(R + k_0) = R' + \Delta$, then, the outputs of both plaintexts, (M, N) and (M', N') verify $N = M' - \Delta$ and $N' = M + f(k_1 + N + \Delta) - \Delta$. With such inputs, all the transformations through the f functions will be the same pairwise through all the rounds for a slid pair. The combination of both halves ensures that the input round difference stays as wanted.

We perform an exhaustive search on Δ. The size of a block and of the whole key is n, while the size of Δ is $n/2$. Considering the value of R fixed to a chosen and known value, we can define an equivalent round-key $k'_0 = f(R + k_0)$ and when we recover k'_0 we can deduce k_0 if f is weak. The exhaustive search can be combined with Kuperberg's algorithm applied to the following function (keeping in mind that R and Δ are therefore known):

$$G : \{0,1\} \times \{0,1\}^{n/2} \to \{0,1\}^{n/2}$$
$$b, x \quad \mapsto \begin{cases} g_0(x) = Trunc_R(E_k(x, R)) + \Delta & \text{if } b = 0, \\ g_1(x) = Trunc_L(E_k(R - \Delta, x + \Delta)) & \text{if } b = 1. \end{cases}$$

From Figure 9.4 we can verify that $g_0(x) = g_1(x + k'_0)$. For each of the tested values for

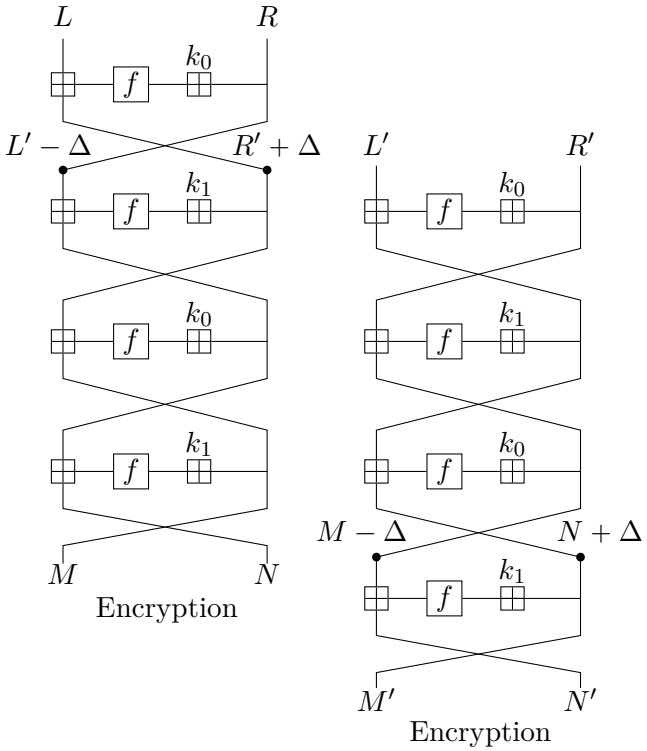


Figure 9.4: Complementation slide attack

Δ, by applying Kuperberg's algorithm we will recover the value of k'_0 for the fixed value of R that we fixed in the beginning, R_0. From k'_0 and R_0 we directly recover k_0 because f is weak, and with Δ, this implies the value of k_1. When the tested value for Δ is the correct one, we should also obtain a collision given by $N' = M + f(k_1 + N + \Delta) - \Delta$, which happens with a random probability $2^{-n/2}$. When this is the case, this implies that we have recovered the correct values of k_0 and k_1.

The cost for this if all the transformations were XORs would be of $n2^{n/4}$ quantum time and superposition queries, compared to $2^{n/2}$ from the quantum accelerated exhaustive search of the key. If we have modular additions instead, the cost becomes $2^{\sqrt{n}+n/4}$ quantum time and superposition queries, which is still better than generic exhaustive search in $2^{n/2}$. We show in Section 9.3 how to do better if the key and branch addition are identical.

Whitened Feistel. A variant of the attack can be performed if the Feistel is whitened with an additional input key k_{pre} and an output key k_{post}. We split the whitening key in they left and right part: $k_{pre} = k^L_{pre}||k^R_{pre}$. This whitening simply transforms the equations we had:

$$g_0(x) = Trunc_R(E_k(x - k^L_{pre}, R - k^R_{pre})) + \Delta - k^R_{post}$$
$$g_1(x) = Trunc_L(E_k(R - \Delta - k^L_{pre}, x + \Delta - k^R_{pre})) - k^L_{post}$$

If we define $\Delta' = \Delta + k^L_{pre} - k^R_{pre}$, subtract k^L_{pre} from x and k^R_{pre} from R, the two functions become

$$g_0(x) = Trunc_R(E_k(x, R)) + \Delta' - k^L_{pre} + k^R_{pre} - k^R_{post}$$
$$g_1(x) = Trunc_L(E_k(R - \Delta', x + \Delta')) - k^L_{post}$$

Given Δ', we cannot directly compute these two functions. However, we can circumvent this issue by adding an independent variable y to the output, that is, consider

$$g_0(x, y) = Trunc_R(E_k(x, R)) + \Delta' + y$$
$$g_1(x, y) = Trunc_L(E_k(R - \Delta', x + \Delta')) + y$$

This pair of function satisfy $g_0(x, y) = g_1(x + k'_0, y + k^L_{pre} - k^R_{pre} + k^R_{post} - k^L_{post})$. Hence, we can perform as before a quantum search on Δ' with these functions. From $k'_0 = f(R + k_0)$, k_0 can be extracted, in at worst $2^{n/4}$ operations. Once k_0 is known, k_1 can be recovered by applying the attack against the FX construction of Chapter 7. Overall, the cost is essentially the same as without any whitening.

9.2.4 Sliding with a twist

A further improved variant of the slide attacks is the *sliding with a twist* technique, also introduced in [BW00], that can be applied against some Feistel constructions (Figure 9.5). The quantum version works as long as the two branches are added with a XOR. We will describe the attack considering key insertions and round combinations by XOR, the addition of the key being irrelevant for the complexity.

The key idea is that encryption of a two-round self similarity Feistel cipher is a slid version of its decryption, modulo the final twists, that are easily taken into account. If we consider the inputs and outputs of the encryption (L, R), (M, N) and the inputs and outputs of the decryption $(M'N')$, (L', R'), we have that if $R = N'$ and $M' = L \oplus f(R \oplus k_0)$, then $R' = N$ and $L' = M \oplus f(N \oplus k_0)$.

We can consider now $R = N'$ as a fixed chosen value. Like the previous attack, if we consider an equivalent key $k'_0 = f(R \oplus k_0)$, we can apply Simon's algorithm. Let us denote the decryption function D_k.

$$G : \{0,1\} \times \{0,1\}^{n/2} \to \{0,1\}^{n/2}$$
$$b, x \quad \mapsto \begin{cases} g_0(x) = Trunc_R(E_k(x, R)) = N & \text{if } b = 0, \\ g_1(x) = Trunc_R(D_k(x, R)) = R' & \text{if } b = 1. \end{cases}$$

From Figure 9.5 we can verify the slide shift equation $g_0(x) = g_1(x \oplus k'_0)$. Simon's algorithm recovers the value of k'_0, and from it, also the one of k_0 with negligible complexity because f is easy to invert. Once k_0 is known, we can repeat a similar attack peeling off one layer in order to recover also k_1 with comparable complexity.

The cost when all the transformations are XORs is n, compared to $2^{n/2}$ for the quantum accelerated exhaustive search. If we have modular additions between branches, this attack does not apply, as the decryption scheme has subtractions instead of additions.

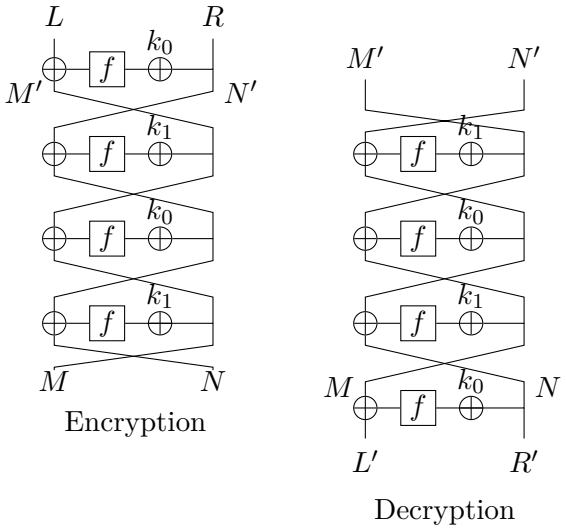


Figure 9.5: Representation of the sliding with a twist technique

9.3 Advanced slide-shift attacks on self-similar Feistels

In this section, we propose efficient key-recovery attacks on self-similar Feistel constructions with a classically strong round function.

In what follows, we show how to construct a quantum oracle to the round function given access to a quantum oracle to the full primitive, which allows to break it when the round function is only quantumly weak. Next, we show a more efficient attack when the round function has the form $f(x + k)$ and the branch and key additions are the same, without any weakness assumption on f.

9.3.1 General attack

In the attacks presented in the previous section, the round function was supposed to be weak. Indeed, the quantum Hidden Shift algorithm recovers the equivalent round key $k' = f_k(R_0)$. We are left with the task of finding k. Classical exhaustive search requires time $2^{n/2}$, Grover's algorithm requires time $2^{n/4}$. But there are some common cases in which we can quantumly attack the round function. In the previous attacks, the value of R_0 was fixed to a random chosen value. This means that we can repeat the same operation for several values R_i.

Case f(x+k). We consider here that the round function is a key addition composed with a public function f, as this is a fairly common design. We use the attack of Section 9.2.2, which, given a value R_0, produces $f(R_0+k)$. This attack can be considered as a circuit that takes an input x, and produces $f(x + k)$. Hence, this effectively implements a quantum oracle to the function $H : x \mapsto f(x + k)$. Given this function,

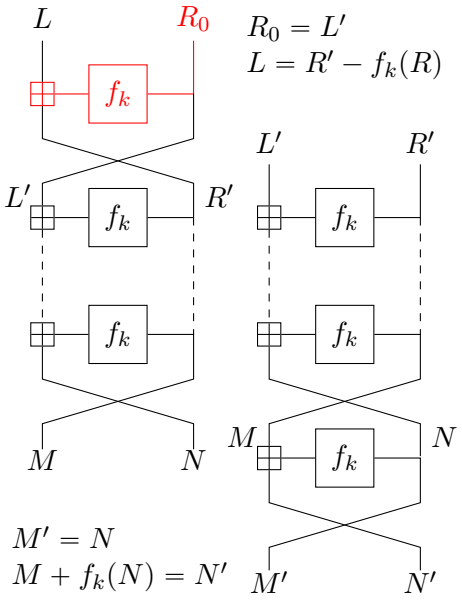


Figure 9.6: Slide attack on a generic one-round self-similar Feistel

we can construct the following function G, whose hidden shift is k:

$$G : \{0,1\} \times \{0,1\}^{n/2} \to \{0,1\}^{n/2}$$
$$b, x \quad \mapsto \begin{cases} g_0(x) = H(x) = f(x+k) & \text{if } b = 0, \\ g_1(x) = f(x) & \text{if } b = 1. \end{cases}$$

Indeed, we have $G(0,x) = G(1,x+k)$. Each call to G requires to solve a hidden shift instance reversibly. Hence, the cost is multiplicative. With xors only, it will be around n^2 queries. The cost of these attacks is summarized in Table 9.1.

Generalization. We described the attack for a Feistel round function of the form $F(x,k) = f(x+k)$, but we can use this for any keyed function vulnerable to quantum key recovery, like the Even-Mansour construction in Figure 9.7. The total cost of the attack is the cost to attack the round function multiplied by the cost of the slide attack. Hence, the Feistel structure does little to increase the security compared to the round function.

For the example in Figure 9.7, the exhaustive search is classically in $2^{n/2}$, quantumly in $2^{n/4}$. The classical slide attack can find a pair $(x, f_k(x))$ in $2^{n/4}$ queries. Once this is done, we can easily retrieve some other slide pairs using the slide pairs at the output of the cipher, and do the classical slide attack to break Even-Mansour in $2^{n/4}$ more queries and time. In comparison, the quantum slide attack performs in around $n2^{\sqrt{n}}$ queries.

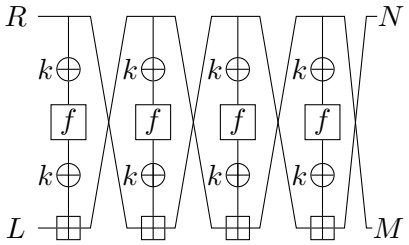


Figure 9.7: Example of a vulnerable Feistel

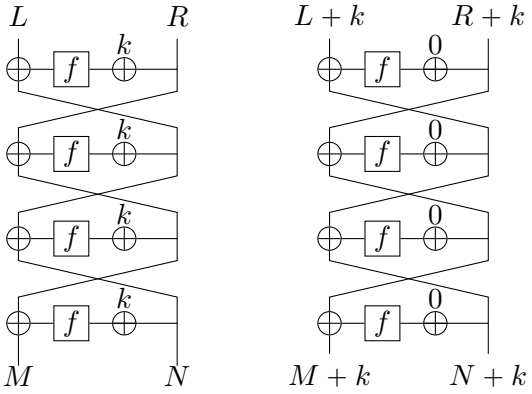


Figure 9.8: Slide attack with identical key and branch addition.

9.3.2 With the same branch and key addition

If the round function is $f(x+k)$ and the branch and the key addition both use the same law, the attack can be made more efficient. We can consider a Feistel construct E_k composed of a certain number of iterations of the function $f_k(x,y) = y, x + f(y+k)$. In that case, we have $f_k(x,y) + (k,k) = f_0(x+k, y+k)$, hence $E_k(x,y) + (k,k) = E_0(x+k, y+k)$, as described in Figure 9.8. We can then consider this function, with hidden shift k:

$$G : \{0,1\} \times \{0,1\}^{n/2} \to \{0,1\}^n$$
$$b, x \quad \mapsto \begin{cases} g_0(x) = E_k(x,x) - (x,x) & \text{if } b = 0, \\ g_1(x) = E_0(x,x) - (x,x) & \text{if } b = 1. \end{cases}$$

Indeed, $G(0,x) = E_k(x,x) - (x,x) = E_0(x+k, y+k) - (k,k) - (x,x) = E_0(x+k, y+k) - (x+k, x+k) = G(1, x+k)$.

Hence, with XORs, this attack costs $n/2$ quantum queries and quantum time, without any assumption on f, while classically $2^{n/4}$ queries would be required. Interestingly, the function G can be split into a secret part (g_0), and a publicly computable part (g_1). Hence, we can apply the offline Simon's algorithm to retrieve k in $2^{n/6}$ *classical* queries and quantum time.

Case f(x+k), 2-rounds self-similarity. If the Feistel has two alternating keys (k_0, k_1), we can use

$$G : \{0,1\} \times \left(\{0,1\}^{n/2}\right)^2 \to \{0,1\}^n$$
$$b, x, y \quad \mapsto \begin{cases} g_0(x) = E_{k_0,k_1}(x,y) - (x,y) & \text{if } b = 0, \\ g_1(x) = E_{0,0}(x,y) - (x,y) & \text{if } b = 1. \end{cases}$$

As we have the same property $E_{k_0,k_1}(x,y) + (k_0,k_1) = E_0(x + k_0, y + k_1)$, the hidden shift is (k_0, k_1). As before, this attack can be performed with only *classical* queries. It requires $2^{n/3}$ classical queries and quantum time, while classically $2^{n/2}$ queries are needed.

Table 9.1: Summary of the slide attack costs, n is the block size.

Branch and key addition	Round function	query cost	Remark
$\oplus$, any	Weak	$n/2$	
$\oplus, \oplus$	$f(x \oplus k)$	$n/2$	
$\oplus, \oplus$	$f_k(x)$	n^2	
$\oplus, +$	$f_k(x)$	$n2^{\sqrt{n}}$	
$+$, any	Weak	$2^{\sqrt{n}}$	
$+, +$	$f(x + k)$	$2^{\sqrt{n}}$	Need the exact same addition
$+, \oplus$	$f_k(x)$	$n2^{\sqrt{n}}$	
$+, +$	$f_k(x)$	$2^{2\sqrt{n}}$	
$\oplus, \oplus$	$f(x \oplus k)$	n	2-Round self-similarity
$+, +$	$f(x + k)$	$2^{\sqrt{2n}}$	2-Round self-similarity Need the exact same addition
$+, +$	$f(x + k)$	$2^{n/6}$	Classical queries, Need the exact same addition
$+, +$	$f(x + k)$	$2^{n/3}$	Classical queries 2-Round self-similarity Need the exact same addition

Table 9.1 presents a summary of the costs of this attack, depending on the group operation for the branch addition, and the key addition. It can be different for the branch and the key, except when stated otherwise. As the variants with classical queries have an exponential cost in time and queries, there is little difference between them and they are not separated in the table.

9.4 Slide attacks against 4-round self-similar Feistels

In this section, we present some attacks on 4-round self-similar Feistel schemes and their whitened variants. Contrary to the previous sections, these attacks require the key additions to be XORs.

9.4.1 Twist and complementation slide attack

Combining twist and complementation slides enables the authors in [BW00] to attack a 4-round self-similar Feistel. In this section we show how to efficiently quantize this attack and extend it to the whitened variant. The main idea is that the sequence of keys for encryption is $k_0k_1k_2k_3\dots$ and for decryption, $k_3k_2k_1k_0\dots$ (see Figure 9.9). If we slide by one round, we make the keys k_0 and k_2 coincide, whereas the keys in the other rounds always have a constant difference $\Delta = k_1 \oplus k_3$, similarly to the *complementation slide* technique. Let E_k and D_k be the encryption and decryption oracles.

We gather from [BW00] that a slide pair $(P, C) = (L, R), (M, N)$ (in input to E_k), $(P', C') = (L', R'), (M', N')$ (in input to D_k) satisfies the following properties:

$$L', R' = M \oplus \Delta \oplus f(k_0 \oplus N), N$$
$$M', N' = L \oplus f(R \oplus k_0) \oplus \Delta, R$$

For all plaintexts $P = (x, R)$ we write $D_k(M', N') = (L', R')$ and $M = Trunc_L(E_k(x, R))$, $N = Trunc_R(E_k(x, R))$, which gives:

$$D_k(x \oplus f(R \oplus k_0) \oplus \Delta, R) = M \oplus \Delta \oplus f(k_0 \oplus N), N$$
$$\implies Trunc_R(D_k(x \oplus f(R \oplus k_0) \oplus \Delta, R)) = Trunc_R(E_k(x, R))$$

Hence, for a fixed R, we have the following function G:

$$G : \{0,1\} \times \{0,1\}^{n/2} \to \{0,1\}^{n/2}$$
$$b, x \quad \mapsto \begin{cases} g_0(x) = Trunc_R(D_k(x, R)) & \text{if } b = 0, \\ g_1(x) = Trunc_R(E_k(x, R)) & \text{if } b = 1. \end{cases}$$

The hidden shift between g_0 and g_1 is $f(R \oplus k_0) \oplus \Delta$. As in Section 9.3, we remark that the function $R \mapsto f(R \oplus k_0) \oplus \Delta \oplus f(R)$ has k_0 as hidden period, which can be recovered with Simon's algorithm. Contrary to [DDW18], we do not need to assume that f is weak.

The total cost is around n^2 queries, as we need to combine two Simon's algorithms to recover k_0. Once k_0 is known, we can directly use the first attack to get $\Delta = k_1 \oplus k_3$, at a negligible cost of $n/2$.

Moreover, we can "shift" the cipher, that is, suppress the first round of encryption, and add it at the last round, in order to simulate access to the same cipher, but with the encryption keys (k_1, k_2, k_3, k_0) instead of (k_0, k_1, k_2, k_3). Hence, we can repeat the same attack, to get k_1, and $\Delta' = k_2 \oplus k_0$. With k_1 and Δ, we obtain k_3, and with k_0 and Δ', we obtain k_2. The total cost is around n^2, for a total key size of $2n$ bits.

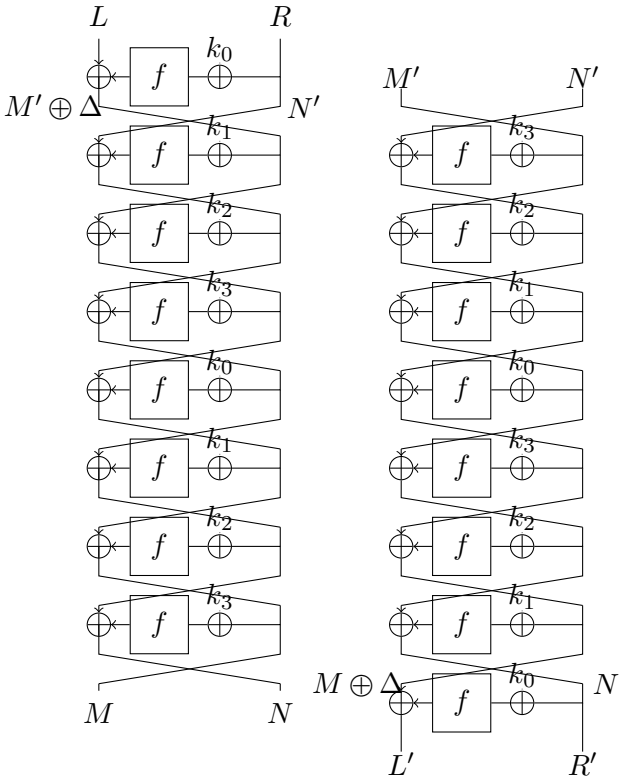


Figure 9.9: Complementation and twist combined on a 4k-Feistel scheme

Attacking 4k-WFeistel. Building on the previous 4k-Feistel attack, we can further extend it to 4k-WFeistel, with key whitenings k_{pre} and k_{post}. Indeed, if we rewrite the slide shift equation:

$$Trunc_R(D_k(x \oplus f(R \oplus k_0) \oplus \Delta, R)) = Trunc_R(E_k(x, R))$$

using the whole primitive $E(\cdot) = k_{post} \oplus E_k(k_{pre} \oplus \cdot)$, we obtain:

$$Trunc_R(D(x \oplus k^L_{post} \oplus f(R \oplus k_0) \oplus \Delta, R \oplus k^R_{post})) \oplus k^R_{pre} = \\ Trunc_R(E(x \oplus k^L_{pre}, R \oplus k^R_{pre})) \oplus k^R_{post}$$

which we rewrite:

$$Trunc_R(D(x \oplus \delta_L \oplus f(R \oplus k_0 \oplus k^R_{post}), R)) = Trunc_R(E(x, R \oplus \delta_R)) \oplus \delta_R$$

where $\delta_R = k^R_{pre} \oplus k^R_{post}$ and $\delta_L = k^L_{pre} \oplus k^L_{post} \oplus k_1 \oplus k_3$.

Now, we can use the function

$$G : \{0,1\} \times \left(\{0,1\}^{n/2}\right)^2 \to \{0,1\}^{n/2}$$

$$b, x, y \quad \mapsto \begin{cases} g_0(x,y) = Trunc_R(D(x \oplus f(y \oplus (k^R_{post} \oplus k_0)), y) \oplus y & \text{if } b = 0, \\ g_1(x,y) = Trunc_R(E(x,y)) & \text{if } b = 1. \end{cases}$$

We have $g_0(x, y) = g_1(x \oplus \delta_L, y \oplus \delta_R)$. However, to implement g_0, we need to know the value $(k^R_{post} \oplus k_0)$. Hence, we first guess the value of $(k^R_{post} \oplus k_0)$ using Grover's algorithm, and combine it with Simon's algorithm. In time and queries $2n2^{n/4}$, this attack recovers δ_L, δ_R and $(k^R_{post} \oplus k_0)$. Now, we can perform the analogue of the classical attack: swapping the encryption and the decryption, we obtain $k^L_{pre} \oplus k^L_{post}$ and $k^R_{pre} \oplus k^R_{post} \oplus k_0 \oplus k_2$, which allows to obtain $k_0 \oplus k_2$, $k_1 \oplus k_3$ and $k_{pre} \oplus k_{post}$. This step costs $n2^{n/4}$.

But we can also move differently δ_R in the slide shift property:

$$Trunc_R(D(x \oplus \delta_L \oplus f(R \oplus k_0 \oplus k^R_{pre}), R \oplus \delta_R)) = Trunc_R(E(x, R)) \oplus \delta_R \ .$$

We now interpret this as a hidden shift equation on x only and, given an arbitrary R, we can recover $f(R \oplus k_0 \oplus k^R_{pre})$. In turn, this gives us the value $k_0 \oplus k^R_{pre}$ for a total of n^2 queries.

Since we have already guessed $(k^R_{post} \oplus k_0)$, we deduce $k_0, k^R_{pre}, k^R_{post}$, and k_2 from $k_0 \oplus k_2$. Only $n/2$ bits of key material remain unknown in the Feistel scheme, as $k_1 \oplus k_3$ has been previously obtained. We can finish the attack by seeing the cipher as an instance of the FX construction, at a cost of $n2^{n/4}$ queries.

As this attack uses the *sliding with a twist* technique, it does not transpose well to modular additions.

A variant of 4k-WFeistel. We consider a variant of the 4k-WFeistel studied in [DKS15, Section 7.4]. It has two whitening keys k_{pre} and k_{post} and four alternating keys k_0, k_1, k_2, k_3 scheduled in $4m + 1$ rounds as: $(k_0, k_1, k_2, k_3)^m, k_0$.

Slide pairs $(P, C), (P', C')$ have the properties:

$$\begin{aligned} P \oplus C' &= k_{pre} \oplus k_{post} \oplus (k_1 \oplus k_3||0) \\ E_k(P \oplus k_{pre}) &= E_k^{-1}(C' \oplus k_{post}) \oplus (k_1 \oplus k_3||0) \end{aligned}$$

Where the term $\Delta = k_1 \oplus k_3$ intervenes as in the *complementation slide* to correct the inversion of E_k. We can rewrite this as a slide shift equation holding for all input x:

$$\begin{aligned} E_k(x) &= E_k^{-1}(x \oplus (\Delta||0)) \oplus (\Delta||0) \\ \implies E(x \oplus k_{pre}) \oplus k_{post} &= E^{-1}(x \oplus k_{post} \oplus (\Delta||0)) \oplus k_{pre} \oplus (\Delta||0) \\ \implies E(x) \oplus x &= E^{-1}(x \oplus \Delta') \oplus x \oplus \Delta' \end{aligned}$$

where $\Delta' = k_{pre} \oplus k_{post} \oplus (\Delta||0)$, a slide shift equation holding for all x. We can retrieve Δ' in only n queries, achieving a very efficient distinguisher. Obtaining the rest of the key seems more difficult.

9.4.2 Enhanced reflection attack

The *enhanced reflection attack* was introduced in [Din+15] for ciphers of the form $E = E_2 \circ E_1 \circ E_0$ where E_1 is an involution. It requires to find P such that $E_0(P)$ is a *fixpoint* of E_1. This happens with probability $2^{-n/2}$ for 1 round of Feistel ciphers. In this case we get directly $C = E_2(E_0(P))$.

Enhanced reflection attack on 4k-Feistel. In [Din+15], the authors use a reflection attack on 4k-Feistel, where the four alternating subkeys are denoted k_0, k_1, k_2, k_3. The core idea of the enhanced reflection attack ([Din+15, Property 3]) is that if at round $2m$ (which will use either k_0 or k_2 as a round key) the round function adds $\Delta = k_1 \oplus k_3$, then the remaining rounds will behave as if k_1 and k_3 were swapped. This means that all the following rounds will effectively *decrypt*, the only difference being an offset of Δ on one branch. All the rounds would be decrypted at round $4m + 1$, and the value would correspond to the original plaintext swapped with Δ added on the right branch. Hence, for the ciphertexts, we have $P^R = C^R$ and $P^L = C^L \oplus \Delta \oplus f(C^R \oplus k_0)$.

Reflection points, that satisfy these properties can be detected as $P^L = C^L$. Only $\mathcal{O}\left(2^{n/2}\right)$ known plaintexts are required. Given at least three of them, the adversary guesses Δ, then tries to obtain k_3 from the equation: if Δ is good, this works and both k_1 and k_3 can be obtained.

To complete the attack, the authors note that a similar reflection property holds with the equation $P^L = C^L$ to detect reflection points, and $P^R = C^R \oplus f(C^L \oplus k_3) \oplus (k_0 \oplus k_2)$ (this corresponds to the same property, but with the decryption, which swaps the two branches and inverts the key schedule).

In a quantum setting, only $\mathcal{O}\left(2^{n/4}\right)$ superposition queries are enough to retrieve the reflection points, using Grover search over all plaintexts P. Again, trying all values of Δ requires $\mathcal{O}\left(2^{n/4}\right)$ work. All subkeys are recovered in time $\mathcal{O}\left(2^{n/4}\right)$.

Enhanced reflection attack on 4k-WFeistel. The reflection attack on 4k-Feistel can be classically turned into an attack for 4k-WFeistel. We note k_{pre} and k_{post} the two whitening keys. The adversary first has to guess $k^L_{pre} \oplus k^L_{post}$. To do this, remark that reflection points for E' of the form P', C' turn into reflection points for E that satisfy $P^L \oplus C^L = k^L_{pre} \oplus k^L_{post}$.

In this, the correct value of $k^L_{pre} \oplus k^L_{post}$ appears with probability $2 \cdot 2^{-n/2}$, whereas all incorrect values have a probability $2^{-n/2}$ of appearance. This allows to retrieve $k^L_{pre} \oplus k^L_{post}$ using $\mathcal{O}\left(n2^{n/2}\right)$ memory and time, the bottleneck of the attack.

We leave a quantization of this attack as an open problem. Indeed, we do not know of a quantum algorithm that would solve this problem (among $n2^{n/2}$ arbitrary values, finding the one that appears twice more often than the others) with more than a constant speedup.

9.5 Cycle-based slide attacks

A generic framework of cycle-based slide attacks is presented in [Bar+18, Section 4.1]. The authors suggest that it could be accelerated in a similar way as the slide attacks from [Kap+16], expecting for instance exponential speedups. In this section we study these attacks. As they do not seem to have an exploitable slide-shift structure, we find smaller improvements than expected.

9.5.1 Definition of a cycle slide attack

We suppose that $E_k = f_k^\ell$ for some function f_k, which happens to be immune to simpler slide attacks such as those presented above for 1k-, 2k- and 4k-Feistel schemes. Consider a message P and the cycle built from P by iterating E_k: $P, E_k(P), E_k^2(P), \dots$ Let m_2 be the period of this cycle. Let also m_1 be the period of the f_k-cycle, that is, the smallest integer such that $f_k^{m_1}(P) = P$. Then one has $m_2 = m_1/gcd(m_1, \ell)$. Moreover, suppose that $gcd(m_1, \ell) = 1$, then $m_1 = m_2 = m$. This condition cannot be checked directly by the attacker, since he does not have access to f_k.

By Bezout's theorem, there exists d_1, d_2 such that $d_1 m - d_2 \ell = 1$. This gives:

$$\begin{aligned} f_k^{d_1 m - d_2 \ell + 1}(P) &= P \\ f_k^{d_1 m + 1} = f_k^{d_2 \ell}(P) &= E_k^{d_2}(P) \\ f_k(P) &= E_k^{d_2}(P) \end{aligned}$$

Hence $(P, E_k^{d_2}(P))$ is a slide pair. Moreover, $(E_k^t(P), E_k^{d_2+t}(P))$ is one for every t. This gives a certain number of slide pairs "for free", up to the length of the cycle. Once they have been obtained, we can use them to perform an attack on f_k and try to recover the key material.

General cycle size. We assume that E_k is a random permutation. In that case, the i-th largest cycle has size $e^{-i+1}\,(1 - 1/e)\,2^n$ (on average), the largest having size $(1 - 1/e)\,2^n$. In particular, on average, half of the points lie on the largest cycle. Finding a cycle of E_k then requires $c2^n$ chosen plaintext queries for some $c < 1$, which is a little less than the entire codebook.

The main interest is the time complexity: suppose that the attack on f_k needs time $\mathcal{O}(t)$, then the total time complexity is $\mathcal{O}(t + 2^n)$ and not $\mathcal{O}(t2^n)$ as would require a standard slide attack (see [Bar+18, Section 4.1]).

Combining cycles. It is important to note that the probability of success (i.e, m_1 is prime with ℓ) is strictly smaller than one, exactly $\phi(\ell)/\ell$ where ϕ is Euler's totient function. The only way to check that the slide pairs obtained were good is to try to attack f_k. Hence, it may be difficult to combine (when needed) the data obtained from multiple cycles.

In particular, if one is not able to tell if a given cycle is a good one (i.e, m_1 is prime with ℓ), the complexity can increase dramatically, since we would require all cycles to

be good at the same time: it happens only with probability $(\phi(\ell)/\ell)^t$ if there are t of them.

9.5.2 Quantization of a cycle-based slide attack

At the end of [Bar+18, Section 4.1], the authors suggest that a quantum period-finding algorithm could be applied to cycle-based slide attacks. The issue that we found when trying to do this is that, given a point P, the period of interest is the one of the function $G : d \in \mathbb{Z} \mapsto E_k^d(P)$. If G is implemented using a quantum circuit, we can indeed use Shor's period-finding algorithm to retrieve the cycle length. However, computing G is costly: to compute E_k^d, there is no more efficient way than performing d *successive calls* to E_k. To identify a cycle, we need to compute a d which is at least the size of the cycle: there is no quantum speedup.

Quantization. What is quantumly easier is to find a fixpoint of E_k using Grover search; or a point lying on a small cycle. In a random permutation, there are on average $1/d$ cycles of length d; and d points of period less than or equal to d (among 2^n). Finding if a point lies on such a cycle can be done with d queries to E_k. So using Grover's algorithm, one can find a superposition of points that lie on a cycle of length less than d in $\mathcal{O}\left(d\sqrt{2^n/d}\right) = \mathcal{O}\left(\sqrt{d2^n}\right)$ queries to E_k (and time).

In the classical case, we do not have much choice: the cycle slide pairs found will fall on a large cycle, with high probability. On the contrary, in the quantum setting, we can *specifically* look for points lying on a short cycle. Surprisingly, finding fixed points (or points on very short cycles) costs less than finding points on bigger cycles, due to the cost of iterating E_k. But the smaller the cycle, the less slide pairs we can get from it. Consequently, cycle-based slide attacks seem to be eligible to an interesting quantum speedup when the cipher $E_k = f_k^r$ does not enjoy a slide-shift property as before, but has a sufficiently weak round function f_k, so that a small number of slide pairs suffice to get the subkey material.

9.5.3 Examples

We are inspired by [Bar+18] and the attacks against the SA construction. In the classical as in the quantum versions, most of the computation time required is due to finding the actual slide pairs (via the cycle).

Two keys and two permutations. Consider a cipher with alternating keys k_0, k_1, xored or modularly added, and two permutations Π_1, Π_2 (Figure 9.10). In the case of a SPN, $\Pi_1 = \Pi_2 = \Pi$ are the same.

This scheme resists to the basic slide attack, but we can write $E_k \circ \Pi_2 = f_k^r(x)$ where $f_k(x) = \Pi_2(k_1 \oplus \Pi_1(k_0 \oplus x))$, and apply the cycle-finding technique. In $\mathcal{O}\left(2^{n/2}\right)$ superposition queries to E_k and computations, we can recover a small number of slide pairs, say two, from small cycles of $E_k \circ \Pi_2$. Recall that n is the block size here; the

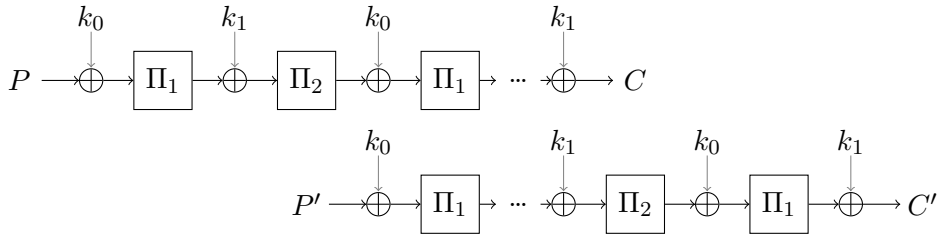


Figure 9.10: Slide attack against a key- and permutation-alternating cipher

key length is $2n$. Therefore we obtain two equations:

$$y = \Pi_2(k_1 \oplus \Pi_1(k_0 \oplus x))$$
$$y' = \Pi_2(k_1 \oplus \Pi_1(k_0 \oplus x'))$$

Since the permutations can be inverted, we find:

$$\Pi_2^{-1}(y) \oplus \Pi_2^{-1}(y') = \Pi_1(k_0 \oplus x) \oplus \Pi_1(k_0 \oplus x')$$

Solving this equation on k_0, if Π_1 has no specific property, can be done in $\mathcal{O}\left(2^{n/2}\right)$ time using Grover's algorithm, the same complexity as the first stage. This improves on the attack of Section 9.2, as the test is simpler than applying Simon's algorithm, but it cannot benefit from the query reduction method of Chapter 6.

Attacking 3k-SPN. Cycle-finding can further be applied on a 3k-SPN construction, where there is a unique permutation $\Pi = A \circ S$, with A a linear layer and S a non-linear layer of S-Boxes. Still using $\mathcal{O}\left(2^{n/2}\right)$ queries, we now write the slide equations as:

$$y = \Pi(k_2 \oplus \Pi(k_1 \oplus \Pi(k_0 \oplus x)))$$
$$y' = \Pi(k_2 \oplus \Pi(k_1 \oplus \Pi(k_0 \oplus x')))$$
$$\implies \Pi^{-1}(y) \oplus \Pi^{-1}(y') = \Pi(k_1 \oplus \Pi(k_0 \oplus x)) \oplus \Pi(k_1 \oplus \Pi(k_0 \oplus x'))$$

To solve efficiently this equation in k_0 and k_1, we first guess k_0 using Grover's algorithm. The equation on k_1 becomes:

$$A^{-1}(\Pi^{-1}(y) \oplus \Pi^{-1}(y')) = S(k_1 \oplus \Pi(k_0 \oplus x)) \oplus S(k_1 \oplus \Pi(k_0 \oplus x'))$$

Furthermore, we may consider each S-Box separately and solve the equation on k_1, S-Box by S-Box. If s is the bit size of an S-Box, the final complexity of this attack is $\mathcal{O}\left(2^{(n+s)/2}\right)$ computations, with $\mathcal{O}\left(2^{n/2}\right)$ oracle queries.

Against 3k-Feistel. A Feistel scheme with a mixing function f, alternating three keys k_0, k_1, k_2, xored or modularly added, is immune to the *complementation slide* and *sliding with a twist* techniques. It seems difficult to write a slide shift property for this cipher. Let us write the round function g as:

$$L, R \mapsto R + f(k_1 + L + f(k_0 + R)),$$
$$L + f(k_0 + R) + f(k_2 + f(k_1 + L + f(k_0 + R)))$$

and suppose that we can invert f. In $\mathcal{O}\left(2^{n/2}\right)$ queries, we can find two slide equations $g(L,R) = L', R'$, which imply $f(k_1 + L + f(k_0 + R)) = L' - R$. Regardless of the function f, we can invert it in time $\mathcal{O}\left(2^{n/4}\right)$ using Grover and recover two equations $k_1 + L + f(k_0 + R) = X$. We take the difference (or sum if we replace $+$ by $\oplus$) to eliminate k_1, and we can solve the remaining equation on k_0 using Grover in $\mathcal{O}\left(2^{n/4}\right)$ time. Once this is done, k_1 can be found via the relation $k_1 = f^{-1}(L'-R)-L-f(k_0+R)$ and k_2 via $L + f(k_0 + R) + f(k_2 + f(k_1 + L + f(k_0 + R))) = R'$.

The whole attack requires $\mathcal{O}\left(2^{n/2}\right)$ time and queries due to the cycle finding, with any function f.

Against 4k-Feistel. If we append one more round key k_3, the round function g becomes:

$$L, R \mapsto L + f(k_0 + R) + f(k_2 + f(k_1 + L + f(k_0 + R))),$$
$$R + f(k_1 + L + f(k_0 + R)) + f(k_3 + L + f(k_0 + R) + f(k_2 + f(k_1 + L + f(k_0 + R))))$$

Again, we can find some slide equations $g(L,R) = L', R'$ from a cycle in $\mathcal{O}\left(2^{n/2}\right)$ queries. We guess the subkey k_0. For each guess, we can rewrite the equations as if there were only 3 subkeys, and solve them in time $\mathcal{O}\left(2^{n/4}\right)$ using multiple Grover instances, as seen above, regardless of the properties of f. The whole attack requires $\mathcal{O}\left(2^{n/2}\right)$ time and queries, the two steps (cycle finding and solving equations) are now balanced. The time complexity is greater than the other 4k-Feistel attacks seen above, but there is no restriction on the function f and the operations used; furthermore, we only use encryption queries, not decryption queries (which is the case of the twist).

9.6 Attacks on Feistels with weak key schedules

The attack of Section 9.3.2 is slightly different from the other slide attacks presented in this chapter, as the similarity is not between one round and the next of the same function, but between one round and the same round, with a different key. It only relies on the complementation property of key-alternating Feistels, which is described in Figure 9.11. This round property can be extended to the whole cipher, that is, we have $E_k(x,y) + (k,k) = E_0(x + k, y + k)$. This holds for any type of addition, as long as the key and branch addition are the same. Moreover, this property does not require to have an identical round function, but only to have an identical key for each round. In particular, if the round function is of the form $f(x + k + c_i)$, that is, only round constants are used for the key schedule, we have this property. As in Section 9.3.2, this attack can also be generalized when two keys k_0, k_1 are used alternatively.

9.6.1 Classical attacks on MiMC and GMiMC

MiMC [Alb+16] and GMiMC [Alb+19] are families of block ciphers and hash functions designed for multipartite computation. They are defined in a finite field, which may be $\mathbb{F}_{2^n}$ for a large n (in which case the addition is a xoring) or $\mathbb{F}_p$ for a large prime number p (in which case the addition is done modulo p).

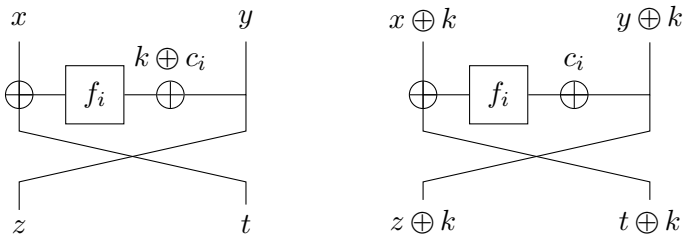


Figure 9.11: Complementation property

MiMC. Two block ciphers are proposed in [Alb+16], an SPN named MiMC-n/n (n-bit block size, n-bit key size) and a Feistel named MiMC-$2n/n$ ($2n$-bit block size, n-bit key and branch size). Their round function is of the form $(x + k + c_i)^3$.

GMiMC. GMiMC [Alb+19] is a family of generalizations of MiMC-$2n/n$ that use a generalized Feistel structure. Two key schedules are proposed: the univariate key schedule, in which only round constants are used from a fixed key, and the multivariate key schedule, which uses a matrix multiplication to produce t new round keys from the t previous ones. It turns out that with the univariate key schedule, we have in most cases the same property, but with more branches, that is, $E_k(x_1, \ldots, x_t) + (k, \ldots, k) = E_0(x_1 + k, \ldots, x_t + k)$. The different cases proposed are detailed below.

GMiMC-crf. GMiMC-crf has t branches and adds a function of $t - 1$ branches on one branch. The round function is

$$R_k^i(x_1, \ldots, x_t) = x_2, \ldots, x_t, x_1 + \left(\sum_{j=2}^{t} x_j + k + c_i\right)^3 \quad .$$

In order to have the round property, we must have

$$\left(\sum_{j=2}^{t} x_j\right) + k = \sum_{j=2}^{t} (x_j + k)$$

Hence, it only works if $(t-2)k = 0$, which either imposes $t = 2$ (which corresponds to MiMC-$2n/n$) or $t - 2$ is a multiple of the field caracteristic, which occurs if we have a xoring and an even number of branches.

GMiMC-erf. GMiMC-erf has t branches, and adds a function of one branch on all the other. The round function is

$$R_k^i(x_1, \ldots, x_t) = x_2 + (x_1 + k + c_i)^3, \ldots, x_t + (x_1 + k + c_i)^3, x_1 \quad .$$

Hence, we have that $R_k^i(x_1, \ldots, x_t) = R_0^i(x_1 + k, \ldots, x_t + k)$, and the same property on the full cipher.

GMiMC-Nyb. GMiMC-Nyb has $2t$ branches, and adds a function of each odd branch to the next branch. The round function is

$$R_k^i(x_1, \ldots, x_{2t}) = x_2 + (x_1 + k + c_{ti})^3, \\ x_3, x_4 + (x_3 + k + c_{ti+1})^3, \ldots, x_{2t} + (x_{2t-1} + k + c_{ti+t-1})^3, x_1.$$

Hence, we have that $R_k^i(x_1, \ldots, x_{2t}) = R_0^i(x_1 + k, \ldots, x_{2t} + k)$, and the same property on the full cipher.

GMiMC-mrf. GMiMC-mrf is a generalization of the previous construction with a permutation of the branches that change for each round, which maintains the property.

Attack. The attack can be performed as before: if we define f and g as

$$f(x) = E_k(x, \ldots, x) - (x, \ldots, x) \quad , \quad g(x) = E_0(x, \ldots, x) - (x, \ldots, x)$$

then we have $f(x) = g(x + k)$. Moreover, g only depends on public parameters, which allows to use the offline approach of Chapter 6. Hence, for MiMC-$2n/n$ and all univariate versions of GMiMC except some versions of GMiMC-crf:

- with quantum queries, key recovery can be done in polynomial (in $\mathbb{F}_{2^n}$) or subexponential (in $\mathbb{F}_p$) time,
- with a quantum computer and classical queries, key recovery can be done in around $2^{n/3}$ classical queries and quantum time,
- with classical resources only, a collision recovers the key and requires around $2^{n/2}$ queries.

If the designers did not make any quantum security claim, the classical security claim was of 2^n operations, which makes this attack a break of all the affected ciphers. The attack is independent of the round function and the number of rounds, but relies on the weak key schedule of the construction. Hence, using a stronger key schedule would suffice to thwart the attack.

9.7 Conclusion

This chapter presented the quantization of many classical slide attacks, which are summarized in Table 9.2. The attacks that could be described with a slide-shift property have benefited of pretty significant speedup, often exponential or subexponential. The slide on Feistel cipher is also the first example of a quantum cryptanalysis that constructs a periodic function from another quantum period finding algorithm, which demonstrates that the relevant notion of security for a round function can be its quantum security.

Classical slide attacks have shown the importance of a good key schedule, as self-similarity in a cipher allows for powerful breaks. In the quantum setting, these results seem to put even more weight on this design principle, as the attacks become much

more efficient. This incentive to express the slide property as a functional equality has also helped us to find a *classical* attack against MiMC-2n/n and univariate GMiMC.

As the cost of quantum slide-shift attacks is very small, a possible future direction for improvement would be to consider new attack patterns, intrinsically unfeasible in a classical setting, with stronger functions relating the slide pairs. Another direction would be to find new applications that do not require quantum queries. Indeed, if the quantum slide attacks generally require quantum queries, this is not the case of some attacks in Section 9.3.2 and Section 9.6, which are the first example of a quantum hidden period attack with classical queries on an iterated construction.

Table 9.2: Quantum slide attacks. n is the block size. The attacks are a complete key-recovery, except for the three distinguishing attacks.

Cipher	Attack details	Queries	Dec. oracle	Source
1k-Feistel (XOR)	Basic slide	$n/2$		[DDW18]
1k-Feistel (additions)	Basic slide	$2^{\sqrt{n}}$		Section 9.2.2
1k-Feistel (any)	Composed slide	$n/2$ to $2^{2\sqrt{n}}$		Section 9.3
1k-Feistel (any)	Composed slide, classical queries	$2^{n/6}$		Section 9.3
2k-Feistel (XOR)	Complementation	$n2^{n/4}$		Section 9.2.3
2k-Feistel (XOR)	Slide with a twist	$n/2$	Yes	[DDW18]
2k-Feistel (XOR)	Composed slide	n		Section 9.3
2k-Feistel (additions)	Complementation	$2^{\sqrt{n}+n/4}$		Section 9.2.3
2k-Feistel (additions)	Composed slide	$2^{\sqrt{2n}}$		Section 9.3
2k-Whitened Feistel	Complementation	$n2^{n/4}$		Section 9.2.3
2k-Feistel (any)	Composed slide, classical queries	$2^{n/3}$		Section 9.3
3k-Feistel (any)	Cycle finding	$2^{n/2}$		Section 9.5
4k-Feistel (XOR)	Complementation, slide with a twist	n (Dist.)	Yes	[DDW18]
4k-Feistel (XOR)	Complementation, slide with a twist	$4n^2$	Yes	Section 9.4.1
4k-Feistel (XOR)	Enhanced reflection	$2^{n/4}$		Section 9.4.2
4k-Feistel (any)	Cycle finding	$2^{n/2}$		Section 9.5
4k-Whitened Feistel (XOR)	Complementation, slide with a twist	$n^2 2^{n/4}$ $n2^{n/4}$ (Dist.)	Yes	Section 9.4.1
4k-Whitened Feistel (variant) (XOR)	Mirror slidex	n (Dist.)	Yes	Section 9.4.1
1k-SPN (XOR)	Basic slide	n		[Kap+16]
1k-SPN (additions)	Basic slide	$2^{\sqrt{2n}}$		Section 9.2
2k-SPN (any)	Grover-meet-Simon	$n2^{n/2}$		Section 9.2
2k-SPN (any)	Cycle finding	$2^{n/2}$		Section 9.5
*k-SPN (XOR)	Related-key	n		[HA17]

Chapter 10
Computing Isogenies

The use of isogenies in cryptography is fairly recent. A first key exchange mechanism was proposed by Couveignes [Cou06] and independently discovered by Rostostev and Stolbunov [RS06]. These early designs were mostly of theoretical interest, due to their poor performance. A first quantum attack against them in subexponential time was proposed by Childs, Jao and Soukharev [CJS14]. Two improved variants on these designs have been proposed recently, by De Feo, Kieffer and Smith [DKS18], and CSIDH, by Castryck, Lange, Martindale, Panny and Renes [Cas+18]. This last scheme has proven itself quite popular, with a signature based on the same structure [DG19] and multiple improvements proposed since [MR18; Jal+19; MCR19; DPV19]. Another approach for key exchange, which does not suffer from any subexponential quantum attack has been proposed by De Feo, Jao and Plût [JD11]: SIDH. The only NIST candidate using isogenies, SIKE [SIKE], is based on SIDH. We studied the asymptotic security of CSIDH with Jean-François Biasse, Benjamin Pring, André Schrottenloher and William Youmans in [Bia+19], and its concrete security, with André Schrottenloher, in [BS18].

Contents

10.1 Key exchange from hard homogeneous spaces

Couveignes [Cou06] proposed the following notion:

Definition 10.1 (Hard homogeneous space)**.** Let G be a group, X a set, with a free and transitive action $\cdot$ of G on X, that is, for any $x_1, x_2 \in X$, there exists a unique $g \in G$ such that $x_2 = g \cdot x_1$. X is a *hard homogeneous space* for G if it is easy to compute the group action, but, given x_1, x_2, it is hard to find the unique g such that $x_2 = g \cdot x_1$.

If G is abelian, we can easily make a key exchange protocol from a hard homogeneous space, as decribed in Algorithm 10.1. In fact, if X is the set of elements of maximal order of a cyclic group of order N and G is $(\mathbb{Z}/(N))^*$, this is the Diffie-Hellman key exchange.

Algorithm 10.1 Key exchange from an abelian hard homogeneous space

1: **Initialization**: choose a public $x_0 \in X$
2: **Alice**: chooses a secret value $g_A \in G$, and sends $g_A \cdot x_0$.
3: **Bob**: chooses a secret value $g_B \in G$, and sends $g_B \cdot x_0$.
4: **Outcome**: the shared secret is $g_B \cdot g_A \cdot x_0 = g_A \cdot g_B \cdot x_0$.

Quantum security. In the case of a Diffie-Hellman key exchange, we can apply Shor's algorithm, which breaks the key exchange. In the general case, as G is abelian, we have a hidden shift property. Indeed, from x and $x' = g \cdot x$, we have, for all h, $h \cdot x' = (h \cdot g) \cdot x$. Hence, we can consider $f_0(h) = h \cdot x'$ and $f_1(h) = h \cdot x$, which satisfies $f_0(h) = f_1(h \cdot g)$. Then, from f_0 and f_1, we can recover g in subexponential time, using the algorithms of Chapter 5.

Instances. To date, the only known instances of hard homogeneous spaces, besides Diffie-Hellman, are ordinary isogeny-based schemes and CSIDH. If the principles of SIDH are quite similar, its underlying structure is different, as there is no group, which prevents this formalism and the quantum subexponential attacks to be applied.

10.2 Group action with isogenies

The aim of this section is to present how we can construct hard homogeneous spaces from ordinary curves (for [Cou06; RS06; DKS18]) and from supersingular curves defined over $\mathbb{F}_p$ (for [Cas+18]). A more detailed explanation of the mathematical background behind elliptic curve can be found in [Sil86]. At their core, these protocols use elliptic curves over a finite field of large characteristic.

Definition 10.2 (Elliptic curve)**.** An elliptic curve over a field $\mathbb{K}$ when $\mathrm{char}(\mathbb{K}) \neq 2, 3$ is the set of points solution of an equation of the form $y^2 = x^3 + ax + b$.

It is well known that if we add the point at infinity, O, to the set of points on an elliptic curve, we obtain an abelian group. We note, for a point P, $[m]P = \underbrace{P + \cdots + P}_{m \text{ times}}$.

Definition 10.3 (Points of m-torsion)**.** We note $E[m] = \{P \in E | [m]P = O\}$ the set of points of m-torsion.

Definition 10.4 (j-invariant)**.** The j-invariant of an elliptic curve E is the quantity

$$j(E) = 1728 \frac{4a^3}{4a^3 + 27b^2}$$

Definition 10.5 (Isogeny)**.** Let E_1, E_2 be two elliptic curves over $\mathbb{K}$. An isogeny $\phi : E_1 \to E_2$ is an algebraic morphism that satisfies $\phi(O_{E_1}) = O_{E_2}$. The *degree* of an isogeny is its degree as an algebraic map.

Definition 10.6 (Isogenous curves)**.** Two elliptic curves are isogenous if there is a non-zero isogeny between them.

Proposition 10.1 (Isogenous curves over $\mathbb{F}_q$ [Tat66])**.** *Two curves defined over a finite field are isogenous if and only if they have the same number of points.*

The previous proposition allows to define the graph of isogenous curves, which is the structure in which the key exchange will take place.

Proposition 10.2 (Isomorphic curves)**.** *Two isogenous curves are isomorphic if and only if they have the same j-invariant.*

As we are interested in isogenies up to isomorphism, this property allows to use the j-invariant to represent a curve in the isogeny graph.

Definition 10.7 (Endomorphism ring)**.** The endomorphism ring $\mathrm{End}(E)$ over a field $\mathbb{K}$ of an elliptic curve E is the set of isogenies from E to E defined over $\mathbb{K}$. It forms a ring under addition and composition.

Definition 10.8 (Order)**.** Let $\mathcal{A}$ be a finitely generated $\mathbb{Q}$-algebra. An order $\mathcal{O} \subset \mathcal{A}$ is a subring which is also a $\mathbb{Z}$-lattice of maximal dimension, that is, there exists a set $\{b_1, \ldots, b_d\}$ such that: $\mathcal{O} = b_1\mathbb{Z} + \cdots + b_d\mathbb{Z}$ and $\mathcal{A} = b_1\mathbb{Q} + \cdots + b_d\mathbb{Q}$.

Definition 10.9 (Quaternion algebra)**.** A quaternion algebra is an algebra of the form

$$\mathcal{A} = \mathbb{Q} + \alpha\mathbb{Q} + \beta\mathbb{Q} + \alpha\beta\mathbb{Q}$$

with the constraints

$$\alpha^2, \beta^2 \in \mathbb{Q}, \quad \alpha^2, \beta^2 < 0, \quad \alpha\beta = -\beta\alpha.$$

Proposition 10.3 (Endomorphism ring structure)**.** *When $\mathbb{K}$ is a finite field, the endomorphism ring of an elliptic curve over $\overline{\mathbb{K}}$ is either:*

- *An order in a quadratic imaginary field ($\mathbb{Q}[\sqrt{D}]$, with the discriminant $D < 0$), in which case $\mathrm{End}(E)$ has dimension 2 and E is said to be ordinary,*
- *An order in a quaternion algebra in which case $\mathrm{End}(E)$ has dimension 4 and E is said to be supersingular.*

Proposition 10.4 (Supersingular curves)**.** *Any supersingular curve is defined over an $\mathbb{F}_{p^2}$. If it is defined over $\mathbb{F}_p$, then its endomorphism ring over $\mathbb{F}_p$ is an order in a quadratic imaginary field.*

Proposition 10.5 (Number of points)**.** *An elliptic curve over $\mathbb{F}_q$ has a number of points N which satisfies $|N - (q+1)| \leqslant 2\sqrt{q}$. A supersingular curve has $q+1$ points.*

Definition 10.10 (Integral ideal)**.** An integral ideal (or ideal) I over an abelian ring R is a subgroup of R which is absorbing, that is, $RI = IR = I$. An ideal is principal if it is of the form iR, with $i \in R$.

Definition 10.11 (Fractional ideal)**.** A fractional ideal of an order $\mathcal{O}$ is a module of $\mathbb{K}$ of the form αI, with $\alpha \in K^*$ and I an integral ideal of $\mathcal{O}$. A fractional ideal I is invertible if there exists a fractional ideal J such that $IJ = JI = \mathcal{O}$.

Definition 10.12 (Class group)**.** The set of invertible fractional ideals of $\mathcal{O}$, $I(\mathcal{O})$ forms a group, which has the set of principal fractional ideals $P(\mathcal{O})$ as a normal subgroup. The class group of $\mathcal{O}$ is the quotient

$$\mathcal{C}\ell(\mathcal{O}) = I(\mathcal{O})/P(\mathcal{O}).$$

Remark 10.1 (Supersingular curves)*.* A quaternion algebra is not commutative, which restricts the ideals to either left or right ideals, and prevents to define a class group.

Proposition 10.6 (Class group size)**.** *On average,* $\mathcal{C}\ell(\mathcal{O}) \sim \sqrt{q}$.

Proposition 10.7 (Class group and isogenies)**.** *An element* $[\mathfrak{a}] \in \mathcal{C}\ell(\mathcal{O})$ *corresponds to the kernel of an isogeny, unique up to isomorphism, that we note* $\phi_\mathfrak{a} : E \to E/\mathfrak{a}$*. We have* $End(E) = End(E/\mathfrak{a})$*.*

Moreover, the group law of the class group corresponds to the composition of isogenies (even if their domains are different elliptic curves, their endomorphism ring is the same, which allows to define the same isogeny with a different domain by considering the corresponding class group element).

Proposition 10.8 (Class group action)**.** *Let* E_0 *be an elliptic curve over* $\mathbb{F}_q$ *with a class group* $\mathcal{C}\ell(End(E_0))$*. The class group acts freely and transitively over the set* $Ell(E_0, \mathbb{F}_q)$ *of curves isogenous to* E_0 *with the same class group, with the action*

$$\begin{array}{ccc} \mathcal{C}\ell(End(E_0)) \times Ell(E_0, \mathbb{F}_q) & \to & Ell(E_0, \mathbb{F}_q) \\ ([\mathfrak{a}], E) & \mapsto & E/\mathfrak{a} \end{array}$$

10.3 Isogeny evaluation

10.3.1 For a key exchange

In order to make a key exchange, we need an efficient method to compute an isogeny given an element of the class group, that is, a subgroup of an elliptic curve. Unfortunately, this is hard to do in general: an isogeny of degree m over $\mathbb{F}_q$ costs $O(m)$ operations in $\mathbb{F}_q$ to be evaluated. Hence, to be efficient, we cannot evaluate in one step an isogeny of large degree, and we need to evaluate a sequence of isogenies of small degree. The curve has to be chosen such that it has many isogenies of small degree. This means the curve has many small subgroups, which implies that its cardinality is smooth. In the key exchange, the set of small isogenies is a public parameter, and the secret key is a vector that states how many times each small isogeny is applied.

Ordinary curves. This approach has been used with ordinary curves by De Feo, Kieffer and Smith [DKS18]. Unfortunately, with the curve they found, they did not have enough isogenies of small degree defined over the base field, and had to take some isogenies over an extension, up to degree 9. Overall, this makes the protocol uncompetitive.

Supersingular curves. CSIDH [Cas+18] benefits from a much more favorable situation, with everything defined over $\mathbb{F}_p$. The designers chose a prime p of the form $4\ell_1 \dots \ell_u - 1$, with the ℓ_i some small distinct primes. Each ℓ_i will correspond to a small isogeny $[\mathfrak{l}_i]$. An additional parameter m is chosen, and the secret isogeny is of the form $[\mathfrak{l}_1]^{e_1} \dots [\mathfrak{l}_u]^{e_u}$, with $|e_i| \leqslant m$. They take m such that $(2m+1)^u \simeq \sqrt{p}$, and expect that as the set of secret isogenies is roughly the same size as the class group, secret isogenies should almost cover the whole class group.

The proposed parameters are presented in Table 10.1, with their expected security, which is aligned on the odd levels of security proposed by the NIST for its call for new key exchanges and signatures [NIST16]. The rationale behind the size of p is that $\sqrt{p}$ should be the size of the hard homogeneous space, which makes the best classical attack in $\sqrt{\sqrt{p}} = p^{1/4}$. For quantum attacks, in the first version of [Cas+18], they considered the query complexity, and relied on the estimate of $L(\sqrt{2})$ queries from [CJS14, Theorem 5.2]. As shown in Chapter 5, this was a loose estimate for an unoptimized version of the least time-efficient subexponential algorithm. The final version relied on a weaker notion of security, that considers the explicit cost of the attack (taking into account the polynomial factors and the evaluation cost, for the attacker, of an isogeny), and interprets the NIST levels as the cost to break AES, and not as a number of allowed queries.

Table 10.1: Approximate parameters for the three security levels of CSIDH.

Level	Expected quantum security	$\log_2 p$	u	m
NIST 1	As hard as AES-128 key-recovery	512	74	5
NIST 3	As hard as AES-192 key-recovery	1024	132*	7*
NIST 5	As hard as AES-256 key-recovery	1792	209*	10*

*In [Cas+18], a prime p is given only for the first instance. The value of u given for the other instances is an upper bound.

10.3.2 For a key recovery

For key recovery, we want to apply a hidden shift algorithm. This imposes to have a quantum circuit that, given any element of the class group $[\mathfrak{a}]$ and an elliptic curve E, computes $E/\mathfrak{a}$. As for the key exchange, the direct computation is extremely expensive, and we need an efficient approach.

Group structure. First, we need to know the structure of the class group. This is easy to do with a quantum computer, using the polynomial-time algorithm of [ME98]. Alternatively, we can use the subexponential classical algorithm from [HM89].

10.3.2.1 Efficient isogeny evaluation

Once this is known, the approach is simple: take a set of small elements of the class group, and try to write $[\mathfrak{a}]$ as a small combination of elements in the set, using methods from lattice reduction. This approach is standard for isogeny evaluation, and has been proposed in multiple works [Cou06; GHS02; BCL08]. We can describe it as follows:

Let $\mathfrak{p}_1, \dots, \mathfrak{p}_u$ be prime ideals generating $\mathcal{C}\ell(\mathcal{O})$. Let $\mathcal{L}$ be the lattice of relations between $\mathfrak{p}_1, \dots, \mathfrak{p}_u$, i.e. the lattice of all the vectors $(f_1, \dots, f_u) \in \mathbb{Z}^u$ such that $\prod_i \mathfrak{p}_i^{f_i}$ is principal. In other words, the ideal class $\left[\prod_i \mathfrak{p}_i^{f_i}\right]$ is the neutral element of $\mathcal{C}\ell(\mathcal{O})$. The strategy for computing the action of $[\mathfrak{a}] \in \mathcal{C}\ell(\mathcal{O})$ on a curve E is the following:

1. Compute a basis B for $\mathcal{L}$,
2. Reduce B to B' using the BKZ algorithm [SE94],
3. Find $(h_1, \dots, h_u) \in \mathbb{Z}^u$ such that $[\mathfrak{a}] = \left[\prod_i \mathfrak{p}_i^{h_i}\right]$,
4. Use Babai's nearest plane method on B' to find short $(h'_1, \dots, h'_u) \in \mathbb{Z}^u$ such that $[\mathfrak{a}] = \left[\prod_i \mathfrak{p}_i^{h'_i}\right]$,
5. Evaluate the action of $\left[\prod_i \mathfrak{p}_i^{h'_i}\right]$ on E by applying repeatedly the action of the $\mathfrak{p}_i$ for $i = 1, \dots, u$.

Step 1 is the previous precomputation. Step 2 can be performed as a precomputation requiring only classical gates. Step 3 is Shor's algorithm. Step 4 is cheap. The cost of Step 5 depends on the norm of the h'_i, which depends on the quality of B'.

Classical/quantum tradeoff. Step 2 is purely classical, and impacts the quantum Step 5. Hence, there is a natural classical/quantum tradeoff on this algorithm. This can be combined with the hidden shift algorithms which also have a classical/quantum tradeoff (see Theorem 5.5).

Asymptotic complexity. Asymptotically, we need to have enough small primes to perform the reduction. We rely on the following heuristic:

Heuristic 10.1 (Number of small primes)**.** *Let $0 < \alpha < 1/2$ and $\mathcal{O}$ be an imaginary quadratic order of discriminant D. There are $(\mathfrak{p}_i)_{i \leqslant k}$ for $k = \log^{1-\alpha}(|D|)$ prime ideals of norm in $\mathrm{Poly}(\log(|D|))$ whose classes generate $\mathcal{C}\ell(\mathcal{O})$. Furthermore, each class of $\mathcal{C}\ell(\mathcal{O})$ has a representative of the form $\prod_i \mathfrak{p}_i^{n_i}$ for $|n_i| \leqslant e^{\log^{\alpha}|D|}$.*

Assuming we have enough primes, we can estimate the cost of BKZ:

Lemma 10.1 ([BIJ18]). *Let $\mathcal{L}$ be an n-dimensional lattice with input basis $B \in \mathbb{Z}^{n\times n}$, and let $\beta < n$ be a block size. Then the BKZ variant of [HPS11] used with Kannan's enumeration technique [Kan83] returns a basis $\vec{b'}_1, \ldots, \vec{b'}_n$ such that $\|\vec{b'}_1\| \leqslant e^{\frac{n}{\beta}\ln(\beta)(1+o(1))}\lambda_1(\mathcal{L})$, using time $\mathrm{Poly}(n, \mathrm{Size}(B))\beta^{\beta\left(\frac{1}{2e}+o(1)\right)}$ and polynomial space.*

From this lemma, we can take $n = \log^{1-\alpha}(|D|)$ and $\beta = \log^{1-2\alpha}(|D|)$, to obtain the following corollary:

Corollary 10.1. *Assuming Heuristic 10.1 for α, the precomputation, runs in time $L(1-2\alpha, \frac{\alpha}{e(1-2\alpha)})$ and has polynomial space complexity. It returns a basis of $\mathcal{L}$ whose first vector $\vec{b'}_1$ satisfies $\|\vec{b'}_1\| \leqslant L(\alpha, \frac{1-2\alpha}{1-\alpha})$.*

We rely on another heuristic to estimate the quality of the outcome of Babai's algorithm:

Heuristic 10.2 (GSA). *The precomputed basis B' satisfies the Geometric Series Assumption (GSA): there exists $0 < q < 1$ such that $\|\vec{\tilde{b}'_i}\| \sim q^{i-1}\|\vec{b'_1}\|$ where $\left(\vec{\tilde{b}'_i}\right)_{i\leqslant n}$ is the Gram-Schmidt basis corresponding to B'.*

Proposition 10.9. *Assuming Heuristic 10.1 for $0 < \alpha < 1/2$ and Heuristic 10.2, the quantum evaluation of any isogeny runs in quantum time $L\left(\alpha, \frac{1-2\alpha}{1-\alpha}\right)$ with polynomial space.*

Proof. Each computation of a $[\mathfrak{p}_i]$ has a cost polynomial in $\log(p)$ and in the norm of $[\mathfrak{p}_i]$. Moreover, Babai's algorithm runs in polynomial time and returns a vector $\vec{b}$ such that

$$\|\vec{h} - \vec{v}\| \leqslant \frac{1}{2}\sqrt{\sum_i \|\vec{\tilde{b}'_i}\|^2} \leqslant \frac{1}{2}\sqrt{n}\|\vec{b'}_1\| \in L\left(\alpha, \frac{1-2\alpha}{1-\alpha}\right).$$

Therefore, each $h_i - v_i$ is in $L\left(\alpha, \frac{1-2\alpha}{1-\alpha}\right)$, which is the cost of evaluating the isogenies. □

This classical/quantum tradeoff is in fact more favorable than Theorem 5.5, hence the asymptotic cost is the one of Theorem 5.5, in $L(1-\alpha, 0.72\beta)$ classical time and $L(\alpha, (1-\alpha)/\beta)$ quantum time for any $0 < \alpha < 1/2$.

10.3.2.2 Case of CSIDH

CSIDH comes with a large set of small isogenies. This set reduces the cost of the key exchange, and can also be used to reduce the cost of isogeny evaluation in the attack. We are given the set $([\mathfrak{l}_i])_{i\leqslant u}$, and we assume that the class group is contained in $\prod_{i\leqslant u, |j|\leqslant m}[\mathfrak{l}_i]^j$. We analyzed the cost for the concrete parameters of CSIDH in [BS18].

To efficiently compute the isogeny, we still do a lattice reduction. As a large basis of small primes is provided by CSIDH, we rely on them.

Basis Quality. We want to have a concrete estimate, and not only some asymptotics for the quality of the basis. The quality of the basis is related to the Hermite factor. The first vector of the basis B in output, b_1, is such that

$$||b_1||_2 \leqslant c^u (\mathrm{Vol}(\mathcal{L}))^{1/u}$$

where c^u is the Hermite factor, and c a constant which depends on the algorithm used. For our purposes, it is better to work with the approximation factor, which relates $||b_1||_2$ and $\lambda_1(\mathcal{L})$, the euclidean norm of the smallest vector in $\mathcal{L}$. An approximation factor of c^{2u} is guaranteed, but in practice, it is equal to (and sometimes better than) the Hermite factor. So we consider:

$$||b_1||_2 \leqslant c^u \lambda_1(\mathcal{L}) \ .$$

BKZ-20 gives a heuristic constant c of approximately 1.0128 [GN08]. Furthermore, in [GN08, Fig. 12], the authors give a running time for BKZ-20 of the order 1000 CPU seconds for a dimension of 200.

We assume that the whole group is contained in the set of secret isogenies of CSIDH, which means that there exists at least one vector $\vec{e} = (e_1 \dots e_u)$ with $e_i \in \{-m, \dots, m\}$ such that $\prod_i [\mathfrak{l}_i]^{e_i} = \mathbf{1}$. This only assumption suffices to write that $\lambda_1(\mathcal{L}) \leqslant 2m\sqrt{u}$, hence $||b_1||_2 \leqslant 2c^u m\sqrt{u}$.

Effect on the L_1 Norm. We are interested on the L_1 norm of the difference $\vec{v} - \vec{h}$. Indeed, the L_1 norm counts the number of successive isogenies to be applied. We can count (roughly) the action of $\prod_i [\mathfrak{l}_i]^{h_i - v_i}$ as $\left|\left|\vec{v} - \vec{h}\right|\right|_1 / (um)$ equivalent "legitimate" class group actions. The closest we are to the lattice $\mathcal{L}$, the smallest the representation (via $\vec{v} - \vec{h}$) of class group elements becomes. Naturally, if we manage to obtain a reduced basis of $mathcal L$ which allows to obtain always the closest vector to $\vec{h}$, any class group action evaluation will have exponents in $\{-m, \dots, m\}$ and cost exactly the same as a "legitimate" one. We have:

$$\left|\left|\vec{v} - \vec{h}\right|\right|_1 \leqslant \sqrt{u} \left|\left|\vec{v} - \vec{h}\right|\right|_2 \leqslant u^{3/2} m c^u \ .$$

The multiplicative factor w.r.t the classical group action (mu) is $u^{1/2} c^u$.

Better bound. If the Shortest Vector Problem is solvable exactly for lattices from the CSIDH parameters, the cost overhead of the group action with respect to a legitimate key exchange computation becomes $\sqrt{u}$ (with $c = 1$ as approximation factor), which is smaller than 2^4 for all the proposed instances of CSIDH.

Quantum Circuits for the CSIDH Group Action. The number of quantum gates required to evaluate a CSIDH group action, with exponents in the range $-m \dots m$, has been estimated in full detail in [Ber+19b]. The authors give 765325228976 nonlinear bit operations for the CSIDH-512 instance, in order to reach a success probability of the order 2^{-32}, necessary since we require that many queries. This cost comes mainly

Table 10.2: Number of Clifford + T gates required to compute $[x] \cdot E$ in superposition over the class group $\mathcal{C}\ell(\mathcal{O})$.

Targeted level in [Cas+18]	$\log_2 p$	Number of gates ($\log_2$)
NIST 1	512	48
NIST 3	1024	57
NIST 5	1792	60

from the $2^{21.4}$ multiplications in $\mathbb{F}_p$ needed, each one costing $2^{18.7}$ Toffoli gates, with $\log_2 p = 512$. The number of T gates is $2^{43.3}$. The total number of gates (Clifford + T) is of the order $2^{45.3}$. Furthermore, in order to keep the number of ancilla qubits sufficiently low, some inner levels of uncomputation are needed, possibly increasing the computation by some factor (at least 4). In the end, the quantum CSIDH group action oracle for a prime of 512 bits should cost, in our setting, approx. 2^{48} Clifford+T gates.

This cost is not given in [Ber+19b] for other values of p, but we can roughly estimate the increasing number of multiplications to be performed (counting the increasing dimension, the increasing value of the little primes and the increasing precision needed). Furthermore, when p is doubled, the cost of a multiplication at most quadruples. For CSIDH-1024, we can take 2^{53} gates and 2^{56} for CSIDH-1792. Notice that from the point of view of depth, the oracle contains almost all the depth of the whole circuit (due to the structure of the hidden shift algorithms).

By combining these costs with the approximation factors of 2^4 that we estimated above, we are now able to give an estimation of the number of Clifford + T gates required for an evaluation of $[x] \cdot E$ in superposition over the whole class group $\mathcal{C}\ell(\mathcal{O})$, for a given bit-size of p. Moreover, recently, the structure of the class group of CSIDH-512 has been computed [BKV19], with its reduced basis. Hence, there is overhead for CSIDH-512. This is Table 10.2.

10.4 Concrete cost estimates for CSIDH

This section estimates the cost of different approaches to attack CSIDH, using the estimates of the previous section and the different algorithms presented in Chapter 5.

The security claims of CSIDH are aligned on the NIST security levels. There are multiple ways to interpret them. A simple approach is to consider that the minimal number of evaluations of the key exchange required to break it shall match the number of AES computations in the exhaustive search. In this model, CSIDH does not offer its claimed security, as the parameters have not been based on the most time-efficient algorithms.

Another approach is to consider that the costs have to correspond to a number of quantum gates that any attack needs to match. From Section 11.3, we estimate the time complexity of a key recovery on AES-128, 192 and 256 to respectively $2^{83.7}$, $2^{115.9}$ and $2^{148.7}$ quantum gates.

We can use Algorithm 5.8. From Table 5.5, we can estimate that it costs $2^{1.8\sqrt{n}+2.3}$ quantum queries, time and memory to recover the labels. The shift is then recovered with probability greater than $4/\pi^2$. This leads to the estimates of Table 10.3.

Table 10.3: CSIDH attack cost with Algorithm 5.8 in $\log_2$ scale, compared with the corresponding Grover key-recovery on AES.

Level	Reference AES instance	Grover Cost	Attack quantum time cost	Attack quantum memory cost
NIST 1	AES-128	83.7	80.5	31
NIST 3	AES-192	115.9	101.5	43
NIST 5	AES-256	148.7	115.5	56

Minimal quantum cost. As the quantum queries are very costly in the case of CSIDH, we can use the variant of Theorem 5.6. The quantum time complexity now falls far below the expected level, but the dominating complexity is the cost of the classical subset-sum instance. We estimate that each instance costs $2^{0.291 \log_2 N}$, dismiss the subset-sum polynomial factor, and compare this cost to the classical cost of the AES exhaustive search. We also take into account a factor $8 \log_2 N$ due to the number of subset-sum instances that have to be solved (one for each label produced before the final QFT, and a success probability of $\frac{1}{8}$ in total). We also count as $8(\log_2 N)^2$ the number of quantum queries to the oracle, where N is the cardinality of the class group. This is presented in Table 10.4.

These attacks have a quantum cost that falls far below the limit, with an additional classical cost. Hence, if we consider that the NIST levels allows for both a quantum time below the quantum exhaustive search and a classical time below the classical exhaustive search, the levels NIST 1 and NIST 3 are broken.

Classical/Quantum tradeoffs. We can use the classical/quantum tradeoffs proposed in Table 5.3. We summarize in Table 10.5 the cost with 4-list merging, minimal and equal quantum query and classical memory (excluding polynomial factors). Hence, we considered that we had lists of size $2^{\sqrt{2\log_2(N)/3}}$ everywhere and $\sqrt{\log_2(N)/6}$ steps, and computed the costs accordingly.

Table 10.4: CSIDH quantum attack with minimal quantum cost, in $\log_2$ scale.

Level in [Cas+18]	Expected classical time	Grover cost	$\log_2 p$ in [Cas+18]	Attack Q. time	Attack Class. time/mem
NIST 1	128	83.7	512	$19 + 48 = 67$	86
NIST 3	192	115.9	1024	$21 + 57 = 78$	161
NIST 5	256	148.7	1792	$22 + 60 = 82$	274

Table 10.5: A possible tradeoff with Kuperberg's algorithm, in $\log_2$ scale.

Level	Expected classical time	Grover time	Attack Quantum time	Attack Class. time	Attack Class. mem
NIST 1	128	85.9	73	51	18
NIST 3	192	119.1	88	69	25
NIST 5	256	151.3	97	87	31

10.5 Conclusion

Ordinary isogeny-based schemes and CSIDH are peculiar designs for quantum security: indeed, they are the only known example of key-exchange protocols for which we can have safe instances and for which the gap between the classical and quantum security is more than quadratic. As the quantum cost is subexponential in the size, the choice of a safe set of parameters is arduous. Indeed, a small change in how we estimate the quantum cost can have a high impact in the safe sizes. For CSIDH, if we only consider the asymptotic exponent, then NIST level 1 requires a prime of at least 4096 bits, and 6144 if we allow for more classical than quantum resources, which would represent a drastic increase from the current 512 bits. If the cost of the oracle is taken into account, then smaller instances can be considered safe, but this means that any improvement on the evaluation of the isogenies in the scheme can improve the attack. In particular, a new technique that would make CSIDH fast may also allow to break an otherwise safe instance. How memory is counted can also drastically change the possible tradeoffs for the hidden shift algorithms. While the proposed parameters appear to be too small, in particular in light of Table 10.5, between the many tradeoffs of hidden shift algorithms and the choices in how we count an attack, the size of a safe set of parameters is unclear.

Chapter 11
Quantum security analysis of AES

AES [AES], designed by Daemen and Rijmen, is nowadays the most widely used block cipher. It was the winner of the competition issued by the NIST in 1997 to replace DES [DES], and has been a standard since 2001.

Since its creation, AES has been the subject of an extensive analysis in the classical setting as for instance [DKR97; Fer+01; GM00; KW02; DS08; BA08; Lu+08; BK09; BKN09; Mal+10; DKS10; DFJ13; FJP13; Bou+18]. It is this neverending strong effort that allows us to have trust in its security. However, the capabilities of a quantum adversary against AES have been far less studied. To date, the only work in this direction was the cost estimate of a quantum search on an AES key [Gra+16]. This work allows to assess the cost of the generic attack on AES, but says nothing on the efficiency of dedicated attacks.

In this chapter, we present the firsts quantum cryptanalyses on AES, which mostly come from a paper co-written with María Naya-Plasencia and André Schrottenloher published in the journal ToSC in 2019 [BNS19b]. The description of the algorithms uses the formalism of nested search presented in Chapter 3. Namely, we present a quantum version of the square attack [DKR97] and the Demirci-Selçuk meet-in-the-middle [DS08]. This latter attack has allowed us to find some improved time-memory tradeoffs for the *classical* meet-in-the-middle attacks. We also introduced for this attack a quantum circuit to solve the AES S-Box differential equation, which may be used in quantum attacks against ciphers that use a similar S-Box.

We can see the attacks of this chapter as an assessment of the quantum security of AES, but also, for long-term security, as the first estimate of the security margin of AES in a world where quantum computers are available, in which any attack have to be compared with what had become the standard generic search, quantum search. In such a world, most of the classical cryptanalyses of AES are obsolete, and its security have to be assessed by studying quantum attacks.

Contents

11.1 Description of AES

AES [AES] is a Substitution-Permutation Network alternating between a linear layer, a non-linear layer and round key additions. It has three possible key sizes: 128, 192 and 256. The only differences between the 3 versions are the key schedules and their number of rounds, respectively 10, 12 and 14.

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

Figure 11.1: AES state byte ordering. The figures of this chapter use the work of Jérémy Jean [Jea16].

AES State and Round Function. The cipher encrypts message blocks of 128 bits. The state can be seen as a 16-byte matrix n as an array numbered as in Figure 11.1. The state is then modified by the successive application of a round function, which contains four operations:

- **AddRoundKey** (ARK) xors the round key in the current state,
- **SubBytes** (SB) applies the AES S-Box to each state byte,
- **ShiftRows** (SR), rotates the i-th state row by i bytes to the left,
- **MixColumn** (MC) multiplies each column by the AES MDS matrix.

In the last round, the **MixColumn** is omitted. When considering round-reduced versions, we may either keep it or leave it, but this has little incidence on the cryptanalyses presented in this chapter.

AES finite field. The byte-oriented operations of AES are defined in the finite field $\mathbb{F}_{2^8}$ seen as $\mathbb{F}_2[X]/(X^8 + X^4 + X^3 + X + 1)$.

AES S-Box . The AES S-Box is the only non-linear operation in AES. It is a bijection in $\{0,1\}^8$. It can be written as $S(x) = L(x^{-1}) \oplus c$, with L a linear function and x^{-1} the inversion in the AES finite field (with 0 mapped to 0). It has some very good cryptographic properties. In particular, its differential uniformity is 4, that is, for any pair $(a,b) \neq (0,0)$, the number of solutions of the equation $S(x) \oplus S(x \oplus a) = b$ is at most 4. This is the best we know to be achievable for an 8-bit permutation, and finding if we can have a better permutation is an open problem.

MixColumn. The MixColumn operation multiplies each column of the state, seen as a 4-byte vector, by the following 4×4 matrix defined in the AES finite field:

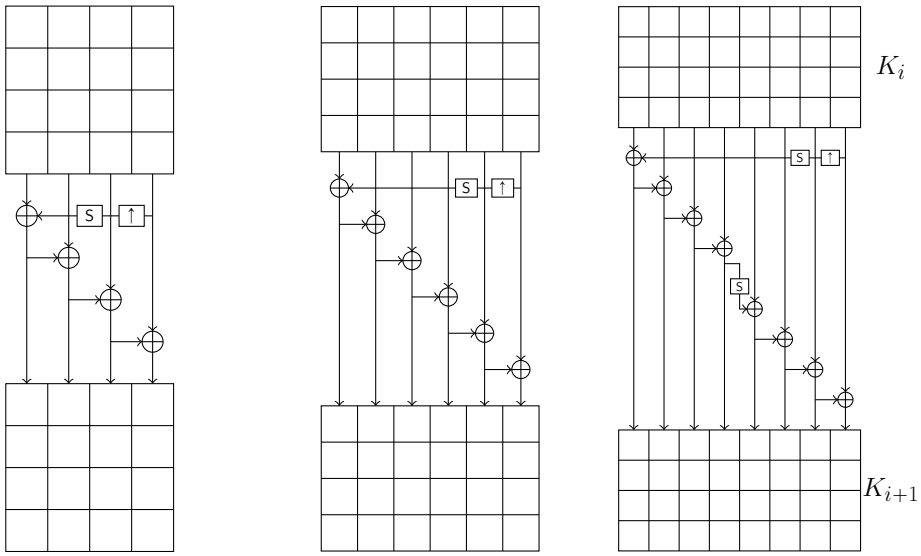
$$\mathrm{MC} = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}.$$

This matrix is Maximum Distance Separable (MDS), which means that for any pair of distinct elements x, x' the sum of the number of bytes that differ between x and x' and the number of bytes that differ between $\mathrm{MC}(x)$ and $\mathrm{MC}(x')$ is at least 5, which is optimal. The name comes from the fact that $(\mathrm{I}|\,\mathrm{MC})$ is the generating matrix of an MDS code.

Key schedule. AES uses one more round key than its number of rounds as there is one final key addition after the last round. As the original secret K is not long enough, it is expanded with the key schedule. It takes the key $K = K_0$, and produces a derived key K_1 which is as long, and iterates until we have enough key bits. While the size of the keys K_i vary, the round keys k_i are always 128-bit long. Hence, for AES-128, the derived keys correspond exactly to the round-keys, for AES-256, one derived key is used for 2 consecutive rounds, and for AES-192, 2 consecutive derived keys are used for 3 consecutive round-keys. The key schedule is drawn in Figure 11.2.

Notations. We write x_i, y_i, z_i, w_i the successive AES states. We note K_i the variable-length keys derived by the key-schedule, and k_i the successive 128-bit round keys and $u_i = MC^{-1}(k_i)$ the "equivalent" round keys, such that adding k_i after the MC operation is equivalent to adding u_i *before* this step. The encryption steps are represented in Figure 11.3. We note $x[0,1,\dots]$ when selecting some bytes from these states. We use the usual AES byte numbering of Figure 11.1. When we consider a pair, the states are denoted x_i, x'_i. We also note $\Delta x_i = x_i \oplus x'_i$. Furthermore, equalities such as $x_1[1,2,3] = x'_1[1,2,3]$ are to be understood byte per byte.

Key-schedule properties. While the round function has a strong design (for example, it ensures total diffusion after two rounds), the key schedule is often considered



(a) AES-128 key schedule (b) AES-192 key schedule (c) AES-256 key schedule

Figure 11.2: AES key schedules. The ↑ denotes a rotation by one byte of the column.

to be the weakest point of AES [DKS10]. In particular, the key-schedule relations can be used to speed up cryptanalysis of reduced-round AES-192 and 256, thanks to the following properties:

Lemma 11.1 (Key schedule relations)**.** *The following properties hold:*

1. *Except for the first column (and column 5 in AES-256), any byte of K_i is equal to the xor of the byte of K_{i+1} at the same position and the byte on its left.*
2. *Each byte of the column 5 of K_i in AES-256 is equal to the xor of the byte of K_{i+1} at the same position and the image through the S-Box of the byte on its left.*
3. *Except for the first two columns (and column 5 and 6 in AES-256), any byte of K_i is equal to the xor of the byte of K_{i+2} at the same position and the byte two cases on its left.*

Proof.

1. From Figure 11.2, except for the first column (and column 5 in AES-256), we have $K_i[x] = K_{i+1}[x] \oplus K_{i+1}[x-4]$.

2. From Figure 11.2, for column 5 in AES-256, we have $K_i[x] = K_{i+1}[x] \oplus S(K_{i+1}[x-4])$.

3. Apply the case 1 for 2 successive keys. □

For AES-256, this lemma shows a simple relation between round keys that are 4 rounds apart.

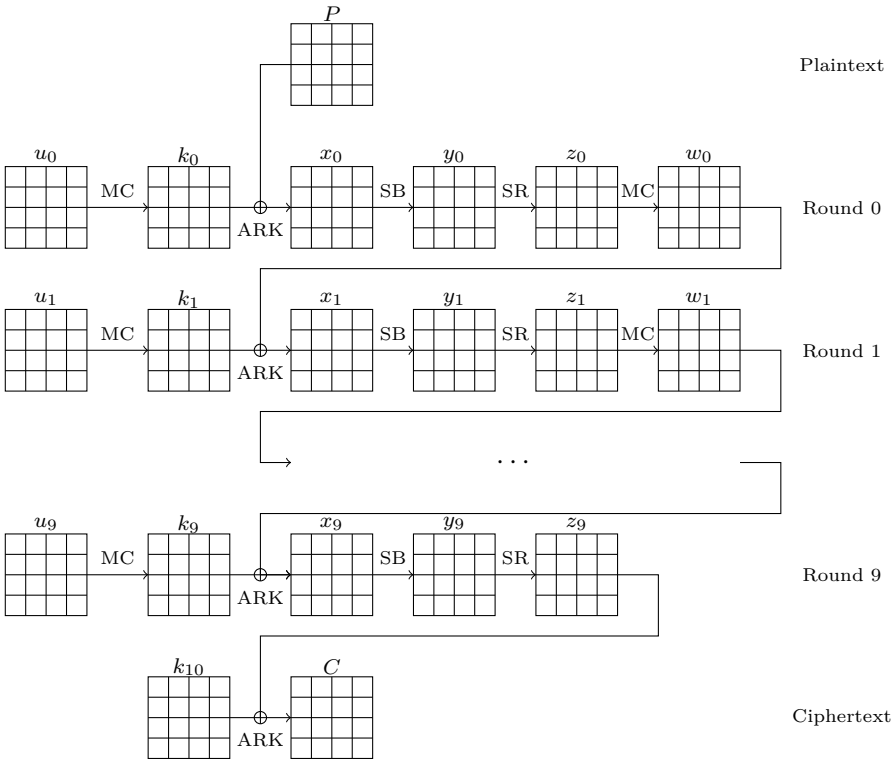


Figure 11.3: Template for the AES state, modified by the AES operations through 10 rounds. Each successive state of the AES round i is noted x_i, y_i, z_i, w_i, and the associated round key is k_i. We note $u_i = \mathrm{MC}^{-1}(k_i)$ the equivalent round key of k_i if it were added before the MC operation and not after. The MixColumn is omitted for the last round.

11.2 Classical cryptanalysis of AES

This section presents the main classical attacks on AES in the secret key model, which are summarized in Table 11.1. It includes some new time-memory tradeoffs we found when we made the quantum attacks presented in this chapter.

Square attack. The square or integral attack have been proposed by Daemen, Knudsen and Rijmen [DKR97]. It relies on the so-called integral distinguisher on 3 rounds of AES, which leverages some properties of the diffusion layer. It targeted 6 rounds, and was extended later to 7 rounds when considering AES-192 and 256 [Fer+01]. It is presented in details in Section 11.4, along with its quantum version.

Demirci-Selçuk meet-in-the-middle. The meet-in-the-middle attack proposed by Demirci and Selçuk [DS08] uses a different distinguisher, based on the fact that the set of differences of some sets of plaintexts encrypted through 4 rounds of AES can only take a restricted amount of values. This attack originally targeted 7 rounds of AES-192

and 8 rounds of AES-256 [DS08], and was later improved to 7 rounds of AES-128, 8 rounds of AES-192 and 9 rounds of AES-256 [DKS10; DFJ13]. A variant of this attack is presented in Section 11.5.

Impossible Differentials. Impossible differential attacks use the fact that some events cannot occur in the cipher (in particular, a differential transition that implies an impossibility through some rounds). This provides a distinguisher for several middle rounds, which is extended a few rounds backwards and forwards, involving some key bits in the path. For good key guesses, the event will not occur by definition, and for bad guesses, it *may* occur. Hence, the attacker sieves the key space by removing the wrong key guesses, and the right one will be the only one left. The best impossible differential attack on AES-128 targets 7 rounds [Bou+18], and provides a comparable tradeoff to the best meet-in-the-middle attacks.

At least with superposition queries, one could hope for an efficient quantum version of an impossible differential, as for a key guess, we can do a quantum search on the pairs that will prove it is a wrong key. However, this simple approach is generally not enough to beat the generic attacks. The most efficient way of building impossible differential attacks is to first obtain a set of pairs that might lead to the impossible middle differential, and next discard the possible keys associated to each pair in a quite efficient way. The good key will be among the ones that have not been discarded. This approach generally uses a large amount of memory, which makes it less interesting for a quantum attack. We took several attempts at quantizing them, but we did not manage to obtain an interesting quantum attack.

Bicliques. Bicliques [BKR11; KRS12] are a generic generic improvement of meet-in-the-middle attacks and have often been used as a generic way to reduce the cost of an exhaustive search by slightly reducing the cost of testing each canditate key. Using bicliques to improve an exhaustive search allows a small gain: for AES it is around a factor 4 [BKR11; Bog+15]. This make the best gain we can expect compared to a simple quantum search around a factor 2, without taking into account the memory and data requirements, which only reduce the efficiency of this approach. Hence, we did not considered bicliques in a quantum setting.

11.3 Generic quantum attacks on AES

We need a cost model and a reference point for generic attacks to assess the efficiency of a quantum attack. The cost of quantum search on AES-128, 192 and 256 has been estimated by Grassl, Langenberg, Roetteler and Steinwandt [Gra+16]. The circuit for AES-128 has been further optimized by Almazrooie, Samsudin, Abdullah and Mutter [Alm+18]. They reduced the required number of qubits by roughly 8%, and the number of Toffoli gates by 0.7%.

These results give a reference point only for the full versions of AES. We only managed to target round-reduced versions, so we need an estimate for quantum search on these smaller AES, which will by definition be cheaper than the search on the full

Table 11.1: Summary of classical cryptanalyses on AES in the single secret key setting. Time is given in equivalent trial encryptions and memory in 128-bit blocks.

Version	Rounds	Data	Time	Mem.	Technique	Reference
Any	6	2^{32}	2^{44}	2^{32}	Square	[Fer+01]
	7	2^{113}	$2^{113} + 2^{80}$	2^{80}	DS MITM	[DFJ13]
	7	2^{105}	$2^{105} + 2^{99}$	2^{90}	DS MITM	[DFJ13]
	7	2^{97}	2^{99}	2^{98}	DS MITM	[DFJ13]
	7	2^{113}	$2^{113} + 2^{84}$	2^{74}	DS MITM	Section 11.5.6
	7	2^{105}	$2^{105} + 2^{95}$	2^{81}	DS MITM	Section 11.5.6
	7	$2^{113.1}$	$2^{113.1} + 2^{105.1}$	$2^{74.1}$	ID	[Bou+18]
	7	2^{105}	$2^{106.88}$	2^{74}	ID	[Bou+18]
192	7	2^{34}	2^{155}	2^{32}	Square	[Fer+01]
	7	2^{99}	2^{99}	2^{96}	DS MITM	[DFJ13]
	8	2^{113}	2^{172}	2^{82}	DS MITM	[DFJ13]
	8	2^{107}	2^{172}	2^{96}	DS MITM	[DFJ13]
256	7	2^{99}	2^{98}	2^{96}	DS MITM	[DFJ13]
	7	2^{34}	2^{172}	2^{32}	Square	[Fer+01]
	8	2^{113}	2^{196}	2^{82}	DS MITM	[DFJ13]
	8	2^{107}	2^{196}	2^{96}	DS MITM	[DFJ13]
	9	2^{113+x}	$2^{210-x} + 2^{196+x}$	2^{210-x}	DS MITM	[DFJ13]
	9	$2^{113+x/2}$	$2^{210-x} + 2^{194+x}$	2^{194+x}	DS MITM	Section 11.5.6

versions. Reducing everything to a number of gates can be quite tedious. In order to simplify the estimates, our reference cost unit is the cost to compute 1 S-Box. This corresponds to most of the cost of the quantum circuits for AES, and multiple parts of our attack compute some parts of AES, which makes it a natural cost unit.

The cost estimates we will use are summarized in Table 11.2. The costs come from the estimates of [Gra+16; Alm+18]. For the reduced versions, we estimated the cost of a reduced version of the circuits of [Gra+16]. The quantum search on an AES key is quite simple: one only needs a few plaintext-ciphertext pairs, and for a given key, check if the encryption of the plaintexts matches the ciphertexts. The number of plaintexts needed to ensure that only the correct key passes the test depends on its size, as the plaintexts are always 128-bit long. As the keys are respectively 128, 192 and 256-bit long, we need respectively 2,2 and 3 plaintexts to ensure that with an overwhelming probability, only the correct key will pass the test. As the search space is of size 2^k, we estimated a time cost of $\frac{\pi}{2} 2^{k/2} \times \text{Cost(AES)} \times \text{pairs}$, with pairs $= 2, 2$ and 3 for respectively AES-128, 192 and 256. For the memory cost, the encryptions of the plaintexts are computed in parallel, except for the key schedule, which is common to all the encryptions. This differs slightly from [Gra+16; Alm+18], which uses respectively $3, 4$ and 5 pairs and a separate key schedule for each encryption.

Table 11.2: Cost benchmarks for AES quantum circuits. Complete versions contains the cost of the key and the key schedule.

Component	Number of S-Boxes	Qubits	Reference
Reference S-Box	1	40	[Gra+16]
Alternative S-Box	0.88	48	[Alm+18]
128-bit full key schedule	40	320	[Gra+16]
192-bit full key schedule	32	256	[Gra+16]
256-bit full key schedule	52	416	[Gra+16]
6-round AES	144	408	
7-round AES	160	536	
8-round AES	192	536	
10-round AES	256	536	[Gra+16]
12-round AES	304	664	[Gra+16]
14-round AES	368	664	[Gra+16]
Complete 6-round AES-128	168	856	
Complete 7-round AES-192	180	984	
Complete 8-round AES-256	224	1208	
Complete 10-round AES-128	294	928	[Alm+18]
Complete 12-round AES-192	336	1112	[Gra+16]
Complete 14-round AES-256	420	1336	[Gra+16]
Quantum search on 6-round AES-128	2^{73}	1265	
Quantum search on 7-round AES-192	$2^{105.1}$	1521	
Quantum search on 8-round AES-256	$2^{137.9}$	2281	
Quantum search on 10-round AES-128	$2^{73.8}$	1521	
Quantum search on 12-round AES-192	2^{106}	1777	
Quantum search on 14-round AES-256	$2^{138.8}$	2665	

11.4 Quantum square attack

In this section, we present the variants of the square attack on AES that have been proposed since its introduction by Daemen, Knudsen and Rijmen [DKR97], and study their quantum counterparts. Classically, it targets 6 rounds, and can be extended to 7 rounds when considering AES-192 and 256 [Fer+01]. This is also the case with our quantum attacks.

11.4.1 The distinguisher

The square attack relies on a distinguisher on 3 rounds of AES (Figure 11.4) which leverages how diffusion is achieved in AES, and is independent of the S-Box. It relies

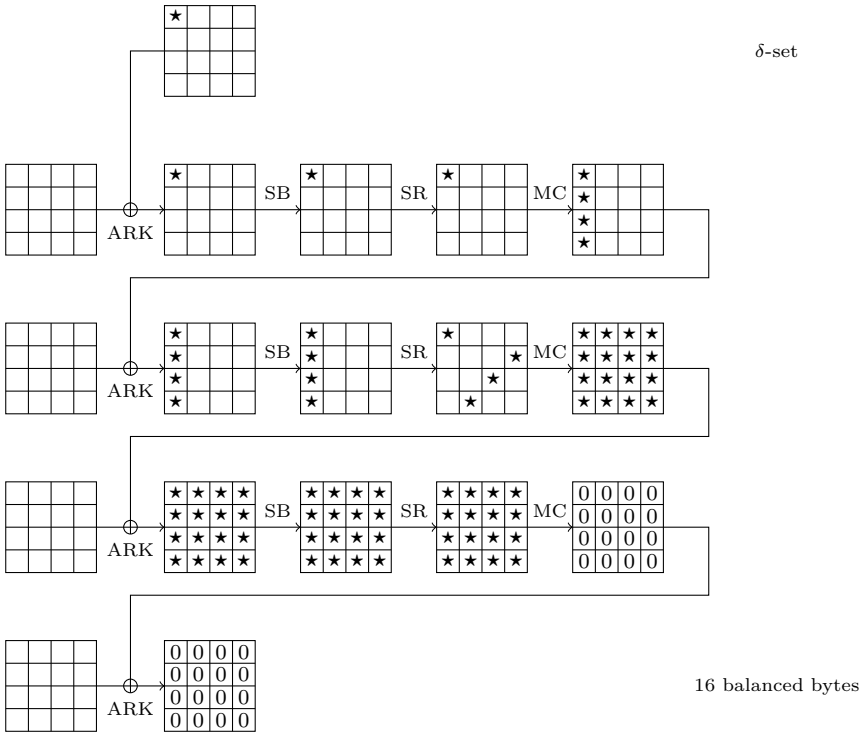


Figure 11.4: Integral distinguisher on 3 rounds of AES. A byte marked by 0 is balanced. Starred bytes take all 256 values in the encryption of the 256 plaintexts. Unmarked bytes are fixed.

on the fact that a 1-byte difference impacts the full AES state after 2 rounds, and until then, the intermediate byte differences do not interfere with each other. Hence, if we consider a δ-set, that is, 2^8 ciphertexts that take all values on one byte but are constant on the other bytes, each byte of the state after two rounds will take all 256 values. This property is kept by the byte-oriented operations SubBytes and ShiftRows. It breaks with MixColumn, but as it is linear, the image of any linear combination of its input will be the linear combination of its output. In particular, the XOR of all 256 values is 0, hence the XOR of all values in each byte after the MixColumn is also 0. This is the distinguisher: this property always happens for 3 rounds of AES, independently of the key. On the contrary, for a random function, the probability that this event happens is 2^{-8} for each byte.

This property holds because as the S-Box is a bijective function, it maps a δ-set to another one. Besides, each byte in the active column at the end of the first round takes 256 values. Finally, the sum of all 256 byte values in any byte position of the last state is equal to the linear combination of some sums of some bytes that each assume all values, and hence is zero.

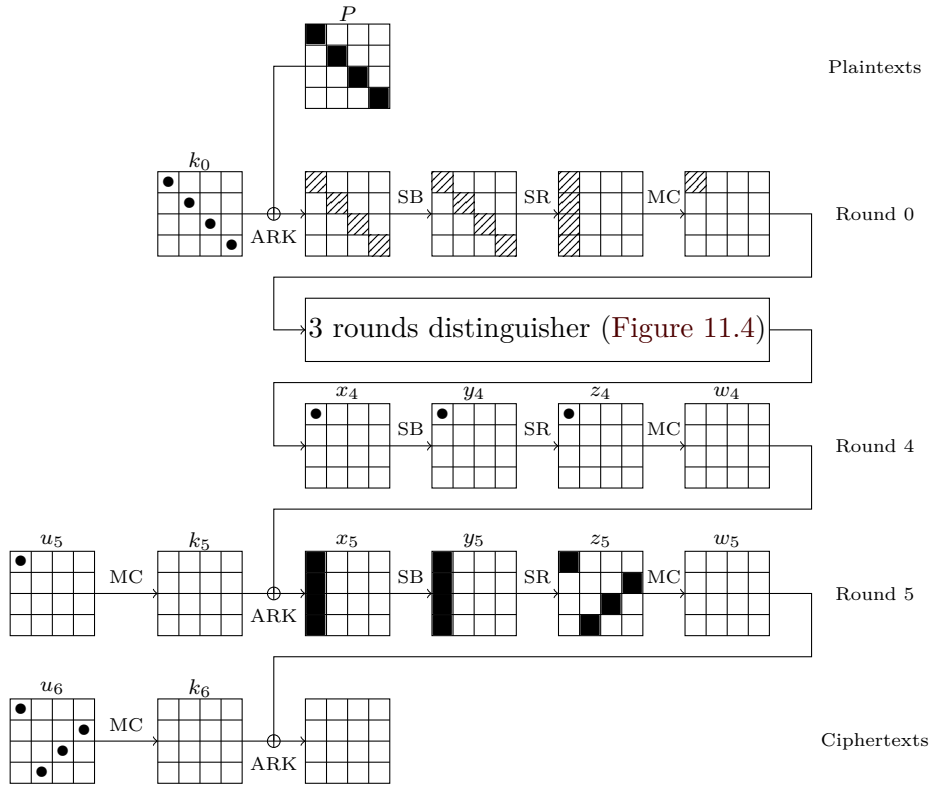


Figure 11.5: Square attack on 6 rounds of AES. • bytes are guessed, and allow to deduce ■ bytes. Unmarked bytes before the distinguisher are fixed. Byte $w_0[0]$ is expected to take all 256 values, and byte $x_4[0]$ is expected to be balanced.

11.4.2 The original square attack on 6-round AES

The original square attack from [DKR97] uses the path of Figure 11.5 and is described in Algorithm 11.1. It guesses some key bytes to obtain the values in 1 byte of the input of the distinguisher and 1 byte of the output, which is enough to check if the 3 inner rounds correspond to an AES. This distinguishes the correct guesses from the wrong ones. As we only check 1 byte of the output of the distinguisher, it keeps a wrong key with probability 2^{-8}. As we guess 9 bytes, we need to repeat the test 10 times, in order to suppress any false positive.

If the guesses are chosen on-the-fly, the classical cost is $10 \times 2^{8\times 9} \times 2^8 \simeq 2^{84}$ adaptative chosen-plaintext encryptions. Each test costs 5 S-Boxes, which are needed for the decryption, for a time cost of around 2^{86} S-Boxes.

Quantumly, we use the framework of Section 3.2. The guess cost is reduced, and we obtain a cost of $\frac{\pi}{2} 2^{36} \times 2^8 \times 10 \simeq 2^{48}$ superposition queries and 2^{50} S-Boxes computations. This approach wastes a lot of queries, as at each iterations we query more than 2^{11} encryptions.

Algorithm 11.1 Square attack on 6-round AES

Input: An oracle to a 6-round AES
Output: The key bytes $k_0[0,5,10,15], u_5[0], u_6[0,7,10,13]$
1: **Filter** $k_0[0,5,10,15], u_5[0], u_6[0,7,10,13]$
2: **loop**10 times
3: From the guess of $k_0[0,5,10,15]$, choose a set of 256 plaintexts that take all the possible values in $w_0[0]$ and are equal in $w_0[1-15]$.
4: Compute the encryption of these values
5: From the guess of $u_5[0], u_6[0,7,10,13]$, compute $x_4[0]$.
6: **If** the sum of all the $x_4[0]$ is not 0: **Abort**
7: **End Filter**

Precomputing the queries. The queries needed for the attack are always fixed on the whole state, except for the diagonal (bytes $(0,5,10,15)$). Hence, a set of 2^{32} plaintexts that are identical, except on bytes $(0,5,10,15)$ where they take all the possible values always contains 2^{24} sets of 256 states suitable for the distinguisher. Hence, we can precompute the queries, that is, choose beforehand the values of the state outside of the diagonal, and query the 2^{32} encryptions. Each test will no longer do any query, but peek among the precomputed set the values it needs. As we only need 10 sets for the test, the 2^{24} available sets are more than enough.

With this precomputation, the classical cost drops to 2^{32} chosen-plaintext queries, and stays at 2^{84} computations. The only overhead is that the queries need to be stocked, which costs 2^{37} bytes of memory.

Quantumly, this will allow to only use classical chosen-plaintext queries instead of superposition queries. In that case, we can either store the 2^{32} values in quantum memory, to remain at 2^{50} quantum computations, or, if we want to avoid quantum memory, we need to sequentially select, at each iterations, the plaintext/ciphertext couples we need, which adds a time overhead, for a total cost of $5 \times \frac{\pi}{2} 2^{36} \times 2^{32} \simeq 2^{71}$ quantum computations. This is barely below the cost of exhaustive search on AES-128 from Table 11.2, at $2^7 3$ computations.

The next sections present the improvements on the square attack from [Fer+01], which can also be applied quantumly.

11.4.3 Improved square attack

The previous precomputation uses the fact that a set of 2^{32} plaintexts with identical bytes except on the diagonal always contains exactly 2^{24} distinct δ-sets. The first key observation from [Fer+01] is that this set is also balanced on each byte of x_4, regardless of the guesses of $k_0[0,5,10,15]$. Hence, the distinguisher can be used without any guess in k_0, at the expense of a larger amount of queries (which can also be precomputed, as they do not depend on any key byte). This allows to reduce the key search space by 32 bits, at the expense of having to compute the sum of 2^{32} values instead of 2^8. This is described in Algorithm 11.2.

The distinguisher still keeps a wrong key with probability 2^{-8}. As we guess 5 bytes, we need 6 sets of 2^{32} plaintexts to uniquely identify the correct key guess. Hence, the classical attack will cost $5 \times 6 \times 2^{5\times 8} \times 2^{32} \simeq 2^{77}$ computations, and $6 \times 2^{32} \simeq 2^{35}$ chosen-plaintext queries. The quantum version has a total cost of $5 \times \frac{\pi}{2} \times 2^{20} \times 6 \times 2^{32} \simeq 2^{58}$ quantum computations, with the same set of 2^{35} chosen-plaintext queries.

In a classical setting, this version is always an improvement, as it needs only 6 times more chosen-plaintext queries but divides the cost by 400. In a quantum setting, the reduction of the key guesses is smaller compared to the overhead of the computation of the sum. Hence, this is not always an improvement compared to the previous algorithm, and only improves the variant with precomputation. Overall, this variant is only relevant if we have access to chosen-plaintext queries and not superposition queries.

Algorithm 11.2 improved square attack on 6-round AES

Output: The key bytes $u_5[0], u_6[0, 7, 10, 13]$
Choose a 6 sets of 2^{32} plaintexts that vary only on the diagonal $(0, 5, 10, 15)$
Compute the encryption of these values
Filter $u_5[0], u_6[0, 7, 10, 13]$
    For each set **do**
        From the guess of $u_5[0], u_6[0, 7, 10, 13]$, compute $x_4[0]$.
        If the sum of all the $x_4[0]$ is not 0: **Abort**
End Filter

11.4.4 Partial sums technique

Another improvement from [Fer+01] consists in remarking that most of the computations are highly redundant, as many intermediate values depend on only a part of the key guesses. Moreover, we have a very limited number of possible values (256 per bytes), which means that a direct computation is highly redundant, and will compute the exact same thing multiple times. Hence, another improvement is to *count* the number of occurrences of each value at each intermediate step and to store it in a table. With this approach, we only need to compute each step of the partial decryption once per possible input.

We want to compute the number of occurrences of each possible value in $x_4[0]$. We have $x_4[0] = S^{-1}(y_4[0])$. We can rewrite it in function of x_5, and obtain

$$x_4[0] = S^{-1}\left(a_0 x_5[0] + a_1 x_5[1] + a_2 x_5[2] + a_3 x_5[3] + u_5[0]\right),$$

where the a_i are the coefficients of the inverse of the MixColumn. With c the ciphertext passed through the inverse of the MixColumn, we have:

$$x_4[0] = S^{-1}\big(a_0 S^{-1}(c[0] + u_6[0]) + a_1 S^{-1}(c[13] + u_6[13]) \\ + a_2 S^{-1}(c[10] + u_6[10]) + a_3 S^{-1}(c[7] + u_6[7]) + u_5[0]\big) \quad .$$

The computation of $x_4[0]$ is split in 4 steps:

- From $(u_6[0], u_6[13])$, compute the list of
$$\left(a_0 S^{-1}(c[0] + u_6[0]) + a_1 S^{-1}(c[13] + u_6[13]), c[10], c[7]\right),$$
- From the previous list and $u_6[10]$, compute the list of
$$a_0 S^{-1}(c[0] + u_6[0]) + a_1 S^{-1}(c[13] + u_6[13]) + a_2 S^{-1}(c[10] + u_6[10]), c[7],$$
- From the previous list and $u_6[7]$, compute the list of
$$a_0 S^{-1}(c[0] + u_6[0]) + a_1 S^{-1}(c[13] + u_6[13]) + a_2 S^{-1}(c[10] + u_6[10]) + a_3 S^{-1}(c[7] + u_6[7]),$$
- From the previous list and $u_5[0]$, compute $x_4[0]$.

Remark 11.1. As we only want to compute the xor of all $x_4[0]$, we only need to know the number of occurrences modulo 2.

Classical cost. The classical time complexity of this procedure (in S-Boxes and inverse S-Boxes) is:
$$\underbrace{2^{2\times 8}}_{\text{Over } u_6[0,13]}\left(6\times 2\times 2^{32} + \underbrace{2^8}_{\text{Over } u_6[10]}\left(2^{24}\times 6 + \underbrace{2^8}_{\text{Over } u_6[7]}\left(2^{16}\times 6 + \underbrace{2^8}_{\text{Over } u_5[0]}\cdot 2^8 \times 6\right)\right)\right).$$
We can bound it by less than 2^{53} S-Boxes. Furthermore, this procedure needs 2^{35} classical queries, 6×2^{24} bits of classical RAM and a memory containing the 2^{35} plaintext-ciphertext pairs.

Quantum equivalent. The quantum equivalent of Algorithm 11.3 performs a nested search, which is detailed in Section 3.2. As its classical counterpart, it uses random-accessible memory. The memory amounts are the same classically and quantumly. The algorithm requires 8×2^{24} qubits and 2^{35} 256-bit registers of *classical* memory to store the chosen-plaintext queries.

The time complexity, adapted from the classical one, is:
$$\left\lfloor \frac{\pi}{2} 2^8 \right\rfloor \left(6 \times 2 \times 2^{32} + \left\lfloor \frac{\pi}{2} 2^4 \right\rfloor \left(2^{24} + \times \left\lfloor \frac{\pi}{2} 2^4 \right\rfloor \left(2^{16} + \left\lfloor \frac{\pi}{2} 2^4 \right\rfloor \times 2^8\right)\right)\right).$$
We obtain a quantum time equivalent of roughly 2^{44} reversible S-Boxes.

11.4.5 Extension to 7 rounds.

The extension to 7 rounds is quite straightforward: we guess the complete round-key k_7. Of course, 7 rounds of AES-128 can no longer be attacked with this approach. As we guess more key bytes, we need to add one structure to check per additional byte to guess.

Algorithm 11.3 Square attack on 6-round AES with the partial sums technique

Input: 6 structures of 2^{32} classical chosen-plaintext queries such that the main diagonal $x_0[0, 5, 10, 15]$ takes all values
Output: The key bytes $u_5[0]$, $u_6[0, 13, 10, 7]$

Filter $u_6[0], u_6[13]$
  For $i \in [1; 6]$ **do**
    $T_1^i = \varnothing$
    For all ciphertext c in structure i **do**
      Compute $(x, y, z) =$

$$\big(a_0 S^{-1}(c[0] + u_6[0]) + a_1 S^{-1}(c[13] + u_6[13]), c[10], c[7]\big)$$

      $T_1^i[(x, y, z)]$^=1
  Filter $u_6[10]$
    For $i \in [1; 6]$ **do**
      $T_2^i = \varnothing$
      For all $(x, y, z) \in T_1^i$ **do**
        Compute $x' = x + a_2 S^{-1}(y + u_6[10])$
        $T_2^i[(x', z)]$^=$T_1^i[(x, y, z)]$
    Filter $u_6[7]$
      For $i \in [1; 6]$ **do**
        $T_3^i = \varnothing$
        For all $(x, z) \in T_2^i$ **do**
          Compute $x' = x + a_2 S^{-1}(z + u_6[10])$
          $T_3^i[(x')]$^=$T_2^i[(x, z)]$
      Filter $u_5[0]$
        For $i \in [1; 6]$ **do**
          $x_4 = 0$
          For all $x \in T_3^i$ **do**
            If $T_3^i[x] = 1$ **then**
              x_4^=$S^{-1}(x + u_5[0])$
          If $x_4 \neq 0$: **Abort**
        Return $u_5[0]$, $u_6[0, 13, 10, 7]$
      End Filter
    End Filter
  End Filter
End Filter

AES-256. For AES-256, a direct guess will lead to a key recovery with a cost of around 2^{183} S-Boxes classically, and 2^{110} S-Boxes quantumly. This can however be improved with Lemma 11.1: the round-key k_7 corresponds to the right part of the 256-bit key K_4, hence we can obtain any value in the last 3 columns of K_3 for free. This corresponds to $k_5[4–15]$. We can perform the attack with any of the 15 bytes of x_4, hence we obtain the guess in u_5 for free. Overall, the cost is 2^{175} S-Boxes classically, and 2^{106} S-Boxes quantumly.

AES-192. We're much closer to the limit with AES-192. Fortunately, the key-schedule allows us to save 2 bytes, as k_5 is the right part of the 192-bit key K_3, k_6 is the left part of K_4, and k_7 is the last two columns of K_4 plus the first two of K_5. With Lemma 11.1, from the two columns of K_5 we can deduce one column of K_4, which corresponds to $k_6[4–7]$. Hence, there is one byte that does not need to be guessed from u_6. The last two columns of K_4 allows to compute the last column of K_3, which corresponds to $k_5[12–15]$. Hence, we can deduce the key guess in u_5 for $x_4[12, 2, 6, 10]$.

With two free bytes in the key guesses, we obtain a cost of 2^{167} S-Boxes classically, and 2^{103} S-Boxes quantumly. This last cost is marginally below the cost of a Grover search, at $2^{105.1}$.

11.5 Quantum Demirci-Selçuk meet-in-the-middle

This section presents a Demirci-Selçuk meet-in-the-middle attack on AES [DS08; DKS10] which can be implemented classically or quantumly. It builds mostly upon the improved version proposed by Derbez, Fouque and Jean [DFJ13].

We first present a technical lemma that shows how to efficiently compute the solutions of the AES S-Box differential equation on a quantum computer. Then, we present the distinguishing properties, that we use in the next section to attack 8 rounds of AES-256. Next, we estimate the cost of the attack, in a classical and in a quantum setting. Interestingly, the quantum attack do not require quantum queries. We finally discuss the interest of the classical implementations of the attack.

11.5.1 S-box differential property

In this section we present an efficient (in time and memory) way to solve the differential equation of the AES S-Box. This operation is classically neglected, as it can be solved with a $2^8 \times 2^8$ lookup table. To make such a table quantum-accessible would mean, however, to use a few kilobytes of quantum RAM. As the cost of using that much memory is still unclear, we have chosen to avoid it, and rely on an efficient low-memory quantum circuit. When counting quantum gates, we rely on the Clifford+T family (see Chapter 2), which we will then translate in an equivalent cost in a number of S-Boxes from [Gra+16].

Lemma 11.2 (S-Box differential property)**.** *Given Δ_x and Δ_y such that $\Delta_x\Delta_y \neq 0$, there exists either zero, two or four pairs x, y, x', y' such that $S(x) = y, S(x') = y'$, $x \oplus x' = \Delta_x$, $y \oplus y' = \Delta_y$.*

There exists a quantum unitary that, given such Δ_x and Δ_y, finds a solution x if it exists and output (x, OK) in this case, and outputs (0, none) otherwise. Its time complexity is around 2 S-Box computations and it uses 22 ancilla qubits. If we only want to know if a solution exists and not an explicit solution, the cost drops to 1 S-Box computation and 15 ancilla qubits.

Proof. The AES S-Box can be written $S(x) = L(x^{-1}) \oplus c$, with L a linear operation and x^{-1} the inversion in $\mathbb{F}_{2^8}$ seen as $\mathbb{F}_2[X]/(X^8 + X^4 + X^3 + X + 1)$ (where 0 is mapped to 0). The main cost of the function is the inversion, which costs around 8 multiplications in the finite field [Gra+16]. Using the same source, we consider that a multiplication costs 981 gates with the multiplier of [Che+08], and L costs 30 gates.

We want to solve the equation

$$S(x) \oplus S(x \oplus \Delta_x) = \Delta_y. \tag{11.1}$$

It can be rewritten as $x^{-1} \oplus (x \oplus \Delta_x)^{-1} = L^{-1}(\Delta_y)$. We note $L^{-1}(\Delta_y)$ as Δ'_y. There are two cases here. If $\Delta_x \Delta'_y = 1$, then 0 and Δ_x are solutions. As there will also be another couple of solutions for the same differences below, it corresponds to the case where the differential equation has 4 solutions.

For every other x, we can multiply the equation by $x(x \oplus \Delta_x)$, and it becomes

$$\Delta'_y x^2 \oplus \Delta_x \Delta'_y x \oplus \Delta_x = 0. \tag{11.2}$$

To solve this quadratic equation, as we are in characteristic 2, we put it in the canonical form

$$(x/\Delta_x)^2 \oplus (x/\Delta_x) \oplus (\Delta_x \Delta'_y)^{-1} = 0. \tag{11.3}$$

We then only need to find a root $R(d)$ of the polynomial $X^2 + X + d$. The solutions will be $\Delta_x R(d)$ and $\Delta_x(R(d) + 1)$.

We do this using the unitary of Lemma 11.3, presented below. The total cost is 7312 gates (or 6864 if we only need to know if a solution exists).

If we only want to know if a solution exists, the complete circuit is:

- Compute $\Delta_x L^{-1}(\Delta_y)$ (uses 8 ancilla qubits and costs 1011 gates).
- Check if a solution exists (6864 gates and 7 ancilla qubits), copy to the output.
- Uncompute $\Delta_x L^{-1}(\Delta_y)$ (costs 1011 gates).

The complete circuit for existence performs on 32 qubits : 16 inputs, 1 output, 15 ancilla qubits, and costs 8886 gates, which is around 1 S-Box computation.

If we want an explicit solution, then the circuit is:

- Compute $\Delta_x L^{-1}(\Delta_y)$ (uses 8 ancilla qubits and costs 1011 gates).
- Check for an explicit solution (costs 7312 gates and uses 14 ancilla qubits).
- Compute Δ_x times the found solution (costs 861 gates) to the output.
- Uncompute the explicit solution (costs 7312 gates).

- Uncompute $\Delta_x L^{-1}(\Delta_y)$ (costs 1011 gates).

The complete circuit to get an explicit solution performs on 47 qubits : 16 inputs, 9 outputs, 22 ancilla qubits, and costs 17507 gates (around 2 S-Box computations). □

Remark 11.2. The cost for the explicit solution can be reduced if we output $x\Delta_x^{-1}$ instead of the real solution x, as we would not need the uncomputation.

Remark 11.3. If we are in a case where 4 solutions exist, the routine will miss 2 solutions. This slightly reduces the success probability (as we fail to find 1 pair in 128), but allows to greatly simplify the generation of a superposition of solutions.

Lemma 11.3 (Solving quadratic equations in characteristic 2)**.** *There exists a quantum unitary that, given $d^{-1} \in \mathbb{F}_{2^8}^*$, outputs a solution of the equation $x^2 \oplus x \oplus d = 0$ in the field $\mathbb{F}_{2^8} = \mathbb{F}_2[X]/(X^8 + X^4 + X^3 + X + 1)$ and a flag indicating if such a solution exists, using 7312 gates and 7 ancilla qubits. If we only need to know if there is a solution, then the cost is reduced by 448 gates.*

Proof. The principle of the circuit is quite simple: it has to check if the input corresponds to one of the precomputed inputs that have a solution, and in that case, write the corresponding solution and set a flag. All of this can be done with a boolean circuit, in 2 steps:

- Check the equality between the register and the precomputed value.
- Copy the precomputed solution if the two values are equal.

We first precompute the 127 d that accept a solution (as the input is d^{-1}, $d = 0$ cannot occur) and their corresponding root $R(d)$, and next check if the corresponding d matches. For the first part, we do a zero test of the xor of the two values. As we check against precomputed values, we can do the check against d^{-1} instead of d, which allows us to avoid doing an inversion. The second part is easy to do once we have computed a control bit that checks if the two values are equal: we only need CNOTs.

We first put the input in the register d, and negate its bits. Next, the equality test consists in the `and` of the seven bits in $d : d_1 \wedge d_2 \cdots \wedge d_7$. We compute it iteratively, with a first ancilla qubit that contains $d_7 \wedge d_6$, a second one that contains $d_7 \wedge d_6 \wedge d_5$, and so on. In order to save on the equality test, we make use of the fact that between two values that we check, we only changed a few bits, hence we do not need to recompute all the partial `and`.

At each step, we xor a fixed value to the d register (which is done with NOT gates) and recompute the `and` from the first affected bit. It is to be noted that the first change can be computed with a CNOT gate (as $a \wedge \neg b = (a \wedge b) \oplus a$), while the other ones need two Toffoli (one to uncompute the preceding computation, one to compute). We then have a bit that checks for equality in our circuit. We CNOT it to an output qubit (that will carry the OK/none information), and do a control-write of the solution associated to the given d to an external register. This can be done with one CNOT per bit at 1 in the solution, as they are precomputed.

We chose the order corresponding to sorting the possible values of d^{-1} in increasing order. The sequence is presented in Appendix 11.6. As two consecutive values are close, the total cost is reduced. In order to compute consecutively all the values for d^{-1}, we need 273 NOT. To check for equality, we need 127 CNOT plus 408 Toffoli (this may be lowered by using another ordering, but we did not investigate further). The writing of the solution needs 448 CNOT. The OK qubit can be updated with one CNOT per step. We also need 14 Toffoli for the initialization and finalization of the equality testing, plus 7 NOT to initialize the first value to be tested. As we do a sequential test, and as the last value we test is 0xff, the input value will be restored at the end. The total number of gates is then $7 + 273 = 280$ NOT, $127 + 127 + 448 = 702$ CNOT and $14 + 408 = 422$ Toffoli. As one Toffoli costs 15 (Clifford+T) gates, the total cost is then of 7312 gates.

As we only write x when we find a match, it will be 0 if there is no solution. If we only need existence check and not an explicit solution, we can reduce the cost by the 448 gates that write the solution. □

Remark 11.4. The sequential test uses $a \wedge \neg b = (a \wedge b) \oplus a$ to compute one $\wedge$ using one CNOT instead of two Toffoli gates. This saves $127(2 \times 15 - 1) = 3683$ gates overall, which is more than half the number of gates in this circuit.

Remark 11.5. For each d, we have chosen the even solution of the equation. Hence, we only need 7 qubits to output the solution. If we want the odd solution, we only need to xor 1 to the even solution.

Other approaches. We also considered different ways to solve this problem. We can test sequentially all the couples (Δ_x, Δ_y). There are 2^{15} of them, so we can estimate that it would cost 100 times more. We could also do a quantum search on the solutions of the equation. As it has 2 S-Boxes, it would cost around 2^5 times more.

Further applications. This analysis can be used for any attack on AES that relies on the S-Box differential equation. Moreover, it can be generalized to other S-Boxes based on the inverse (a complete survey on the use of such S-Boxes can be found in Léo Perrin's PhD thesis [Per17, Section 8.3.1.1]).

11.5.2 Distinguishing properties

Derbez, Fouque and Jean proved the following lemma to attack 7 rounds of AES-128 and 8 rounds of AES-192 and 256, which uses the differential path of Figure 11.6.

Lemma 11.4 (4-round property [DFJ13, Proposition 2])**.** *Suppose that we are given a plaintext-ciphertext pair $(P, P'), (C, C')$ active in one byte i before and one byte j after 4 AES rounds. Consider the plaintexts $P_0 = P, \dots P_{255}$ obtained from P by making the difference in byte i assume all values, that is, P_1 corresponds to adding 1 in byte i, etc. Collect the corresponding ciphertexts $C_0 = C, \dots C_{255}$ and the unordered multiset of differences with C in byte j. There are only 2^{80} possibilities, among the $\binom{2^8+2^8-1}{2^8} \simeq 2^{506}$ 256-byte multisets.*

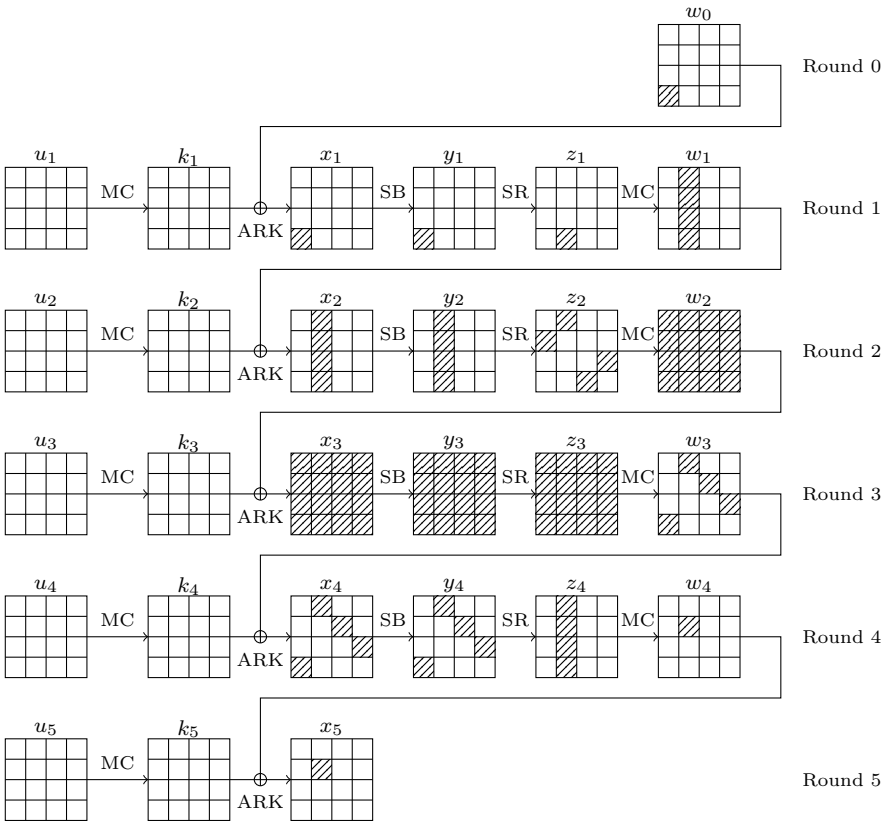


Figure 11.6: 4-round differential path for Lemma 11.4. Active bytes are hatched. The position of the active byte in w_0 and x_5 do not matter.

This property comes from the fact that only 10 bytes (namely, $\Delta z_1[7]$, $x_2[4\text{–}7]$, $z_4[4\text{–}7]$, $\Delta w_4[5]$) determine entirely the multiset when the set of plaintexts-ciphertexts satisfy the path of Figure 11.6.

This distinguisher has been extended to 5 rounds to tackle 9 rounds of AES-256, as represented in Figure 11.7. Its property is given by the following lemma:

Lemma 11.5 (5-round property [DFJ13, Section 4.2])**.** *Suppose that we are given a plaintext-ciphertext pair active in one byte before and after 5 AES rounds. If we make the difference in the input take all* 2^8 *values and collect the multiset of output differences in the output, there are only* $2^{26\times 8} = 2^{208}$ *possibilities among the* 2^{506} *multisets.*

The differential path is the same, with the exception of an additional round in the middle with all the bytes active. As the state has only 128 bits, adding a round adds only 2^{128} possibilities, hence there are only $2^{128+80} = 2^{208}$ possibilities.

Using the property in a quantum attack. The attacks in [DFJ13] use the two previous lemmas by precomputing all the possible values and storing them in a large table. Then, some key guesses allow to perform a partial decryption to check if the

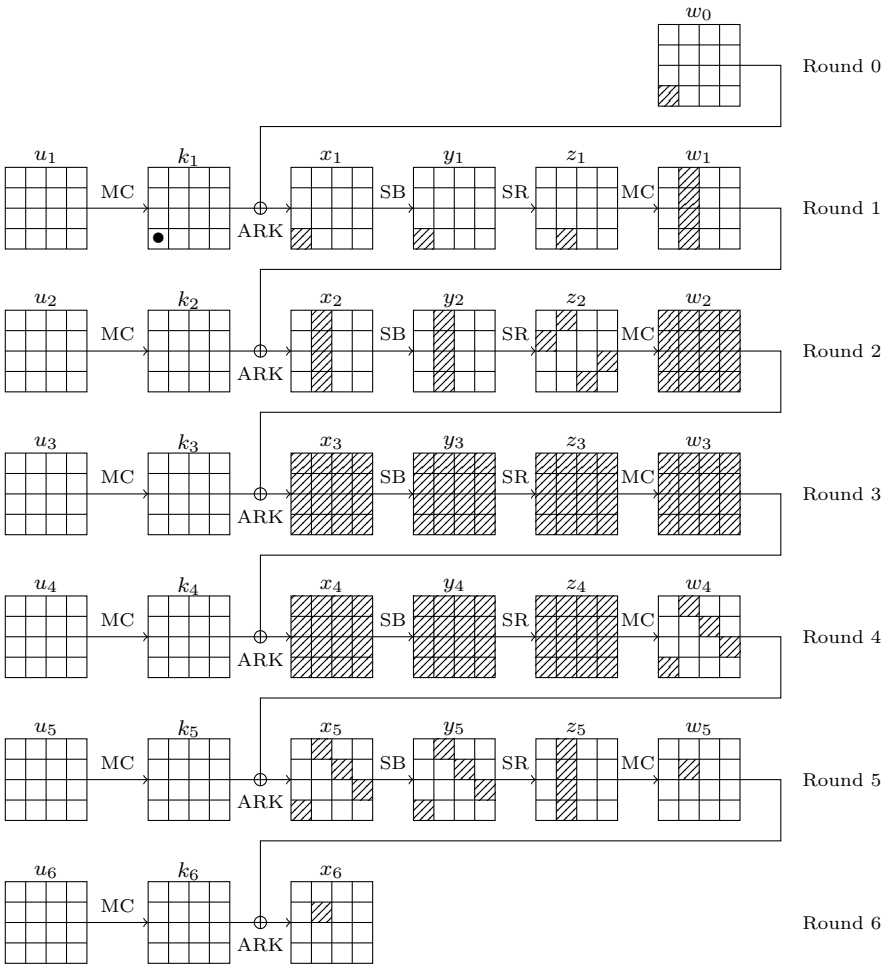


Figure 11.7: 5-round differential path for Lemma 11.6. Active bytes are hatched. The • byte is guessed. The position of the active byte in w_0 and x_5 do not matter.

multiset is possible, which allows to identify the correct key guess. This approach is classically interesting, but its very large memory requirements make it much less appealing for a quantum computer. Hence, instead of using a large array, we would rather compute and check the expected value on-the-fly. This has led us to use a slightly different property, that came back from the multisets introduced in [DKS10] to the original approach from [DS08]: ordered sequences, that we called a *δ-sequence*, as opposed to a δ-set. The quantum attack uses the 5-round distinguisher to attack 8 rounds of AES-256, while classically it is used for 9 rounds. Finally, we only use a part of the whole sequence, as we do not need the 255 values to distinguish 5 rounds of AES. This forces to guess one additional byte of key ($k_1[3]$, as presented in Figure 11.7), but allows to reduce the cost of checking if a sequence passes the test.

Definition 11.1 (δ-sequence)**.** Consider a pair P, C which satisfies the full differential path of Figure 11.8. Make the single-byte difference $\Delta x_1[3]$ assume the sequence of val-

ues $1, \dots 32$. We name *δ-sequence* the corresponding sequence of single-byte differences $\Delta x_6[5]$.

Lemma 11.6 (5-round property for sequences)**.** *Suppose that we are given a plaintext-ciphertext pair $(P, P'), (C, C')$ active in one byte i before and one byte j after 5 AES rounds. Suppose also that the input and output differences in these bytes are given. Consider the plaintexts $P_0 = P, \dots P_{32}$ obtained from P by making the difference in byte i assume all values from 1 to 32. Collect the corresponding ciphertexts $C_0 = C, \dots C_{32}$ and the ordered sequence of differences with C in byte j. Then there are only 2^{192} possibilities among 2^{256}.*

Proof. The proof follows that of Lemma 11.5, except that, since the input and output differences of the pair are known, the whole space of δ-sequences is reduced by 2^{16} (two bytes) in size. □

11.5.3 The attack

The attack extends the previous distinguisher by adding one round before and 2 rounds after it. It requires to guess 9 additional key bytes to gain access to the values of interest for the distinguisher. It is summarized in Figure 11.8 and Algorithm 11.6. It uses Lemma 11.6 to identify the correct key guesses. From Lemma 11.6, we obtain that a random key can pass the test with probability 2^{-64}, and we test 80 bits. As this is not precise enough, we will use key schedule relations of Lemma 11.10 to reduce the number of possibilities.

In the attack, we will first compute enough plaintext-ciphertext pairs so that, given a guess for the outer key bytes (denoted by • in Figure 11.8), we find one that satisfies the middle round differential (y_1 to x_6). After that, we compute the corresponding δ-sequence, by making the difference in $x_1[3]$ assume the values $1, \dots 32$. This δ-sequence is of length 256 bits. With Lemma 11.6, we ensure that the sequence takes one out of 2^{192} possibilities, as it is determined by 24 state and key bytes.

If the guess of the key bytes • is correct, then the computed δ-sequence can be obtained by some choice of these 24 inner byte-conditions. To verify this, we will do another Grover search. If the guess of the bytes • is not the good one, then with high probability, the δ-sequence that we computed will not appear.

Overall, the attack works in 5 steps, for each key guess:

- Find a pair that matches the differential path of Figure 11.6,
- Compute the corresponding δ-sequence,
- Guess the inner state bytes,
- Filter out the guesses that do not match the key schedule constraints,
- Check if the state guesses match the δ-sequence.

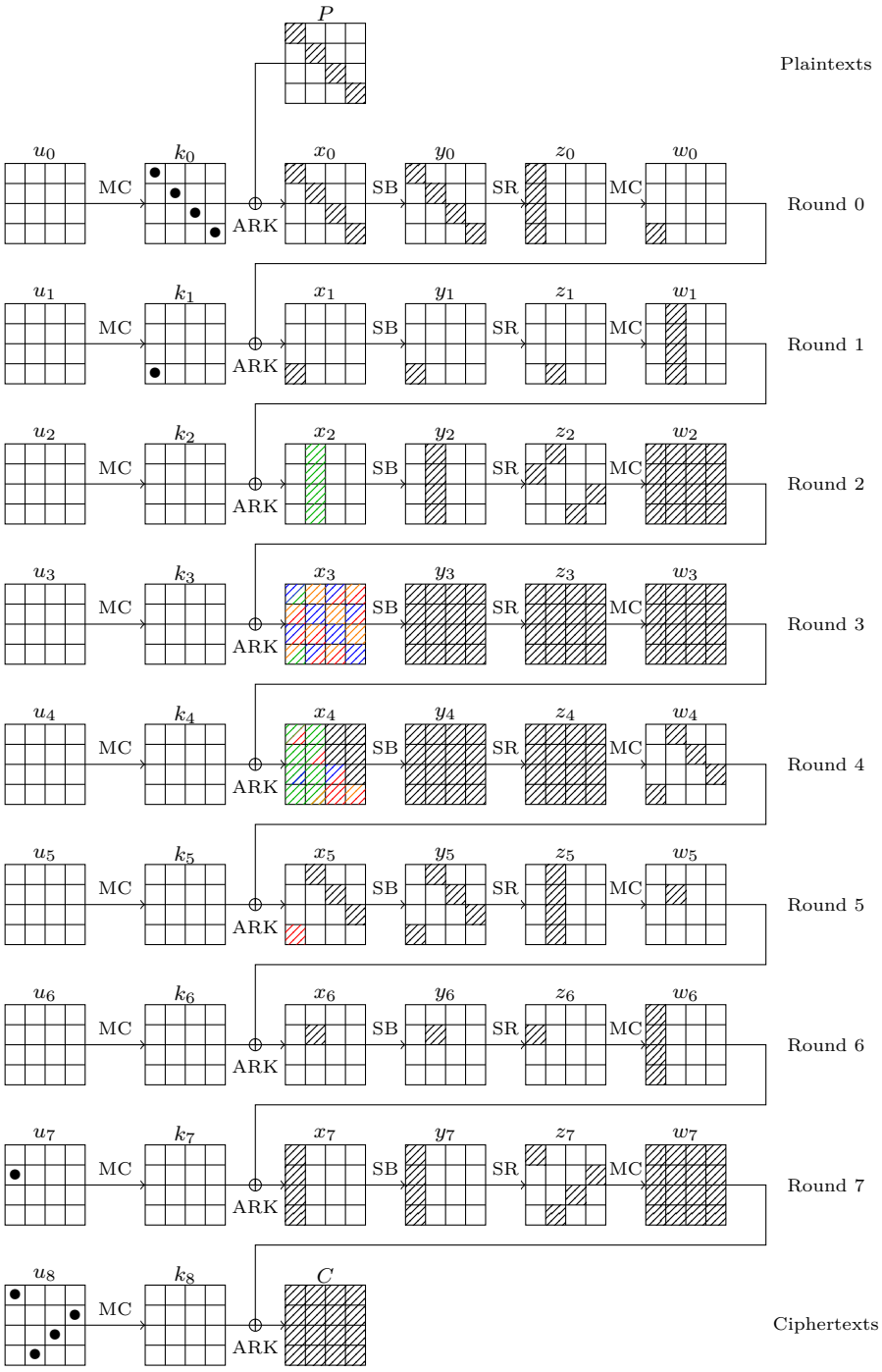


Figure 11.8: Full differential path used in the quantum attack. Key bytes guessed in the outer Grover procedure are denoted by •. Colored bytes are constrained by the corresponding equation described in Figure 11.9.

11.5.3.1 Finding the pairs.

We use the same method as in [DFJ13] to obtain the pairs. This part is classical, as it is a precomputation whose running time is below what we expect of the quantum search.

Lemma 11.7 (Finding pairs)**.** *Algorithm 11.4, with* 2^{113} *encryption queries, returns* 2^{48} *plaintext-ciphertext pairs* P, C, P', C' *such that:*

- *The difference* $\Delta P = P \oplus P'$ *is active only in bytes* $0, 5, 10, 15$*,*
- *The difference* $\mathrm{MC}^{-1}(\Delta C) = \mathrm{MC}^{-1}(C \oplus C')$ *is active only in bytes* $0, 7, 10, 13$*.*

Proof. Each set of 2^{32} plaintexts that varies on the diagonal contains $2^{32}(2^{32}-1)/2$ pairs, and the output difference is correct with probability 2^{-96}. Hence, each set contains one pair with probability 2^{-33}. We will obtain 2^{48} pairs after $2^{48+33} = 2^{81}$ sets, which corresponds to 2^{113} encryption. □

Algorithm 11.4 Pair generation

$L \leftarrow \varnothing$
loop 2^{81} times
    Choose a plaintext P.
    Encrypt the 2^{32} values that correspond to P plus a difference in bytes $0, 5, 10, 15$.
    For all 2^{63} pairs $(P, C), (P', C')$ of plaintexts/ciphertexts **do**
        If $\mathrm{MC}^{-1}(C \oplus C')$ is active only in bytes $0, 7, 10, 13$ **then**
            Add (P, P') to L
Return L

Algorithm 11.5 Finding a matching pair

Input: The list of 2^{48} pairs of Lemma 11.7, the key guesses of $k_0[0, 5, 10, 15]$ and $u_8[0, 7, 10, 13]$.
Output: A pair that fulfills the path of Figure 11.6.
For all Pairs in the list **do**
    Use the guesses to compute $\Delta w_0[0\text{–}3]$ and $\Delta z_6[0\text{–}3]$
    If $\Delta w_0[0\text{–}2] = 0$ and $\Delta z_6[0, 2, 3] = 0$ **then**
        Return the pair

Lemma 11.8 (The Good Pair)**.** *Algorithm 11.5 finds a matching pair for the differential of Figure 11.6 in approximately* 2^{53} *S-Box computations.*

Proof. We test sequentially each of the 2^{48} possible pairs, and expect that one pair will pass the test. As this is a sequential test, we do the same thing classically and quantumly. There are 16 S-Box computations to do for each pair (4 in round 0 and round 8 for both members of the pair), to check if it is the good one. When we compute quantumly, we need to uncompute each test, which adds a factor 2. □

11.5.3.2 Computing the δ-sequence.

From the pair, we need to compute the δ-sequence. The associated plaintexts are computed thanks to $k_1[3]$ and $k_0[0, 5, 10, 15]$. We encrypt these 2^5 plaintexts, and partially decrypt the ciphertexts thanks to our guesses of u_8 and u_7 in order to obtain the sequence of differences in $x_6[5]$. This list contains 2^5 byte values, hence 256 bits are sufficient to store it.

Lemma 11.9 (Computing the δ-sequence)**.** *Given the subkey guesses in k_0, k_1, u_7, u_8, and a pair satisfying the inner differential, we can compute the corresponding δ-sequence using 2^{13} S-Box computations.*

Proof. Each of the 2^5 elements of the δ-sequence requires a call to the secret-key oracle, which costs 2^8 S-Boxes. The other computations are negligible. □

11.5.3.3 Number of possible δ-sequences.

There are 10 key bytes (•) to guess. For each of these key bytes, Lemma 11.6 gives 24 more byte-degrees of freedom to go through, in order to test all δ-sequences. So there would be a total number of 34 bytes to sieve, which is higher than the 32 bytes of exhaustive search of the key. We reduce this crucially by making use of the key schedule relations (Lemma 11.10), which are translated to 4 one-byte state equations (11.4), (11.5), (11.6) and (11.7). This reduces the total search space to 30 bytes, and allows the success of our approach on 8 rounds of AES-256.

Key-schedule relations. We use the following relations to obtain some state equations which allow to reduce the search space.

Lemma 11.10 (AES-256 key schedule properties)**.** *Let $k_0, \dots k_8$ be the 8-round expansion of the key-schedule of AES-256. The following relations hold:*

1. $k_0[10] = k_4[2] \oplus k_4[10]$
2. $k_0[15] = k_4[7] \oplus k_4[15]$
3. $k_2[4\text{–}7] = k_4[0\text{–}3] \oplus k_4[4\text{–}7]$
4. $k_5[3] = k_1[3] \oplus S(k_4[15]) \oplus S(k_4[11] \oplus k_4[15])$

Proof. Relations 1,2 and 3 are direct applications of Lemma 11.1. For relation 4, Lemma 11.1 tells us that $k_3[3] = S(k_4[15]) \oplus k_5[3]$, $k_1[3] = S(k_2[15]) \oplus k_3[3]$, and $k_2[15] = k_4[11] \oplus k_4[15]$. Combining the 3 equations produces the wanted equality. □

State equations. Equations (11.4), (11.5), (11.6) and (11.7) below are respectively derived from the four key-schedule relations given in Lemma 11.10. We replace some key bytes by the sum of known state bytes, up to linear transformations.

$$k_0[10] = k_4[2] \oplus k_4[10]$$
$$\implies k_0[10] = x_4[2] \oplus x_4[10] \oplus \ell_2(y_3[0,5,10,15]) \oplus \ell_2(y_3[8,13,2,7]) \quad (11.4)$$

$$k_0[15] = k_4[7] \oplus k_4[15]$$
$$\implies k_0[15] = x_4[7] \oplus x_4[15] \oplus \ell_3(y_3[3,4,9,14]) \oplus \ell_3(y_3[1,6,11,12]) \quad (11.5)$$

$$k_5[3] = k_1[3] \oplus S(k_4[15]) \oplus S(k_4[11] \oplus k_4[15])$$
$$\implies x_5[3] \oplus \ell_3(y_4[0,5,10,15]) = k_1[3] \oplus S\Big(x_4[15] \oplus \ell_3(y_3[1,6,11,12])\Big)$$
$$\oplus S\Big(x_4[15] \oplus \ell_3(y_3[1,6,11,12]) \oplus x_4[11] \oplus \ell_3(y_3[8,13,2,7])\Big) \quad (11.6)$$

$$k_2[4\text{–}7] = k_4[0\text{–}3] \oplus k_4[4\text{–}7]$$
$$x_2[4\text{–}7] \oplus \mathrm{MC}(y_1[3,4,9,14]) =$$
$$x_4[0\text{–}3] \oplus \mathrm{MC}(y_3[0,5,10,15]) \oplus x_4[4\text{–}7] \oplus \mathrm{MC}(y_3[3,4,9,14])$$
$$\implies \ell(x_2[4\text{–}7]) \oplus y_1[3] = \ell(x_4[0\text{–}3]) \oplus y_3[0] \oplus \ell(x_4[4\text{–}7]) \oplus y_3[3] \quad (11.7)$$

where ℓ_2 and ℓ_3 are the linear functions that, on input a column, give the third (resp. fourth) byte of this mixed column, and ℓ is the linear function which, on input a column C, gives the first byte of $\mathrm{MC}^{-1}(C)$.

Sieving with the state equations. At some point in our attack, we have two choices for each byte of $x_2[0\text{–}3]$ (one column of x_2), each byte of x_3, each byte of x_4 and each byte of $x_5[3,4,9,14]$. We then sieve these possible choices with the 4 key relations obtained above, translated into relations between the bytes of these states. As there are 40 bit-degrees of freedom and 4 byte constraints, we expect 2^8 possibilities to pass. These relations are expected to constrain completely 32 of the byte values and leave the 8 others free. This is represented in Figure 11.9, with the bytes of x_2, x_3, x_4 and x_5 concerned. For each relation, (11.4) to (11.7), we represent the bytes of the states that appear in it. These values are always either mixed or passed through an S-Box, so we may consider the relations to be independent. In the end, 8 bytes appear in none of the relations: these are exactly the 8 free choices remaining.

11.5.3.4 Final matching

From the guesses that pass all the previous tests, we can compute the expected δ-sequence and match it against the one computed in the previous step.

Algorithm 11.6 The attack on 8-round AES-256

Input: The 2^{48} pairs of Lemma 11.7
Output: The key bytes $k_0[0,5,10,15]$, $k_1[3]$, $u_7[1]$, $u_8[0,7,10,13]$
Filter $k_0[0,5,10,15]$, $k_1[3]$, $u_7[1]$, $u_8[0,7,10,13]$
  Find a matching pair with Algorithm 11.5 ▷ 2^{53} S-Boxes
  Compute the δ-sequence in $x_6[5]$ by making $x_1[3]$ vary
  Compute $x_1[3], x_1'[3]$ and deduce $\Delta x_2[4\text{–}7]$
  Compute $x_6[5], x_6'[5]$ and deduce $\Delta y_5[3,4,9,14]$
  Filter $\Delta y_2[4\text{–}7]$, $\Delta x_5[3,4,9,14]$, Δx_4
    If $\Delta y_2[4\text{–}7]$ and $\Delta x_2[4\text{–}7]$ do not match: **Abort** ▷ prob. 2^{-4}
    If $\Delta x_5[3,4,9,14]$ and $\Delta y_5[3,4,9,14]$ do not match: **Abort** ▷ prob. 2^{-4}
    Compute the possible values of $x_2[4\text{–}7], x_5[3,4,9,14]$ with Lemma 11.2
    From $\Delta y_2[4\text{–}7]$, compute Δx_3
    From $\Delta x_5[3,4,9,14]$, compute Δy_4
    If Δx_3 and Δy_4 do not match: **Abort** ▷ prob. 2^{-32}
  ▷ At this point, one guess over 2^{40} has passed the S-Box differential equations
    Compute the possible values of x_3, x_4 with Lemma 11.2
    Match $x_4[2,10], x_3[0,5,10,15,8,13,2,7]$ with Equation 11.4 ▷ 4 solutions expected
    Match $x_4[7,15], x_3[3,4,9,14,1,6,11,12]$ with Equation 11.5 ▷ 4 solutions expected
    Match $x_5[3], x_4[0,5,11]$ with the remaining $x_3[1,6,11,12,\ 2,7,8,13]$ and Equation 11.6 ▷ 1 solution expected
    Match $x_2[4\text{–}7], x_4[1,3,4,6]$ with the remaining $x_4[0,5,2,7]$, $x_3[0,3]$ and Equation 11.7 ▷ 1 solution expected
    For all Choices for $x_4[8,9,12,13,14]$ and $x_5[4,9,14]$ **do**
      Compute the expected δ-sequence in $x_6[5]$
      If it matches the computed δ-sequence **then**
        Return $k_0[0,5,10,15]$, $k_1[3]$, $u_7[1]$, $u_8[0,7,10,13]$
  End Filter
End Filter

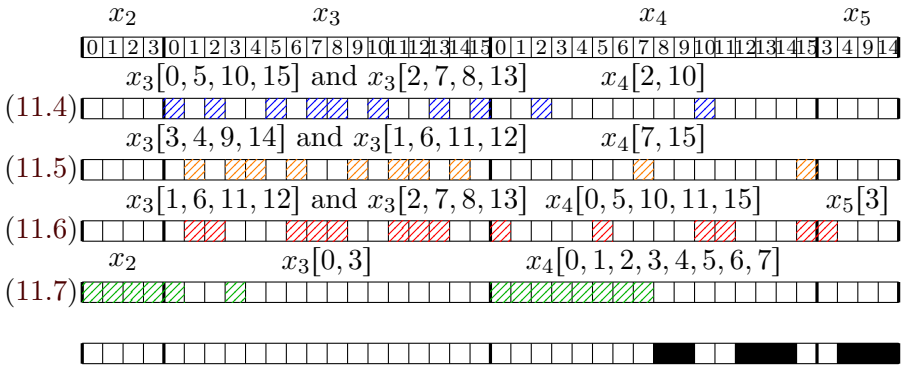


Figure 11.9: All state relations (11.4), (11.5), (11.6), (11.7) and the 8 remaining unconstrained bytes, in black.

11.5.4 Complexity analysis

11.5.4.1 Classical complexity

There are three levels of filtering. We go from the outermost to the inner one:

1. Filtering on key byte guesses: there are $2^{10\times 8}$ guesses to look at, among which we expect exactly one solution. Given these byte guesses, we find a good pair and start computing the states in the middle. The cost is in total:

$$2^{80}\left(2^{53}+\mathbf{f}\right)$$

   where $\mathbf{f}$ is the next filter. The 2^{53} term comes from Lemma 11.8.

2. Filtering on the 16 + 8 differences: there are 40 S-Box differential equations to solve in total. In the middle, we can match x_4 and y_4 column by column: this is more efficient than solving all 32 equations at once. At this point, we have obtained 2 possibilities for the full sequence of states, as each state byte of $x_2[4\text{–}7], x_3, x_4, x_5[3,4,9,14]$ has two possibilities. We then pass the key-conditions.

   a) Equations 11.4 and 11.5 each need 2^{10} computations, without S-Boxes involved. This is negligible.

   b) To check Equation 11.6 for all possibilities, one needs to compute some S-Box evaluations. Indeed, due to the constrained choices, $x_4[15]\oplus\ell_3(y_3[1,6,11,12])$ can take on average only 4 values and there are only 8 possible values for $x_4[11]\oplus\ell_3(y_3[8,13,2,7])$. So in total we need not more than $4+8$ S-Boxes evaluations, which is negligible.

   c) Again, to check Equation 11.7, we need only linear computations, without any S-Boxes.

   So the full cost of the filter is:

$$2^{(16+8)\times 8-40}\left(16+4\times(2^8\times 8)+\mathbf{f}'\right)$$

where 16 stems from the outer S-Box equations, $4 \times (2^8 \times 8)$ is the term for the inner columns equations, and **f**' is the next filter, and 2^{-40} is the probability of one guess of verifying the differential equations.

3. Filter on the state sequences: there are 8 remaining bit-degrees of freedom, that is, bytes that can take two values. For each possibility, we compute the δ-sequence using $2^5 \times 40 = 1280$ S-Box evaluations and match against the wanted one. We expect one or zero solution. The cost is $2^8 \times 1280$.

In total we obtain

$$\mathbf{2^{80}} \left(2^{53} + \mathbf{2^{152}} \left(\left(4 \times (\mathbf{2^8} \times 8)\right) + \mathbf{2^8} \times 1280\right)\right),$$

where we have highlighted the search terms.

A direct computation gives a classical complexity of $2^{250.3}$ S-Boxes, which is actually the optimal cost for our algorithm. Indeed, the differential path we use contains in total 30 byte-degrees of freedom (including key byte guesses), when discounting the key-schedule relations. The best expected complexity is then 2^{240} times the computation of a δ-sequence, which is exactly what we get (δ-sequences are the dominant term). We now turn ourselves towards the quantum time complexity of this procedure.

11.5.4.2 Quantum complexity

The main difference with the classical analysis is that we need to estimate the number of solutions at each step, and cannot afford a rare event in which we have much more or much fewer solutions, while classically all balances out. While the classical analysis uses the average number of solutions and allows us to choose the number of iterations we perform, the actual number of solutions at each step for the correct answer may differ from the average value.

A simplified estimation of nested search would lead to a time cost of

$$\frac{\pi}{2}2^{40}\left(\frac{\pi}{2}2^{76}\left(2^5\frac{\pi}{2}2^4 + \frac{\pi}{2}2^4 \times 1280\right)\right) \simeq 2^{132.3} \text{ S-Boxes.}$$

As this estimate is fairly close to the cost of the exhaustive search (at $2^{137.9}$ S-Boxes), we propose below a more precise estimate, in order to show that we indeed beat it, and the attack on 8 rounds of AES-256 works.

S-Box property. The differential property can yield 4 solutions, and if we only consider two solutions, as we go through 40 S-Boxes, we succeed with probability $\left(\frac{127}{128}\right)^{40} \simeq 2^{-0.45}$. If we want to be more precise in the analysis, we can remark that the differential property is not fulfilled for half of the differentials, but for $\frac{127}{255}$ of the non-zero differentials. Moreover, we can restrict our search space over non-zero differentials.

Hence, we do not have 2^{192} differentials, but $2^{191.86}$, and the 40 differential equations filter $2^{40.23}$ values, for a total space size in the second filter of $2^{151.64}$.

Precision. For each filter, we estimate how much it can deviate from the average number of solutions:

1. Filtering of $k_0[0,5,10,15]$, $k_1[3]$, $u_7[1]$, $u_8[0,7,10,13]$: the test function is expected to accept only the good key guesses, the space search is trivial, so no deviation here.

2. Filtering of $\Delta y_2[4\text{–}7]$, $\Delta x_5[3,4,9,14]$, Δx_4: we filter $\Delta y_2[4\text{–}7]$ and $\Delta x_5[4,9,14]$ by comparing with Δx_2 and Δy_5, and there are exactly 127 solutions per byte. Hence, there is no deviation here. However, for Δx_4, there are two independent constraints: one from Δy_2 and one from Δx_5. This can make the actual number of solutions vary.

3. Sequential test of the 4 key conditions: here, the number of solutions can also vary. As we do a sequential test, we only have to care about the maximal number of solutions that will occur for the correct guess.

4. Last filter: there are exactly 2^8 tuples to iterate on. This has to be done once for each solution that might arise from the previous step. As we know the list of these solutions, we can also perform a quantum search on them.

For the variations of the differences, we have been able to simulate them column by column, and found that the number of solutions when fixing Δx_3 (deduced from Δy_2) and Δy_4 (deduced from Δx_5) were in 98% of the cases in the interval $2^{27.95}(1 \pm 2^{-9})$. As we have 4 columns, we can estimate that in more than 90% cases we will have a number of solutions that varies of a factor less than 2^{-7} around the mean. Hence, the output of the circuit that produces $\Delta y_2[4\text{–}7]$, $\Delta x_5[3,4,9,14]$, Δx_4 will not pass the outer test with probability $2^{-151.6}$, but with a probability between $2^{-151.6}(1-2^{-7})$ and $2^{-151.6}(1+2^{-7})$. If we do $\frac{\pi}{4}2^{76}$ iterations, we obtain a value that passes the test with probability greater than $1-2^{-14}$, which will force a negligible increase in the number of outer iterations and reduce marginally the final success probability.

Regarding the key conditions, we consider that overall, the 4 equations for the good path (there is only one, corresponding to a good guess of all subkey bytes and state differences) might have 4 solutions. As the candidates (and their number) are known, we can generate the superposition of all of them, to be used in the final test. The overhead to generate this superposition is negligible, but this adds a factor 2 to the number of iterations. Hence, the final complexity, taking into account the success probability is

$$\frac{10}{9}2^{0.45}\frac{\pi}{2}2^{40}\left(\frac{\pi}{2}2^{75.8}\left(2^7\times 2\left\lfloor\frac{\pi}{4}2^4\right\rfloor+2\left\lfloor\frac{\pi}{2}2^{4+1}\right\rfloor\times 1280\right)\right)=2^{134.7}\text{ S-Boxes.}$$

The dominating term is the computation of δ-sequences. The cost is below exhaustive search, at $2^{137.9}$ S-Boxes.

11.5.5 Removing the superposition queries

The previous attack computed the queries to the cipher on-the fly, which forced us to have access to superposition queries. It is possible to replace them by classical ones.

Suppose that we are given a guess of 9 key bytes in k_0, k_1, u_8 and a corresponding good pair. To produce the expected δ-sequence, there are encryption queries to perform, making the difference in y_1 vary. These chosen-plaintext queries depend on the 4 guessed bytes of k_0 and the guessed byte of k_1. But this represents only 2^{40} values. This means that the whole procedure needs only $2^{48} \times 2^{40} = 2^{88}$ distinct queries, grouped by their corresponding pairs.

We now perform all these queries beforehand. Instead of computing the δ-sequence on the fly, we go through the set of stored queries and find the ones that interest us (those which correspond to the current pair). This is done with a sequential lookup of the 2^{88} values, for a grand total of 2^{128} lookups. This will not be the dominant term, as it does not involve any S-Box. It only forces us to store classically 2^{88} plaintext/ciphertext pairs, instead of 2^{48}.

11.5.6 Quantum-inspired classical attacks

Managing to make the attacks work quantumly was non-trivial. The memory was especially an issue, there is no generic gain on memory as there is on time. This has forced us to seek lower-memory approaches, which can also be applied classically. We propose in this section new classical attacks, which are summarized in Table 11.1.

Re-ordering the steps. The main idea that helps reducing the memory needs is to first store the results from the previous online phase (key guesses, corresponding pairs and multisets computed from the queried messages), and next perform an exhaustive search over the middle values (what was before a precomputed table), and look for a collision. When the first term is smaller, the memory is reduced, while keeping similar complexities of data and time (as we are basically doing the same computations in a different order).

Further possible improvements. Using multiple differentials in the middle and storing the transitions will allow to provide attacks with reduced data, while partially increasing the previously reduced memory. We believe new interesting trade-offs might result from these combination.

11.5.6.1 Improved attack on 9-rounds AES-256

We consider the 9-round AES-256 attack of [DFJ13, Fig.6]. In this attack, after having obtained 2^{144} plaintext-ciphertext pairs verifying the input differential with 2^{113} queries, we sieve the outer key bytes. Each pair gives 2^{48} possible values for $k_{-1}[0, 5, 10, 15]$, k_8, u_7, that can be enumerated in time 2^{48}, such that it verifies the whole differential pattern. Then, for each of these values, we encrypt a δ-set and compare the associated multiset to the table of precomputed possibilities: as there are 26 byte parameters that determine the middle rounds, the precomputed table has size 2^{210}. This can be reduced by a factor 2^7, if we replay the attack 2^7 times (increasing the data and time complexity).

New attacks with reduced memory. By reordering the steps, we obtain a new attack based on this previous one that still needs 2^{113} data, 2^{210} in time and now only 2^{194} in memory. We can propose a trade-off with different factors as they did, by considering a factor of 2^x less states to try in the middle if we store 2^x times more possible pairs. For this we need a data complexity increased by a factor $2^{x/2}$ (that generates 2^x times more pairs), and a time complexity that will be the max between 2^{210-x} and 2^{194-x}. All in all, we are able to propose better trade offs: indeed, in order to reach a memory of 2^{194} with the attack from [DFJ13], they would need a time complexity of 2^{212} and a data complexity of $2^{124.5}$.

11.5.6.2 New trade-offs on 7-rounds AES-128

In this case, the new trade-offs are not always interesting, in particular when compared to the best impossible differential attacks, but they improve upon previous DS-MITM attacks at least with respect to memory needs. The same way as before, we consider the DS-MITM 7-round AES-128 attack ([DFJ13, Fig.4]). When applying our improvements, we obtain for instance 2^{113} data, $2^{113} + 2^{84}$ time and 2^{74} memory. When considering the multiple differentials idea, we are able to reach 2^{105} data, $2^{105} + 2^{95}$ time and 2^{81} memory.

11.6 Conclusion

This chapter has presented the firsts quantum attacks on round-reduced AES, in the single-key setting. They are summarized in Table 11.3. Overall, the quantum attacks manage to break one less round than the classical attacks, for each version. This is only the first work of quantum cryptanalysis on AES, but it suggests that, at least for classically-inspired quantum attacks, it is *harder* for a quantum algorithm to beat the quantum search cost than for a classical algorithm to beat the classical search cost. This is mostly due to the fact that the quantization of a search algorithm reduces only the search cost, which makes for less favourable time/memory tradeoffs. Moreover, contrary to the situation in Chapter 7, having access to quantum queries does not improve that much the attacks. This still leaves some room for new quantum attacks without any classical equivalent, but to date, none of the approaches that are known to be much more efficient quantumly (such as Simon's algorithm) have been applied successfully to AES. On the contrary, some classical attacks, such as the impossible differentials, appear to be much less competitive when translated quantumly.

If we consider long-term security, this work allows to show that the best known attack on AES-256 only threatens 8 rounds, which leaves a large security margin. This suggests that AES is a strong primitive even against a quantum computer, and that AES-256 can be safely used for *very* long-term applications.

Table 11.3: Summary of the quantum cryptanalyses on AES we proposed, with the reference quantum search cost. Time is given in amount of computed S-Box.

Version	Rounds	Data	Time	C. mem.	Q. mem.	Model	Technique
128	6	2	2^{73}	None	small	Q1	Ex. search
	6	2^{48}	2^{50}	None	small	Q2	Square
	6	2^{32}	2^{50}	None	2^{32}	Q1	Square
	6	2^{32}	2^{71}	2^{32}	small	Q1	Square
	6	2^{35}	2^{58}	2^{35}	small	Q1	Improved square
	6	2^{35}	2^{44}	2^{35}	2^{27}	Q1	Partial sum square
192	7	2	$2^{105.1}$	None	small	Q1	Ex. search
	7	2^{37}	2^{103}	2^{37}	2^{27}	Q1	Partial sum square
256	7	3	$2^{137.7}$	None	small	Q1	Ex. search
	7	2^{37}	2^{106}	2^{37}	2^{27}	Q1	Partial sum square
	8	3	$2^{137.9}$	None	small	Q1	Ex. search
	8	2^{124}	$2^{134.7}$	2^{48}	small	Q2	DS MITM
	8	2^{113}	$2^{134.7}$	2^{88}	small	Q1	DS MITM

Conclusions

The aim of this thesis was to provide some new tools for the quantum cryptanalysis of symmetric primitives, and to apply them to existing designs. My work turned out to be mainly focused on cryptanalysis based on the hidden period and the hidden shift problems, as the quantum algorithms that solve them are superpolynomially faster than the best classical algorithms. The study of these quantum algorithms has allowed me to improve hidden shift algorithms and to propose some new cost tradeoffs, in Chapter 5, and to extend their scope of application, in Chapter 6. These algorithms allowed many quantum cryptanalyses of symmetric schemes, presented in Chapters 7, 8 and 9, with quantum queries, but also with classical queries and quantum computation. Rewriting some classical attacks in the formalism of hidden periods also allowed me to generalize the classical attack of Chaigneau and Gilbert on AEZv4 to AEZv5 and AEZ10, and to propose a classical attack on MiMC-$2n/n$ and multiple instances of GMiMC. Moreover, hidden shift algorithms can be applied in different contexts, and I studied their application to attack some isogeny-based schemes, in Chapter 10. Finally, I studied some cryptanalysis in cases where we do not have a structure that permits to use these algorithms. A framework to write search algorithms is proposed in Chapter 3, and is applied in the cryptanalyses of reduced-round AES of Chapter 11.

This last work is the first step towards the understanding of AES's security margin in a world where we only consider attacks better than the best generic attack (quantum search), and do not separate between classical and quantum attacks anymore. The quantum algorithms of Part I are tools for cryptanalysis, and we can also hope that they will contribute to the design of safe primitives, as the most efficient quantum algorithms require some specific structure.

There is still a lot to discover on hidden shift algorithms. In particular, only the asymptotic exponent is known for some variants. The problem is easier for parallel additions modulo a power of 2, and it would be interesting to know if something similar happens for a different moduli, and, in particular, if the problem is as hard modulo a smooth integer as modulo a prime number.

Quantum cryptanalysis is a fairly recent and rapidly-changing field, and we can expect that many improvements will be discovered, For instance, we could find new generic algorithms, that could either manage to make some classical ideas work in a quantum setting or use some new cryptanalysis techniques that could not be applied classically, as for many constructions, only the classical attacks have been studied, leading to a quantum security merely estimated from the classical one. I plan to explore these directions in the next years, studying some other families of quantum algorithms, and looking for new applications, in particular in public-key cryptography.

Bibliography

[Aag+19] Mark Aagaard, Riham AlTawy, Guang Gong, Kalikinkar Mandal, and Raghvendra Rohit. *ACE: An Authenticated Encryption and Hash Algorithm*. NIST lightweight competition round 1 candidate. Mar. 2019 (cit. on p. 114).

[AES] "Advanced Encryption Standard (AES)". In: *National Institute of Standards and Technology (NIST), FIPS PUB 197, U.S. Department of Commerce* (Nov. 2001) (cit. on pp. 17, 103, 165, 166).

[Alb+14] Martin R. Albrecht, Benedikt Driessen, Elif Bilge Kavun, Gregor Leander, Christof Paar, and Tolga Yalçin. "Block Ciphers - Focus on the Linear Layer (feat. PRIDE)". In: *CRYPTO 2014, Part I*. Ed. by Juan A. Garay and Rosario Gennaro. Vol. 8616. LNCS. Springer, Heidelberg, Aug. 2014, pp. 57–76 (cit. on p. 109).

[Alb+16] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. "MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity". In: *ASIACRYPT 2016, Part I*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. LNCS. Springer, Heidelberg, Dec. 2016, pp. 191–219 (cit. on pp. 148, 149).

[Alb+19] Martin R. Albrecht, Lorenzo Grassi, Léo Perrin, Sebastian Ramacher, Christian Rechberger, Dragos Rotaru, Arnab Roy, and Markus Schofnegger. "Feistel Structures for MPC, and More". In: *ESORICS 2019*. 2019 (cit. on pp. 148, 149).

[Alm+18] Mishal Almazrooie, Azman Samsudin, Rosni Abdullah, and Kussay N. Mutter. "Quantum reversible circuit of AES-128". In: *Quantum Information Processing* 17.5 (2018), p. 112 (cit. on pp. 170–172).

[And+16] Elena Andreeva, Andrey Bogdanov, Nilanjan Datta, Atul Luykx, Bart Mennink, Mridul Nandi, Elmar Tischhauser, and Kan Yasuda. *COLM v1*. CAESAR competition round 3 candidate. Sept. 2016 (cit. on p. 18).

[AR17] Gorjan Alagic and Alexander Russell. "Quantum-Secure Symmetric-Key Cryptography Based on Hidden Shifts". In: *EUROCRYPT 2017, Part III*. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10212. LNCS. Springer, Heidelberg, Apr. 2017, pp. 65–93 (cit. on pp. 116, 132).

[AS04] Scott Aaronson and Yaoyun Shi. "Quantum lower bounds for the collision and the element distinctness problems". In: *J. ACM* 51.4 (2004), pp. 595–605 (cit. on p. 45).

[BA08] Behnam Bahrak and Mohammad Reza Aref. "Impossible differential attack on seven-round AES-128". In: *IET Information Security* 2.2 (2008), pp. 28–32 (cit. on p. 165).

[Bao+19] Zhenzhen Bao, Avik Chakraborti, Nilanjan Datta, Jian Guo, Mridul Nandi, Thomas Peyrin, and Kan Yasuda. ***PHOTON-Beetle*** *Authenticated Encryption and Hash Family*. NIST lightweight competition round 1 candidate. Mar. 2019 (cit. on p. 115).

[Bar+18] Achiya Bar-On, Eli Biham, Orr Dunkelman, and Nathan Keller. "Efficient Slide Attacks". In: *Journal of Cryptology* 31.3 (July 2018), pp. 641–670 (cit. on pp. 129, 145, 146).

[BB84] C. H. Bennett and G. Brassard. "Quantum cryptography: Public key distribution and coin tossing". In: *Proceedings of IEEE International Conference on Computers, Systems, and Signal Processing*. Bangalore,India, 1984, p. 175 (cit. on p. 23).

[BCD06] Dave Bacon, Andrew M. Childs, and Wim van Dam. "Optimal measurements for the dihedral hidden subgroup problem". In: *Chicago J. Theor. Comput. Sci.* 2006 (2006) (cit. on p. 67).

[BCJ11] Anja Becker, Jean-Sébastien Coron, and Antoine Joux. "Improved Generic Algorithms for Hard Knapsacks". In: *EUROCRYPT 2011*. Ed. by Kenneth G. Paterson. Vol. 6632. LNCS. Springer, Heidelberg, May 2011, pp. 364–385 (cit. on pp. 58–60, 74).

[BCL08] Reinier Bröker, Denis Charles, and Kristin Lauter. "Evaluating Large Degree Isogenies and Applications to Pairing Based Cryptography". In: *PAIRING 2008*. Ed. by Steven D. Galbraith and Kenneth G. Paterson. Vol. 5209. LNCS. Springer, Heidelberg, Sept. 2008, pp. 100–112 (cit. on p. 158).

[BDK07] Eli Biham, Orr Dunkelman, and Nathan Keller. "Improved Slide Attacks". In: *FSE 2007*. Ed. by Alex Biryukov. Vol. 4593. LNCS. Springer, Heidelberg, Mar. 2007, pp. 153–166 (cit. on p. 129).

[BDK08] Eli Biham, Orr Dunkelman, and Nathan Keller. "A Unified Approach to Related-Key Attacks". In: *FSE 2008*. Ed. by Kaisa Nyberg. Vol. 5086. LNCS. Springer, Heidelberg, Feb. 2008, pp. 73–96 (cit. on p. 106).

[Bei+19] Christof Beierle, Alex Biryukov, Luan Cardoso dos Santos, Johann Großschädl, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, and Qingju Wang. SCHWA-EMM *and* ESCH*: Lightweight Authenticated Encryption and Hashing using the* SPARKLE *Permutation Family*. NIST lightweight competition round 1 candidate. Mar. 2019 (cit. on p. 115).

[Ben+97] Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh V. Vazirani. "Strengths and Weaknesses of Quantum Computing". In: *SIAM J. Comput.* 26.5 (1997), pp. 1510–1523 (cit. on p. 38).

[Ben89] Charles H. Bennett. "Time/Space Trade-Offs for Reversible Computation". In: *SIAM J. Comput.* 18.4 (1989), pp. 766–776 (cit. on p. 30).

[Ber+07] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. "Sponge functions". In: *ECRYPT hash workshop* (2007) (cit. on p. 19).

[Ber+08] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "On the Indifferentiability of the Sponge Construction". In: *EUROCRYPT 2008*. Ed. by Nigel P. Smart. Vol. 4965. LNCS. Springer, Heidelberg, Apr. 2008, pp. 181–197 (cit. on p. 114).

[Ber+12a] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "Permutation Based Encryption, Authentication and Authenticated Encryption". In: *Workshop Records of DIAC 2012*. 2012, pp. 159–170 (cit. on p. 114).

[Ber+12b] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications". In: *SAC 2011*. Ed. by Ali Miri and Serge Vaudenay. Vol. 7118. LNCS. Springer, Heidelberg, Aug. 2012, pp. 320–337 (cit. on p. 114).

[Ber+13] Daniel J. Bernstein, Stacey Jeffery, Tanja Lange, and Alexander Meurer. "Quantum Algorithms for the Subset-Sum Problem". In: *Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013*. Ed. by Philippe Gaborit. Springer, Heidelberg, June 2013, pp. 16–33 (cit. on pp. 58, 59).

[Ber+17] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. "Farfalle: parallel permutation-based cryptography". In: *IACR Trans. Symm. Cryptol.* 2017.4 (2017), pp. 1–38. ISSN: 2519-173X (cit. on pp. 100, 108).

[Ber+19a] Daniel J. Bernstein, Stefan Kölbl, Stefan Lucks, Pedro Maat Costa Massolino, Florian Mendel, Kashif Nawaz, Tobias Schneider, Peter Schwabe, François-Xavier Standaert, Yosuke Todo, and Benoît Viguier. *Gimli*. NIST lightweight competition round 1 candidate. Mar. 2019 (cit. on p. 114).

[Ber+19b] Daniel J. Bernstein, Tanja Lange, Chloe Martindale, and Lorenz Panny. "Quantum Circuits for the CSIDH: Optimizing Quantum Evaluation of Isogenies". In: *EUROCRYPT 2019, Part II*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11477. LNCS. Springer, Heidelberg, May 2019, pp. 409–441 (cit. on pp. 160, 161).

[Ber05] Daniel J. Bernstein. "The Poly1305-AES Message-Authentication Code". In: *FSE 2005*. Ed. by Henri Gilbert and Helena Handschuh. Vol. 3557. LNCS. Springer, Heidelberg, Feb. 2005, pp. 32–49 (cit. on pp. 18, 111).

[BH97] Gilles Brassard and Peter Høyer. "An Exact Quantum Polynomial-Time Algorithm for Simon's Problem". In: *Fifth Israel Symposium on Theory of Computing and Systems, ISTCS 1997, Ramat-Gan, Israel, June 17-19, 1997, Proceedings*. IEEE Computer Society, 1997, pp. 12–23. ISBN: 0-8186-8037-7 (cit. on p. 52).

[BHT98] Gilles Brassard, Peter Høyer, and Alain Tapp. "Quantum Cryptanalysis of Hash and Claw-Free Functions". In: *LATIN '98: Theoretical Informatics, Third Latin American Symposium, Campinas, Brazil, April, 20-24, 1998, Proceedings*. Ed. by Claudio L. Lucchesi and Arnaldo V. Moura. Vol. 1380. Springer, Heidelberg, 1998, pp. 163–169 (cit. on p. 45).

[Bia+19] Jean-François Biasse, Xavier Bonnetain, Benjamin Pring, André Schrottenloher, and William Youmans. "Trade-off between classical and quantum circuit size of the attack against CSIDH". In: *J. Mathematical Cryptology* (2019) (cit. on pp. 8, 10, 153).

[Bih94] Eli Biham. "New Types of Cryptanalytic Attacks Using related Keys (Extended Abstract)". In: *EUROCRYPT'93*. Ed. by Tor Helleseth. Vol. 765. LNCS. Springer, Heidelberg, May 1994, pp. 398–409 (cit. on p. 106).

[BIJ18] Jean-François Biasse, Annamaria Iezzi, and Michael J. Jacobson Jr. "A Note on the Security of CSIDH". In: *INDOCRYPT 2018*. Ed. by Debrup Chakraborty and Tetsu Iwata. Vol. 11356. LNCS. Springer, Heidelberg, Dec. 2018, pp. 153–168 (cit. on p. 159).

[BK09] Alex Biryukov and Dmitry Khovratovich. "Related-Key Cryptanalysis of the Full AES-192 and AES-256". In: *ASIACRYPT 2009*. Ed. by Mitsuru Matsui. Vol. 5912. LNCS. Springer, Heidelberg, Dec. 2009, pp. 1–18 (cit. on p. 165).

[BKN09] Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolic. "Distinguisher and Related-Key Attack on the Full AES-256". In: *CRYPTO 2009*. Ed. by Shai Halevi. Vol. 5677. LNCS. Springer, Heidelberg, Aug. 2009, pp. 231–249 (cit. on p. 165).

[BKR00] Mihir Bellare, Joe Kilian, and Phillip Rogaway. "The Security of the Cipher Block Chaining Message Authentication Code". In: *Journal of Computer and System Sciences* 61.3 (2000), pp. 362–399 (cit. on p. 110).

[BKR11] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. "Biclique Cryptanalysis of the Full AES". In: *ASIACRYPT 2011*. Ed. by Dong Hoon Lee and Xiaoyun Wang. Vol. 7073. LNCS. Springer, Heidelberg, Dec. 2011, pp. 344–371 (cit. on p. 170).

[BKV19] Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. "CSI-FiSh: Efficient Isogeny based Signatures through Class Group Computations". In: *IACR Cryptology ePrint Archive* 2019 (2019), p. 498 (cit. on p. 161).

[BL17] Daniel J. Bernstein and Tanja Lange. "Post-quantum cryptography". In: *Nature* 549.7671 (Sept. 2017), pp. 188–194. ISSN: 0028-0836 (cit. on pp. 100, 111).

[BN18] Xavier Bonnetain and María Naya-Plasencia. "Hidden Shift Quantum Cryptanalysis and Implications". In: *ASIACRYPT 2018, Part I*. Ed. by Thomas Peyrin and Steven Galbraith. Vol. 11272. LNCS. Springer, Heidelberg, Dec. 2018, pp. 560–592 (cit. on pp. 8, 9, 57, 77–79, 81, 82, 99, 110, 112).

[BNS19a] Xavier Bonnetain, María Naya-Plasencia, and André Schrottenloher. "On Quantum Slide Attacks". In: *SAC 2019*. Ed. by Kenneth G. Paterson and Douglas Stebila: LNCS. Springer, Heidelberg, Aug. 2019 (cit. on pp. 9, 99, 129).

[BNS19b] Xavier Bonnetain, María Naya-Plasencia, and André Schrottenloher. "Quantum Security Analysis of AES". In: *IACR Trans. Symm. Cryptol.* 2019.2 (2019), pp. 55–93. ISSN: 2519-173X (cit. on pp. 7, 10, 35, 40, 165).

[Bog+15] Andrey Bogdanov, Donghoon Chang, Mohona Ghosh, and Somitra Kumar Sanadhya. "Bicliques with Minimal Data and Time Complexity for AES". In: *ICISC 14*. Ed. by Jooyoung Lee and Jongsung Kim. Vol. 8949. LNCS. Springer, Heidelberg, Dec. 2015, pp. 160–174 (cit. on p. 170).

[Bon+17] Xavier Bonnetain, Patrick Derbez, Sébastien Duval, Jérémy Jean, Gaëtan Leurent, Brice Minaud, and Valentin Suder. "An easy attack on AEZ". In: *FSE 2017 rump session* (Mar. 2017) (cit. on p. 117).

[Bon+19] Xavier Bonnetain, Akinori Hosoyamada, María Naya-Plasencia, Yu Sasaki, and André Schrottenloher. "Quantum Attacks without Superposition Queries: the Offline Simon's Algorithm". In: *ASIACRYPT 2019*. Ed. by Steven Galbraith and Shiho Moriai. LNCS. Springer, Heidelberg, Dec. 2019 (cit. on pp. 9, 87, 90, 99).

[Bon17] Xavier Bonnetain. "Quantum Key-Recovery on Full AEZ". In: *SAC 2017*. Ed. by Carlisle Adams and Jan Camenisch. Vol. 10719. LNCS. Springer, Heidelberg, Aug. 2017, pp. 394–406 (cit. on pp. 9, 99, 117, 124).

[Bon19a] Xavier Bonnetain. *Collisions on Feistel-MiMC and univariate GMiMC*. 2019 (cit. on pp. 9, 129).

[Bon19b] Xavier Bonnetain. *Improved Low-qubit Hidden Shift Algorithms*. 2019. arXiv: 1901.11428 (cit. on pp. 8, 57, 70, 72).

[Bor+12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knežević, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. "PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract". In: *ASIACRYPT 2012*. Ed. by Xiaoyun Wang and Kazue Sako. Vol. 7658. LNCS. Springer, Heidelberg, Dec. 2012, pp. 208–225 (cit. on p. 109).

[Bou+18] Christina Boura, Virginie Lallemand, María Naya-Plasencia, and Valentin Suder. "Making the Impossible Possible". In: *Journal of Cryptology* 31.1 (Jan. 2018), pp. 101–133 (cit. on pp. 165, 170, 171).

[Boy+98] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. "Tight Bounds on Quantum Searching". In: *Fortschritte der Physik* 46.4-5 (1998), pp. 493–505 (cit. on p. 38).

[BPT19] Xavier Bonnetain, Léo Perrin, and Shizhu Tian. "Anomalies and Vector Space Search: Tools for S-Box Analysis". In: *ASIACRYPT 2019*. Ed. by Steven Galbraith and Shiho Moriai. LNCS. Springer, Heidelberg, Dec. 2019 (cit. on p. 10).

[BR00] John Black and Phillip Rogaway. "CBC MACs for Arbitrary-Length Messages: The Three-Key Constructions". In: *CRYPTO 2000*. Ed. by Mihir Bellare. Vol. 1880. LNCS. Springer, Heidelberg, Aug. 2000, pp. 197–215 (cit. on p. 110).

[Bra+02] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. "Quantum Amplitude Amplification and Estimation". In: *Quantum Computation and Information, AMS Contemporary Mathematics 305*. Ed. by Samuel J. Lomo-naco and Howard E. Brandt. 2002 (cit. on pp. 35, 38).

[BS18] Xavier Bonnetain and André Schrottenloher. *Quantum Security Analysis of CSIDH and Ordinary Isogeny-based Schemes*. 2018 (cit. on pp. 8, 10, 57, 77, 80, 153, 159).

[BW00] Alex Biryukov and David Wagner. "Advanced Slide Attacks". In: *EUROCRYPT 2000*. Ed. by Bart Preneel. Vol. 1807. LNCS. Springer, Heidelberg, May 2000, pp. 589–606 (cit. on pp. 108, 129, 131, 134, 136, 141).

[BW99] Alex Biryukov and David Wagner. "Slide Attacks". In: *FSE'99*. Ed. by Lars R. Knudsen. Vol. 1636. LNCS. Springer, Heidelberg, Mar. 1999, pp. 245–259 (cit. on pp. 129, 131).

[Can+19] Anne Canteaut, Sébastien Duval, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, Thomas Pornin, and André Schrottenloher. SATURNIN*: a suite of lightweight symmetric algorithms for post-quantum security*. NIST lightweight competition round 1 candidate. Mar. 2019 (cit. on pp. 100, 116).

[Cas+18] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. "CSIDH: An Efficient Post-Quantum Commutative Group Action". In: *ASIACRYPT 2018, Part III*. Ed. by Thomas Peyrin and Steven Galbraith. Vol. 11274. LNCS. Springer, Heidelberg, Dec. 2018, pp. 395–427 (cit. on pp. 153, 154, 157, 161, 162).

[CG16] Colin Chaigneau and Henri Gilbert. "Is AEZ v4.1 Sufficiently Resilient Against Key-Recovery Attacks?" In: *IACR Trans. Symm. Cryptol.* 2016.1 (2016). http://tosc.iacr.org/index.php/ToSC/article/view/538, pp. 114–133. ISSN: 2519-173X (cit. on pp. 117, 118, 122, 124, 126, 127).

[Cha+18] Avik Chakraborti, Nilanjan Datta, Mridul Nandi, and Kan Yasuda. "Beetle Family of Lightweight and Secure Authenticated Encryption Ciphers". In: *IACR TCHES* 2018.2 (2018). https://tches.iacr.org/index.php/TCHES/article/view/881, pp. 218–241. ISSN: 2569-2925 (cit. on p. 115).

[Che+08] Donny Cheung, Dmitri Maslov, Jimson Mathew, and Dhiraj K. Pradhan. "Theory of Quantum Computation, Communication, and Cryptography". In: ed. by Yasuhito Kawano and Michele Mosca. Berlin, Heidelberg: Springer-Verlag, 2008. Chap. On the Design and Optimization of a Quantum Polynomial-Time Attack on Elliptic Curve Cryptography, pp. 96–104. ISBN: 978-3-540-89303-5 (cit. on p. 180).

[CHS19] Jan Czajkowski, Andreas Hülsing, and Christian Schaffner. "Quantum Indistinguishability of Random Sponges". In: *IACR Cryptology ePrint Archive* 2019 (2019), p. 69 (cit. on p. 114).

[CJS14] Andrew M. Childs, David Jao, and Vladimir Soukharev. "Constructing elliptic curve isogenies in quantum subexponential time". In: *J. Mathematical Cryptology* 8.1 (2014), pp. 1–29 (cit. on pp. 57, 70–72, 74, 153, 157).

[CNS17] André Chailloux, María Naya-Plasencia, and André Schrottenloher. "An Efficient Quantum Collision Search Algorithm and Implications on Symmetric Cryptography". In: *ASIACRYPT 2017, Part II*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10625. LNCS. Springer, Heidelberg, Dec. 2017, pp. 211–240 (cit. on p. 46).

[Cou06] Jean-Marc Couveignes. *Hard Homogeneous Spaces*. Cryptology ePrint Archive, Report 2006/291. http://eprint.iacr.org/2006/291. 2006 (cit. on pp. 153, 154, 158).

[Dae+18] Joan Daemen, Seth Hoffert, Gilles Van Assche, and Ronny Van Keer. "The design of Xoodoo and Xoofff". In: *IACR Trans. Symm. Cryptol.* 2018.4 (2018), pp. 1–38. ISSN: 2519-173X (cit. on pp. 100, 108).

[Dae+19] Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. *Xoodyak, a lightweight cryptographic scheme*. NIST lightweight competition round 1 candidate. Mar. 2019 (cit. on p. 114).

[Dae93] Joan Daemen. "Limitations of the Even-Mansour Construction (Rump Session)". In: *ASIACRYPT'91*. Ed. by Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto. Vol. 739. LNCS. Springer, Heidelberg, Nov. 1993, pp. 495–498 (cit. on p. 108).

[DDW18] Xiaoyang Dong, Bingyou Dong, and Xiaoyun Wang. *Quantum Attacks on Some Feistel Block Ciphers*. Cryptology ePrint Archive, Report 2018/504. https://eprint.iacr.org/2018/504. 2018 (cit. on pp. 129, 141, 152).

[DES] "Data Encryption Standard". In: *National Bureau of Standards, NBS FIPS PUB 46, U.S. Department of Commerce* (Jan. 1977) (cit. on pp. 17, 103, 165).

[DFJ13] Patrick Derbez, Pierre-Alain Fouque, and Jérémy Jean. "Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting". In: *EUROCRYPT 2013*. Ed. by Thomas Johansson and Phong Q. Nguyen. Vol. 7881. LNCS. Springer, Heidelberg, May 2013, pp. 371–387 (cit. on pp. 165, 170, 171, 179, 182, 183, 187, 194, 195).

[DG19] Luca De Feo and Steven D. Galbraith. "SeaSign: Compact Isogeny Signatures from Class Group Actions". In: *EUROCRYPT 2019, Part III*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11478. LNCS. Springer, Heidelberg, May 2019, pp. 759–789 (cit. on p. 153).

[DH76] Whitfield Diffie and Martin E. Hellman. "New Directions in Cryptography". In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654 (cit. on p. 15).

[Din+15] Itai Dinur, Orr Dunkelman, Nathan Keller, and Adi Shamir. "Reflections on slide with a twist attacks". In: *Des. Codes Cryptography* 77.2-3 (2015), pp. 633–651 (cit. on pp. 129, 144).

[Din18] Itai Dinur. *An Algorithmic Framework for the Generalized Birthday Problem*. Cryptology ePrint Archive, Report 2018/575. https://eprint.iacr.org/2018/575. 2018 (cit. on p. 61).

[DKR97] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. "The Block Cipher Square". In: *FSE'97*. Ed. by Eli Biham. Vol. 1267. LNCS. Springer, Heidelberg, Jan. 1997, pp. 149–165 (cit. on pp. 165, 169, 172, 174).

[DKS10] Orr Dunkelman, Nathan Keller, and Adi Shamir. "Improved Single-Key Attacks on 8-Round AES-192 and AES-256". In: *ASIACRYPT 2010*. Ed. by Masayuki Abe. Vol. 6477. LNCS. Springer, Heidelberg, Dec. 2010, pp. 158–176 (cit. on pp. 165, 168, 170, 179, 184).

[DKS12] Orr Dunkelman, Nathan Keller, and Adi Shamir. "Minimalism in Cryptography: The Even-Mansour Scheme Revisited". In: *EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. LNCS. Springer, Heidelberg, Apr. 2012, pp. 336–354 (cit. on pp. 107, 108).

[DKS15] Orr Dunkelman, Nathan Keller, and Adi Shamir. "Slidex Attacks on the Even-Mansour Encryption Scheme". In: *Journal of Cryptology* 28.1 (Jan. 2015), pp. 1–28 (cit. on pp. 129, 143).

[DKS18] Luca De Feo, Jean Kieffer, and Benjamin Smith. "Towards Practical Key Exchange from Ordinary Isogeny Graphs". In: *ASIACRYPT 2018, Part III*. Ed. by Thomas Peyrin and Steven Galbraith. Vol. 11274. LNCS. Springer, Heidelberg, Dec. 2018, pp. 365–394 (cit. on pp. 153, 154, 157).

[DLW19] Xiaoyang Dong, Zheng Li, and Xiaoyun Wang. "Quantum cryptanalysis on some generalized Feistel schemes". In: *SCIENCE CHINA Information Sciences* 62.2 (2019), 22501:1–22501:12 (cit. on pp. 99, 105).

[Dob+19] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. *Ascon v1.2*. NIST lightweight competition round 1 candidate. Mar. 2019 (cit. on pp. 18, 114).

[DPV19] Thomas Decru, Lorenz Panny, and Frederik Vercauteren. "Faster SeaSign Signatures Through Improved Rejection Sampling". In: *Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019*. Ed. by Jintai Ding and Rainer Steinwandt. Springer, Heidelberg, 2019, pp. 271–285 (cit. on p. 153).

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002. ISBN: 3-540-42580-2 (cit. on p. 119).

[DR07] Joan Daemen and Vincent Rijmen. "Probability distributions of correlation and differentials in block ciphers". In: *J. Mathematical Cryptology* 1.3 (2007), pp. 221–242 (cit. on p. 55).

[DS08] Hüseyin Demirci and Ali Aydin Selçuk. "A Meet-in-the-Middle Attack on 8-Round AES". In: *FSE 2008*. Ed. by Kaisa Nyberg. Vol. 5086. LNCS. Springer, Heidelberg, Feb. 2008, pp. 116–126 (cit. on pp. 165, 169, 170, 179, 184).

[EH99] Mark Ettinger and Peter Høyer. "On Quantum Algorithms for Noncommutative Hidden Subgroups". In: *STACS 99, 16th Annual Symposium on Theoretical Aspects of Computer Science, Trier, Germany, March 4-6, 1999, Proceedings*. Ed. by Christoph Meinel and Sophie Tison. Vol. 1563. Springer, 1999, pp. 478–487 (cit. on pp. 57, 66, 67).

[EHK04] Mark Ettinger, Peter Høyer, and Emanuel Knill. "The quantum query complexity of the hidden subgroup problem is polynomial". In: *Information Processing Letters* 91.1 (2004), pp. 43–48 (cit. on p. 47).

[EM97] Shimon Even and Yishay Mansour. "A Construction of a Cipher from a Single Pseudorandom Permutation". In: *Journal of Cryptology* 10.3 (June 1997), pp. 151–162 (cit. on p. 107).

[Fer+01] Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Michael Stay, David Wagner, and Doug Whiting. "Improved Cryptanalysis of Rijndael". In: *FSE 2000*. Ed. by Bruce Schneier. Vol. 1978. LNCS. Springer, Heidelberg, Apr. 2001, pp. 213–230 (cit. on pp. 165, 169, 171, 172, 175, 176).

[Fey82] R. P. Feynman. "Simulating Physics with Computers". In: *International Journal of Theoretical Physics* 21 (June 1982), pp. 467–488 (cit. on p. 23).

[FJP13] Pierre-Alain Fouque, Jérémy Jean, and Thomas Peyrin. "Structural Evaluation of AES and Chosen-Key Distinguisher of 9-Round AES-128". In: *CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. LNCS. Springer, Heidelberg, Aug. 2013, pp. 183–203 (cit. on p. 165).

[FLS15] Thomas Fuhr, Gaëtan Leurent, and Valentin Suder. "Collision Attacks Against CAESAR Candidates - Forgery and Key-Recovery Against AEZ and Marble". In: *ASIACRYPT 2015, Part II*. Ed. by Tetsu Iwata and Jung Hee Cheon. Vol. 9453. LNCS. Springer, Heidelberg, Nov. 2015, pp. 510–532 (cit. on pp. 118, 127).

[FO90] Philippe Flajolet and Andrew M. Odlyzko. "Random Mapping Statistics". In: *EUROCRYPT'89*. Ed. by Jean-Jacques Quisquater and Joos Vandewalle. Vol. 434. LNCS. Springer, Heidelberg, Apr. 1990, pp. 329–354 (cit. on pp. 83, 102).

[Gag17] Tommaso Gagliardoni. "Quantum Security of Cryptographic Primitives". PhD thesis. Darmstadt University of Technology, Germany, 2017 (cit. on p. 21).

[GCM07] Morris Dworkin. "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC". In: *National Institute of Standards and Technology (NIST), Special Publication 800-38D, U.S. Department of Commerce* (Nov. 2007) (cit. on p. 18).

[GHS02] Steven D. Galbraith, Florian Hess, and Nigel P. Smart. "Extending the GHS Weil Descent Attack". In: *EUROCRYPT 2002*. Ed. by Lars R. Knudsen. Vol. 2332. LNCS. Springer, Heidelberg, Apr. 2002, pp. 29–44 (cit. on p. 158).

[GM00] Henri Gilbert and Marine Minier. "A Collision Attack on 7 Rounds of Rijndael". In: *AES Candidate Conference*. 2000, pp. 230–241 (cit. on p. 165).

[GMO19] Danilo Gligoroski, Hristina Mihajloska, and Daniel Otte. *GAGE and InGAGE v1.0*. NIST lightweight competition round 1 candidate. Mar. 2019 (cit. on p. 114).

[GN08] Nicolas Gama and Phong Q. Nguyen. "Predicting Lattice Reduction". In: *EUROCRYPT 2008*. Ed. by Nigel P. Smart. Vol. 4965. LNCS. Springer, Heidelberg, Apr. 2008, pp. 31–51 (cit. on p. 160).

[Gra+16] Markus Grassl, Brandon Langenberg, Martin Roetteler, and Rainer Steinwandt. "Applying Grover's Algorithm to AES: Quantum Resource Estimates". In: *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016*. Ed. by Tsuyoshi Takagi. Springer, Heidelberg, 2016, pp. 29–43 (cit. on pp. 165, 170–172, 179, 180).

[Gro96] Lov K. Grover. "A Fast Quantum Mechanical Algorithm for Database Search". In: *28th ACM STOC*. ACM Press, May 1996, pp. 212–219 (cit. on p. 35).

[HA17] Akinori Hosoyamada and Kazumaro Aoki. "On Quantum Related-Key Attacks on Iterated Even-Mansour Ciphers". In: *IWSEC 17*. Ed. by Satoshi Obana and Koji Chida. Vol. 10418. LNCS. Springer, Heidelberg, Aug. 2017, pp. 3–18 (cit. on pp. 129, 131–133, 152).

[HI19] Akinori Hosoyamada and Tetsu Iwata. "4-Round Luby-Rackoff Construction is a qPRP". In: *ASIACRYPT 2019*. Ed. by Steven Galbraith and Shiho Moriai. LNCS. Springer, Heidelberg, Dec. 2019 (cit. on p. 105).

[HKR15] Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway. "Robust Authenticated-Encryption AEZ and the Problem That It Solves". In: *EUROCRYPT 2015, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. LNCS. Springer, Heidelberg, Apr. 2015, pp. 15–44 (cit. on pp. 18, 110, 117, 118).

[HM18] Alexander Helm and Alexander May. "Subset Sum Quantumly in 1.17^n". In: *13th Conference on the Theory of Quantum Computation, Communication and Cryptography, TQC 2018, July 16-18, 2018, Sydney, Australia*. Ed. by Stacey Jeffery. Vol. 111. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018, 5:1–5:15. ISBN: 978-3-95977-080-4 (cit. on p. 59).

[HM89] James L Hafner and Kevin S McCurley. "A rigorous subexponential algorithm for computation of class groups". In: *Journal of the American mathematical society* 2.4 (1989), pp. 837–850 (cit. on p. 158).

[Hoe63] Wassily Hoeffding. "Probability Inequalities for Sums of Bounded Random Variables". In: *Journal of the American Statistical Association* 58.301 (1963), pp. 13–30. ISSN: 01621459 (cit. on p. 67).

[HPS11] G. Hanrot, X. Pujol, and D. Stehlé. "Terminating BKZ". In: *IACR Cryptology ePrint Archive* 2011 (2011), p. 198 (cit. on p. 159).

[HS18] Akinori Hosoyamada and Yu Sasaki. "Quantum Demiric-Selçuk Meet-in-the-Middle Attacks: Applications to 6-Round Generic Feistel Constructions". In: *SCN 18*. Ed. by Dario Catalano and Roberto De Prisco. Vol. 11035. LNCS. Springer, Heidelberg, Sept. 2018, pp. 386–403 (cit. on p. 30).

[HY18] Akinori Hosoyamada and Kan Yasuda. "Building Quantum-One-Way Functions from Block Ciphers: Davies-Meyer and Merkle-Damgård Constructions". In: *ASIACRYPT 2018, Part I*. Ed. by Thomas Peyrin and Steven Galbraith. Vol. 11272. LNCS. Springer, Heidelberg, Dec. 2018, pp. 275–304 (cit. on p. 101).

[II19] Gembu Ito and Tetsu Iwata. "Quantum Distinguishing Attacks against Type-1 Generalized Feistel Ciphers". In: *IACR Cryptology ePrint Archive* 2019 (2019), p. 327 (cit. on pp. 99, 105).

[IK03] Tetsu Iwata and Kaoru Kurosawa. "OMAC: One-Key CBC MAC". In: *FSE 2003*. Ed. by Thomas Johansson. Vol. 2887. LNCS. Springer, Heidelberg, Feb. 2003, pp. 129–153 (cit. on p. 110).

[Ito+19] Gembu Ito, Akinori Hosoyamada, Ryutaroh Matsumoto, Yu Sasaki, and Tetsu Iwata. "Quantum Chosen-Ciphertext Attacks Against Feistel Ciphers". In: *CT-RSA 2019*. Ed. by Mitsuru Matsui. Vol. 11405. LNCS. Springer, Heidelberg, Mar. 2019, pp. 391–411 (cit. on pp. 99, 105).

[Jal+19] Amir Jalali, Reza Azarderakhsh, Mehran Mozaffari Kermani, and David Jao. "Towards Optimized and Constant-Time CSIDH on Embedded Devices". In: *COSADE*. Vol. 11421. Lecture Notes in Computer Science. Springer, 2019, pp. 215–231 (cit. on p. 153).

[JD11] David Jao and Luca De Feo. “Towards Quantum-Resistant Cryptosystems from Supersingular Elliptic Curve Isogenies”. In: *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*. Ed. by Bo-Yin Yang. Springer, Heidelberg, Nov. 2011, pp. 19–34 (cit. on p. 153).

[Jea+16] Jérémy Jean, Ivica Nikolić, Thomas Peyrin, and Yannick Seurin. *Deoxys v1.41*. CAESAR competition round 3 candidate. Sept. 2016 (cit. on p. 18).

[Jea16] Jérémy Jean. *TikZ for Cryptographers*. https://www.iacr.org/authors/tikz/. 2016 (cit. on pp. 121, 166).

[Jou09] Antoine Joux. *Algorithmic Cryptanalysis*. CRC press, 2009. ISBN: 978-1-4200-7002-6 (cit. on p. 44).

[Kan83] R. Kannan. “Improved Algorithms for Integer Programming and Related Lattice Problems”. In: *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*. 1983, pp. 193–206 (cit. on p. 159).

[Kap+16] Marc Kaplan, Gaëtan Leurent, Anthony Leverrier, and María Naya-Plasencia. “Breaking Symmetric Cryptosystems Using Quantum Period Finding”. In: *CRYPTO 2016, Part II*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9815. LNCS. Springer, Heidelberg, Aug. 2016, pp. 207–237 (cit. on pp. 49, 53, 54, 99, 109, 110, 117, 124, 127, 129, 131, 132, 145, 152).

[Kas63] Friedrich Wilhelm Kasiski. *Die Geheimschriften und die Dechiffrir-Kunst*. Berlin: E. S. Mittler und Sohn, 1863 (cit. on p. 14).

[Ker83] Auguste Kerckhoffs. “La cryptographie militaire”. In: *Journal des sciences militaires* 9 (Jan. 1883), pp. 5–38 (cit. on p. 14).

[Kin09] أبو يوسف يعقوب بن إسحاق الصبّاح الكندي. “رسالة في استخراج الكتب المعماة”. IXth century (cit. on p. 14).

[Kit96] Alexei Y. Kitaev. “Quantum measurements and the Abelian Stabilizer Problem”. In: *Electronic Colloquium on Computational Complexity (ECCC)* 3.3 (1996) (cit. on p. 30).

[KM10] Hidenori Kuwakado and Masakatu Morii. “Quantum distinguisher between the 3-round Feistel cipher and the random permutation”. In: *IEEE International Symposium on Information Theory, ISIT 2010, June 13-18, 2010, Austin, Texas, USA, Proceedings*. IEEE, 2010, pp. 2682–2685 (cit. on pp. 99, 104).

[KM12] Hidenori Kuwakado and Masakatu Morii. “Security on the quantum-type Even-Mansour cipher”. In: *Proceedings of the International Symposium on Information Theory and its Applications, ISITA 2012, Honolulu, HI, USA, October 28-31, 2012*. IEEE, 2012, pp. 312–316. ISBN: 978-1-4673-2521-9 (cit. on pp. 99, 108).

[KR16] Ted Krovetz and Philip Rogaway. *OCB v1.1*. CAESAR competition round 3 candidate. 2016 (cit. on p. 18).

[KR96] Joe Kilian and Phillip Rogaway. “How to Protect DES Against Exhaustive Key Search”. In: *CRYPTO’96*. Ed. by Neal Koblitz. Vol. 1109. LNCS. Springer, Heidelberg, Aug. 1996, pp. 252–267 (cit. on p. 108).

[KRS12] Dmitry Khovratovich, Christian Rechberger, and Alexandra Savelieva. “Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 Family”. In: *FSE 2012*. Ed. by Anne Canteaut. Vol. 7549. LNCS. Springer, Heidelberg, Mar. 2012, pp. 244–263 (cit. on p. 170).

[Kup05] Greg Kuperberg. "A Subexponential-Time Quantum Algorithm for the Dihedral Hidden Subgroup Problem". In: *SIAM Journal on Computing* 35.1 (2005), pp. 170–188 (cit. on pp. 57, 68, 69, 78, 80).

[Kup13] Greg Kuperberg. "Another Subexponential-time Quantum Algorithm for the Dihedral Hidden Subgroup Problem". In: *8th Conference on the Theory of Quantum Computation, Communication and Cryptography*. Ed. by Simone Severini and Fernando G. S. L. Brandão. Vol. 22. LIPIcs. Guelph, Canada: Schloss Dagstuhl, 2013, pp. 20–34 (cit. on pp. 57, 73, 75, 76).

[KW02] Lars R. Knudsen and David Wagner. "Integral Cryptanalysis". In: *FSE 2002*. Ed. by Joan Daemen and Vincent Rijmen. Vol. 2365. LNCS. Springer, Heidelberg, Feb. 2002, pp. 112–127 (cit. on p. 165).

[Lan+16] Adam Langley, Wan-Teh Chang, Nikos Mavrogiannopoulos, Joachim Strömbergson, and Simon Josefsson. "ChaCha20-Poly1305 Cipher Suites for Transport Layer Security (TLS)". In: *RFC* 7905 (2016), pp. 1–8 (cit. on p. 111).

[Lem10] Alexis Lemaire. "Application de l'hypercalculie et de l'informatique quantique gravifique à l'intelligence artificielle générale". PhD thesis. Université de Reims, 2010 (cit. on pp. 1–217).

[LM17] Gregor Leander and Alexander May. "Grover Meets Simon - Quantumly Attacking the FX-construction". In: *ASIACRYPT 2017, Part II*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10625. LNCS. Springer, Heidelberg, Dec. 2017, pp. 161–178 (cit. on pp. 87, 88, 99, 108).

[LR88] Michael Luby and Charles Rackoff. "How to construct pseudorandom permutations from pseudorandom functions". In: *SIAM Journal on Computing* 17.2 (1988) (cit. on p. 103).

[LRW02] Moses Liskov, Ronald L. Rivest, and David Wagner. "Tweakable Block Ciphers". In: *CRYPTO 2002*. Ed. by Moti Yung. Vol. 2442. LNCS. Springer, Heidelberg, Aug. 2002, pp. 31–46 (cit. on p. 110).

[LS18] Gaëtan Leurent and Ferdinand Sibleyras. "The Missing Difference Problem, and Its Applications to Counter Mode Encryption". In: *EUROCRYPT 2018, Part II*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10821. LNCS. Springer, Heidelberg, Apr. 2018, pp. 745–770 (cit. on p. 111).

[Lu+08] Jiqiang Lu, Orr Dunkelman, Nathan Keller, and Jongsung Kim. "New Impossible Differential Attacks on AES". In: *INDOCRYPT 2008*. Ed. by Dipanwita Roy Chowdhury, Vincent Rijmen, and Abhijit Das. Vol. 5365. LNCS. Springer, Heidelberg, Dec. 2008, pp. 279–293 (cit. on p. 165).

[Mal+10] Hamid Mala, Mohammad Dakhilalian, Vincent Rijmen, and Mahmoud Modarres-Hashemi. "Improved Impossible Differential Cryptanalysis of 7-Round AES-128". In: *INDOCRYPT 2010*. Ed. by Guang Gong and Kishan Chand Gupta. Vol. 6498. LNCS. Springer, Heidelberg, Dec. 2010, pp. 282–291 (cit. on p. 165).

[Mar10] Luther Martin. "XTS: A Mode of AES for Encrypting Hard Disks". In: *IEEE Security & Privacy* 8.3 (2010), pp. 68–69 (cit. on p. 110).

[MCR19] Michael Meyer, Fabio Campos, and Steffen Reith. "On Lions and Elligators: An Efficient Constant-Time Implementation of CSIDH". In: *Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019*. Ed. by Jintai Ding and Rainer Steinwandt. Springer, Heidelberg, 2019, pp. 307–325 (cit. on p. 153).

[MD79] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco: Freeman, 1979 (cit. on p. 58).

[ME98] Michele Mosca and Artur Ekert. "The Hidden Subgroup Problem and Eigenvalue Estimation on a Quantum Computer". In: *Quantum Computing and Quantum Communications, First NASA International Conference, QCQC'98, Palm Springs, California, USA, February 17-20, 1998, Selected Papers*. Ed. by Colin P. Williams. Vol. 1509. Lecture Notes in Computer Science. Springer, 1998, pp. 174–188. ISBN: 3-540-65514-X (cit. on pp. 65, 158).

[Mil82] Frank Miller. *Telegraphic Code to Insure Privacy and Secrecy in the Transmission of Telegrams*. New York: Charles M. Cornwell, 1882 (cit. on p. 14).

[Mou+14] Nicky Mouha, Bart Mennink, Anthony Van Herrewege, Dai Watanabe, Bart Preneel, and Ingrid Verbauwhede. "Chaskey: An Efficient MAC Algorithm for 32-bit Microcontrollers". In: *SAC 2014*. Ed. by Antoine Joux and Amr M. Youssef. Vol. 8781. LNCS. Springer, Heidelberg, Aug. 2014, pp. 306–323 (cit. on p. 110).

[MR18] Michael Meyer and Steffen Reith. "A Faster Way to the CSIDH". In: *INDOCRYPT 2018*. Ed. by Debrup Chakraborty and Tetsu Iwata. Vol. 11356. LNCS. Springer, Heidelberg, Dec. 2018, pp. 137–152 (cit. on p. 153).

[MS12] Lorenz Minder and Alistair Sinclair. "The Extended k-tree Algorithm". In: *Journal of Cryptology* 25.2 (Apr. 2012), pp. 349–382 (cit. on p. 61).

[MZ04] Michele Mosca and Christof Zalka. "Exact quantum Fourier transforms and discrete logarithm algorithms". In: *International Journal of Quantum Information* 02.01 (2004), pp. 91–100 (cit. on p. 30).

[NC10] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010 (cit. on p. 23).

[ND19] Boyu Ni and Xiaoyang Dong. "Improved quantum attack on Type-1 Generalized Feistel Schemes and Its application to CAST-256". In: *IACR Cryptology ePrint Archive* 2019 (2019), p. 318 (cit. on pp. 99, 105).

[NIST16] National, Institute, of Standards, and Technology (NIST). *Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process*. Dec. 2016 (cit. on pp. 15, 157).

[NS15] Ivica Nikolic and Yu Sasaki. "Refinements of the k-tree Algorithm for the Generalized Birthday Problem". In: *ASIACRYPT 2015, Part II*. Ed. by Tetsu Iwata and Jung Hee Cheon. Vol. 9453. LNCS. Springer, Heidelberg, Nov. 2015, pp. 683–703 (cit. on p. 61).

[Pat92] Jacques Patarin. "New Results on Pseudorandom Permutation Generators Based on the DES Scheme". In: *CRYPTO'91*. Ed. by Joan Feigenbaum. Vol. 576. LNCS. Springer, Heidelberg, Aug. 1992, pp. 301–312 (cit. on p. 103).

[Pen19] Daniel Penazzi. *Yarará and Coral v1*. NIST lightweight competition round 1 candidate. Mar. 2019 (cit. on p. 114).

[Per17] Лео Перрин. "Cryptanalysis, Reverse-Engineering and Design of Symmetric Cryptographic Algorithms". PhD thesis. University of Luxembourg, 2017 (cit. on p. 182).

[PM19] Daniel Penazzi and Miguel Montes. *Shamash (and Shamashash) (version 1)*. NIST lightweight competition round 1 candidate. Mar. 2019 (cit. on p. 114).

[Pol75] John M. Pollard. "A monte carlo method for factorization". In: *BIT Numerical Mathematics* 15.3 (Sept. 1975), pp. 331–334. ISSN: 1572-9125 (cit. on pp. 44, 46).

[Pol78] John M. Pollard. "Monte Carlo Methods for Index Computation (mod p)". In: *Mathematics of Computation* 32.143 (1978), pp. 918–924. ISSN: 00255718, 10886842 (cit. on p. 46).

[Reg04] Oded Regev. *A Subexponential Time Algorithm for the Dihedral Hidden Subgroup Problem with Polynomial Space*. 2004. eprint: arXiv:quant-ph/0406151 (cit. on pp. 57, 70, 71).

[Res18] Eric Rescorla. "The Transport Layer Security (TLS) Protocol Version 1.3". In: *RFC* 8446 (2018), pp. 1–160 (cit. on pp. 18, 111).

[Rog04] Phillip Rogaway. "Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC". In: *ASIACRYPT 2004*. Ed. by Pil Joong Lee. Vol. 3329. LNCS. Springer, Heidelberg, Dec. 2004, pp. 16–31 (cit. on p. 110).

[RS06] Alexander Rostovtsev and Anton Stolbunov. *Public-Key Cryptosystem Based On Isogenies*. Cryptology ePrint Archive, Report 2006/145. http://eprint.iacr.org/2006/145. 2006 (cit. on pp. 153, 154).

[RS15] Martin Roetteler and Rainer Steinwandt. "A note on quantum related-key attacks". In: *Inf. Process. Lett.* 115.1 (2015), pp. 40–44 (cit. on pp. 99, 106).

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: *Communications of the Association for Computing Machinery* 21.2 (1978), pp. 120–126 (cit. on p. 15).

[Saa19] Markku-Juhani O. Saarinen. *SNEIKEN and SNEIKHA*. NIST lightweight competition round 1 candidate. Mar. 2019 (cit. on p. 114).

[SE94] Claus-Peter Schnorr and M. Euchner. "Lattice basis reduction: Improved practical algorithms and solving subset sum problems". In: *Math. Program.* 66 (1994), pp. 181–199 (cit. on p. 158).

[SHA2] "Secure Hash Standard". In: *National Institute of Standards and Technology (NIST), NIST FIPS PUB 180-2, U.S. Department of Commerce* (Aug. 2001) (cit. on p. 101).

[SHA3] "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions". In: *National Institute of Standards and Technology (NIST), NIST FIPS PUB 202, U.S. Department of Commerce* (Aug. 2015) (cit. on pp. 19, 116).

[Sha49] Claude E. Shannon. "Communication theory of secrecy systems". In: *Bell Systems Technical Journal* 28.4 (1949), pp. 656–715 (cit. on pp. 14, 101).

[Sho94] Peter W. Shor. "Algorithms for Quantum Computation: Discrete Logarithms and Factoring". In: *35th FOCS*. IEEE Computer Society Press, Nov. 1994, pp. 124–134 (cit. on pp. 15, 30, 46, 63, 64).

[Sho96] Victor Shoup. "On Fast and Provably Secure Message Authentication Based on Universal Hashing". In: *CRYPTO'96*. Ed. by Neal Koblitz. Vol. 1109. LNCS. Springer, Heidelberg, Aug. 1996, pp. 313–328 (cit. on pp. 18, 111).

[SIKE] Reza Azarderakhsh, Brian Koziel, Matt Campagna, Brian LaMacchia, Craig Costello, Patrick Longa, Luca De Feo, Michael Naehrig, Basil Hess, Joost Renes, Amir Jalali, Vladimir Soukharev, David Jao, and David Urbanik. *Supersingular Isogeny Key Encapsulation*. Nov. 30, 2017. URL: http://sike.org (cit. on p. 153).

[Sil86] Joseph H. Silverman. *The arithmetic of elliptic curves*. Vol. 106. Graduate Texts in Mathematics. Springer-Verlag, 1986 (cit. on p. 154).

[Sim94] Daniel R. Simon. "On the Power of Quantum Computation". In: *35th FOCS*. IEEE Computer Society Press, Nov. 1994, pp. 116–123 (cit. on pp. 46, 49, 51).

[SMS19] Sumanta Sarkar, Kalikinkar Mandal, and Dhiman Saha. *Sycon v1.0*. NIST lightweight competition round 1 candidate. Mar. 2019 (cit. on p. 114).

[SS17] Thomas Santoli and Christian Schaffner. "Using Simon's algorithm to attack symmetric-key cryptographic primitives". In: *Quantum Information & Computation* 17.1&2 (2017), pp. 65–78 (cit. on pp. 99, 110).

[SS81] Richard Schroeppel and Adi Shamir. "A T=O($2^{n/2}$), S=O($2^{n/4}$) Algorithm for Certain NP-Complete Problems". In: *SIAM J. Comput.* 10.3 (1981), pp. 456–464 (cit. on pp. 58, 59).

[Sue21] Caius Suetonius Tranquillus. In: *Vita divi Iuli*. Vol. I. De Vita Caesarum. 121 (cit. on p. 13).

[Tat66] John Tate. "Endomorphisms of abelian varieties over finite fields". In: *Inventiones mathematicae* 2.2 (Apr. 1966), pp. 134–144. ISSN: 1432-1297 (cit. on p. 155).

[Tof80] Tommaso Toffoli. "Reversible Computing". In: *ICALP 80*. Ed. by J. W. de Bakker and Jan van Leeuwen. Vol. 85. LNCS. Springer, Heidelberg, July 1980, pp. 632–644 (cit. on p. 29).

[Ver19] Gilbert Sandford Vernam. "Secret signaling system". US 1310719A. June 1919 (cit. on p. 14).

[Vig86] Blaise de Vigenère. *Traicté des chiffres, ou Secrètes manières d'escrire*. Premier pillier de la grand' Salle du Palais, Paris: Abel l'Angelier, 1586 (cit. on p. 14).

[vW99] Paul C. van Oorschot and Michael J. Wiener. "Parallel Collision Search with Cryptanalytic Applications". In: *Journal of Cryptology* 12.1 (Jan. 1999), pp. 1–28 (cit. on p. 45).

[Wag02] David Wagner. "A Generalized Birthday Problem". In: *CRYPTO 2002*. Ed. by Moti Yung. Vol. 2442. LNCS. Springer, Heidelberg, Aug. 2002, pp. 288–303 (cit. on pp. 61, 62).

[WC81] Mark N. Wegman and Larry Carter. "New Hash Functions and Their Use in Authentication and Set Equality". In: *Journal of Computer and System Sciences* 22 (1981), pp. 265–279 (cit. on pp. 18, 111).

[Wie83] Stephen Wiesner. "Conjugate Coding". In: *SIGACT News* 15.1 (Jan. 1983), pp. 78–88. ISSN: 0163-5700 (cit. on p. 23).

[WP16] Hongjun Wu and Bart Preneel. *AEGIS*. CAESAR competition round 3 candidate. Sept. 2016 (cit. on p. 18).

[Wu16] Hongjun Wu. *ACORN*. CAESAR competition round 3 candidate. Sept. 2016 (cit. on p. 18).

[WZ82] W. K. Wootters and W. H. Zurek. "A single quantum cannot be cloned". In: *Nature* 299 (Oct. 1982), p. 802 (cit. on p. 26).

[Zal99] Christof Zalka. "Grover's quantum searching algorithm is optimal". In: *Physical Review A* 60.4 (1999), p. 2746 (cit. on p. 38).

[ZMI90] Yuliang Zheng, Tsutomu Matsumoto, and Hideki Imai. "On the Construction of Block Ciphers Provably Secure and Not Relying on Any Unproved Hypotheses". In: *CRYPTO'89*. Ed. by Gilles Brassard. Vol. 435. LNCS. Springer, Heidelberg, Aug. 1990, pp. 461–480 (cit. on p. 104).

Appendix A
Sequence of values to test to solve the quadratic equation of the AES S-Box

(d^{-1},x)	(d^{-1},x)	(d^{-1},x)	(d^{-1},x)	(d^{-1},x)	(d^{-1},x)
(0x1, 0xbc)	(0x3, 0x7e)	(0x5, 0x88)	(0x8, 0xd6)	(0x9, 0xb6)	(0xc, 0xf2)
(0xd, 0xec)	(0xe, 0xc4)	(0x11, 0xda)	(0x12, 0x72)	(0x13, 0x9e)	(0x17, 0x9a)
(0x18, 0x24)	(0x1a, 0x6e)	(0x1c, 0x44)	(0x1d, 0x8e)	(0x1e, 0xd4)	(0x1f, 0xd8)
(0x22, 0xe)	(0x23, 0xc0)	(0x24, 0x36)	(0x25, 0x9c)	(0x26, 0x58)	(0x29, 0xac)
(0x2b, 0xb8)	(0x2d, 0xa6)	(0x2e, 0xf4)	(0x31, 0x1a)	(0x34, 0xea)	(0x37, 0xa4)
(0x38, 0x56)	(0x3d, 0xe2)	(0x3e, 0x98)	(0x3f, 0x16)	(0x40, 0x3e)	(0x41, 0xf8)
(0x46, 0xe8)	(0x48, 0x60)	(0x4a, 0xce)	(0x4b, 0xba)	(0x4c, 0x8a)	(0x4e, 0x6a)
(0x4f, 0x3a)	(0x50, 0x42)	(0x51, 0xc)	(0x52, 0x94)	(0x54, 0x20)	(0x57, 0xca)
(0x58, 0xaa)	(0x5b, 0x7c)	(0x5c, 0x1e)	(0x5d, 0xfe)	(0x5f, 0x92)	(0x60, 0x2e)
(0x61, 0x26)	(0x62, 0xe6)	(0x64, 0xb4)	(0x65, 0xdc)	(0x67, 0x18)	(0x68, 0x54)
(0x69, 0x30)	(0x71, 0x4c)	(0x74, 0x2c)	(0x75, 0x66)	(0x76, 0x5e)	(0x78, 0xf0)
(0x7b, 0x2)	(0x7c, 0x62)	(0x7d, 0xd0)	(0x86, 0x76)	(0x87, 0xa0)	(0x8c, 0xc2)
(0x8d, 0x2a)	(0x8e, 0xc8)	(0x8f, 0xf6)	(0x99, 0x4)	(0x9c, 0x46)	(0xa0, 0x6c)
(0xa5, 0x74)	(0xa7, 0x8)	(0xaa, 0x6)	(0xab, 0x22)	(0xad, 0xee)	(0xb0, 0xae)
(0xb1, 0x50)	(0xb2, 0x14)	(0xb3, 0x68)	(0xb4, 0x90)	(0xb8, 0x4a)	(0xbc, 0xe0)
(0xbd, 0x5c)	(0xbf, 0x1c)	(0xc0, 0x10)	(0xc3, 0x48)	(0xc6, 0x78)	(0xc7, 0x38)
(0xc8, 0xe4)	(0xca, 0x34)	(0xcb, 0x28)	(0xcc, 0x3c)	(0xcd, 0xd2)	(0xce, 0x70)
(0xcf, 0x52)	(0xd1, 0xbe)	(0xd2, 0x5a)	(0xd6, 0x7a)	(0xd7, 0xfc)	(0xdd, 0xfa)
(0xe0, 0x4e)	(0xe1, 0x12)	(0xe3, 0x40)	(0xe5, 0x84)	(0xe7, 0xcc)	(0xe8, 0x86)
(0xe9, 0xa)	(0xeb, 0xc6)	(0xec, 0xb0)	(0xed, 0xa2)	(0xee, 0xa8)	(0xef, 0x64)
(0xf0, 0xb2)	(0xf5, 0x8c)	(0xf6, 0x96)	(0xfb, 0xde)	(0xfd, 0x80)	(0xfe, 0x32)
(0xff, 0x82)					

Table A.4: sequence of values to test, associated with the solution to write, from left to right and top to bottom.

Dans la même collection

Au sein de l'équipe SECRET :

– *Mathématiques discrètes appliquées à la cryptographie symétrique.* Yann Rotella, 2018

– *Constructions pour la cryptographie à bas coût.* Sébastien Duval, 2018

Au sein de l'équipe COSMIQ :

– *Cryptographie fondée sur les codes : nouvelles approches pour constructions et preuves ; contribution en cryptanalyse.* Thomas Debris-Alazard, 2019