Algebraic methods for security analysis of cryptographic algorithms implementations

Rina Zeitoun

UNIVERSITÉ PIERRE ET MARIE CURIE

École doctorale Informatique, Télécommunications et Électronique (Paris)

THÈSE

Pour obtenir le titre de

Docteur en Sciences

de l'UNIVERSITÉ PIERRE ET MARIE CURIE - PARIS 6

Mention Informatique

Présentée par

Rina Zeitoun

Méthodes algébriques pour l'analyse de sécurité des implantations d'algorithmes cryptographiques

Thèse dirigée par Jean-Charles Faugère et Guénaël Renault

soutenue le **jeudi 16 juillet 2015**

après avis des **rapporteurs**

M. Damien Stehlé	Professeur, École Normale Supérieure de Lyon
M. Jaime Gutierrez	Professeur, Université de Cantabria

devant le **jury** composé de

M. Jean-Charles Faugère	Directeur de recherche, INRIA
M. Guénaël Renault	Maître de conférences, Université Pierre et Marie Curie
M. Damien Stehlé	Professeur, École Normale Supérieure de Lyon
M. Jaime Gutierrez	Professeur, Université de Cantabria
M. Jean-Sébastien Coron	Professeur, Université du Luxembourg
M. Phong Nguyen	Directeur de recherche, INRIA
M. Stef Graillat	Professeur, Université Pierre et Marie Curie

---

Oberthur Technologies, Équipe Cryptographie, Technologie & Innovation

Contents

Notations

Notations

In this manuscript, the following notations are used to represent some widespread mathematical tools.

— We use row representation for matrices.
— Matrices are denoted by uppercase letters, and their coefficients are denoted by lowercase letters.
— The transposition of a matrix M is M^T.
— Vectors are row vectors denoted by bold lowercase letters.
— The transposition of a vector $\mathbf{v}$ is $\mathbf{v^T}$.
— The Euclidean norm is represented by $\|\|$.
— The inner product of $\mathbb{R}^n$ is represented by $\langle , \rangle$.
— The Euclidean norm is naturally extended to polynomials as follows : if $f(x) = \sum_{i=0}^{n} f_i x^i \in \mathbb{R}[x]$, then $\|f\| = (\sum_{i=0}^{n} f_i^2)^{1/2}$.
— We use the following matrix norms: if $M = (m_{i,j})$ is an $n \times m$ matrix, then
  — $\|M\|_2 = \max_{\|\mathbf{x}\| \neq 0} \frac{\|\mathbf{x}M\|}{\|\mathbf{x}\|}$,
  — $\|M\|_\infty = \max_{1 \leq j \leq m} \sum_{i=1}^{n} |m_{i,j}|$,
  — And we have $\|M\|_2 \leq \sqrt{n}\|M\|_\infty$.
— If $x \in \mathbb{R}$, we respectively denote by $\lfloor x \rfloor, \lceil x \rceil, \lceil x \rfloor$ the lower integer part, the upper integer part of x and the closest integer to x.
— All logarithms are in base 2.
— We write $f(n) = \mathcal{O}(g(n))$ if there exist constants n_0 and $c > 0$ such that $|f(n)| \leq c|g(n)|$ for all $n \geq n_0$.
— We write $f(n) = \Omega(g(n))$ if $g(n) = \mathcal{O}(f(n))$. Therefore $f(n) = \Omega(g(n))$ if and only if there exist constants n_0 and $c > 0$ such that $|f(n)| \geq c|g(n)|$ for all $n \geq n_0$.

Introduction

Cryptologie

À l'aube du 21$^{\text{ème}}$ siècle, les besoins cryptographiques explosent. Le chiffrement n'est plus réservé aux communications classifiées des armées : il gagne tous les domaines, avançant au rythme des découvertes mathématiques. Les applications civiles du chiffrement (transactions en ligne, vote électronique, usage de systèmes de communication, passeports, stockage dans le nuage, paiements électroniques, etc.) deviennent un moteur fondamental de progrès dans ce domaine. Les révélations récentes d'Edward Snowden concernant la surveillance mondiale secrète effectuée par la NSA renforcent amplement la nécessité du chiffrement. Ainsi, la cryptologie devient une science dynamique à l'intersection des mathématiques et de l'informatique. De nombreux protocoles cryptographiques sont continuellement élaborés. Dans le même temps, des études sont menées pour en assurer la sécurité, car ces cryptosystèmes ont des fins très concrètes : ils sont aussi bien utilisés pour sécuriser l'emploi d'Internet, qu'intégrés dans nos cartes bleues, cartes SIM, passeports, etc. Nombre de ces cryptosystèmes se révèlent vulnérables et sont abandonnés, d'autres plus robustes perdurent bien que nécessitant souvent des réadaptations face à des attaques mettant en avant la vulnérabilité de certains choix de paramètres. De nombreuses techniques d'analyse de vulnérabilité des cryptosystèmes sont employées. On peut citer par exemple les études basées sur des méthodes algébriques, c'est-à-dire modélisées par la résolution d'équations non-linéaires. D'autres types d'attaques telles que l'analyse des fuites physiques générées par les systèmes embarqués peuvent également permettre d'obtenir des informations secrètes. De fait, les deux types d'attaques peuvent naturellement parfois être combinées. Cette thèse se situe précisément dans ce contexte, celui où les attaques physiques constituent un apport crucial d'informations permettant de rendre la résolution du problème algébrique réalisable.

On distingue classiquement deux grandes catégories en matière de chiffrement : la cryptographie à clé secrète et la cryptographie à clé publique.

La cryptographie à clé secrète est de loin la plus ancienne. Elle nécessite au préalable la mise en commun entre les destinataires d'une clé secrète, puis consiste à utiliser cette même clé pour le chiffrement et le déchiffrement, (pour cette raison, on l'appelle également cryptographie symétrique). Elle est intuitive de par sa similarité avec ce que l'on s'attend à utiliser pour verrouiller et déverrouiller une porte : la même clé. Cependant, la principale difficulté de la mise en œuvre de ce système est l'échange en toute sûreté de la clé secrète

entre les deux parties.

La cryptographie à clé publique, dite asymétrique, s'attache à résoudre ce problème. Elle repose quant à elle sur un autre concept faisant intervenir pour chaque utilisateur une paire de clés : l'une pour le chiffrement, rendue publique, et l'autre pour le déchiffrement, conservée secrète. Les clés sont différentes mais elles sont liées et seul l'utilisateur associé à la paire de clés en connaît le lien. Afin de chiffrer un message à l'intention d'un utilisateur, le correspondant emploie la clé publique de cet utilisateur. Le déchiffrement du message chiffré nécessite la connaissance de la clé secrète, que seul l'utilisateur détient. Ce concept naturel permet de communiquer de manière confidentielle sans avoir à partager la moindre information secrète initialement. La cryptographie asymétrique est fondée sur l'utilisation d'une fonction à trappe : une fois cette fonction appliquée à un message, il est extrêmement difficile de retrouver le message original, à moins de posséder une information particulière tenue secrète : la clé privée. Toutefois, il reste une difficulté : trouver une fonction à trappe.

Cryptosystème RSA

Le premier modèle de chiffrement à clé publique, appelé RSA, proposant une fonction à trappe, a été mis en place en 1977 par Ron Rivest, Adi Shamir et Leonard Adleman [RSA78]. Ce cryptosystème a été le plus utilisé pendant de nombreuses années et est encore l'un des plus utilisés de nos jours (même si un remplacement progressif tend à s'effectuer vers des cryptosystèmes plus performants), notamment dans les systèmes embarqués tels que les cartes bancaires, cartes SIM, passeports, où une sécurité des ressources sensibles qu'ils contiennent doit être assurée.

Le cryptosystème RSA repose sur la fonction qui, à deux grands nombres premiers p et q associe leur produit $p \times q$. Elle est à sens unique car étant donné p et q, il est aisé de calculer $N = p \times q$, mais à l'inverse, connaissant un entier N produit de deux grands nombres premiers, il est très difficile de retrouver les facteurs p et q.

Le protocole cryptographique RSA fonctionne de la manière suivante. Un utilisateur souhaitant recevoir des messages de manière sécurisée, et dont il sera le seul à pouvoir en déchiffrer le contenu, choisit deux grands nombres premiers distincts p et q et calcule leur produit $N = p \times q$. Il choisit un entier e premier avec $\phi(N) = (p-1)(q-1)$ et calcule d tel que $ed = 1 \bmod \phi(N)$. Le couple (N, e) constitue la clé publique de l'utilisateur. Elle sera utilisée par ses correspondants pour le chiffrement. L'utilisateur garde secrète sa clé privée d et en fera usage pour déchiffrer. Un correspondant désirant lui envoyer un message m se procure la clé publique (N, e) de l'utilisateur puis calcule le message chiffré $C = m^e \bmod N$. C'est ce dernier nombre qu'il lui envoie. L'utilisateur reçoit C. Il calcule grâce à sa clé privée $D = C^d \bmod N$. D'après le théorème d'Euler, on a $D = m^{de} = m \bmod N$. Il a donc reconstitué le message initial.

Le cryptosystème à clé publique RSA a également été adapté à d'autres fins applicatives telles que la signature électronique permettant de garantir l'intégrité d'un document et de certifier son auteur comme tel. Le principe de la signature RSA est similaire à celui du chiffrement RSA (voir Chapitre 1.1) à ceci près que l'utilisateur fera usage de sa clé

privée d pour signer ses messages, et que la clé publique (N, e) de l'utilisateur sera utilisée par ses correspondants afin de vérifier ses signatures. Ainsi un utilisateur souhaitant signer un message m calcule la signature $S = m^d \bmod N$ et envoie le couple (m, S) au correspondant. Ce dernier calcule alors à l'aide de la clé publique (N, e) de l'utilisateur la valeur $S^e \bmod N$. Si la signature est correcte, ce résultat correspond précisément au message m.

Si le calcul de la vérification de signature $S^e \bmod N$ (respectivement le calcul du chiffrement d'un message $m^e \bmod N$) est généralement peu couteux car en pratique la clé publique e est toujours choisie petite (à cette fin justement, ainsi que pour des raisons de consommation mémoire), il n'en est pas de même du calcul de la signature $S = m^d \bmod N$ (respectivement du calcul du déchiffrement d'un message $C^d \bmod N$) car l'exposant secret d est nécessairement grand pour des raisons de sécurité. Aussi, dans les systèmes embarqués tels que les cartes à puces où les critères de performances sont souvent cruciaux, la plupart des implantations de RSA utilisent le mode CRT basé sur le Théorème des Restes Chinois, qui permet une accélération du calcul de cette exponentiation modulaire d'un facteur 4 [CQ82]. Ainsi, dans le cadre de la signature RSA, le mode CRT consiste à effectuer le calcul $S = m^d \bmod N$ en deux temps : une fois modulo p et une autre modulo q, puis la signature finale modulo N est obtenue par recombinaison des deux résultats, en utilisant par exemple la formule de Garner [Gar59] (un rappel est fourni au Chapitre 1.1).

Dans le cryptosystème RSA, il est aisé d'observer que la connaissance des entiers premiers p et q permet de retrouver la clé privée d de l'utilisateur. Actuellement, il n'y a aucune méthode connue, capable de factoriser dans un temps convenable de très grands entiers. Le fonctionnement du cryptosystème RSA est ainsi basé sur cette difficulté. RSA est donc un protocole cryptographique que l'on peut présumer sûr dès lors que la taille des entiers p et q est suffisamment grande. Typiquement, aujourd'hui la taille des premiers utilisés est de 512, 1024 ou 1536 bits, à savoir qu'une taille de 512 bits n'est déjà plus recommandée.

De toute évidence, le cryptosystème RSA a été une cible notable des attaquants. Mais de fait, si RSA est encore l'un des cryptosystèmes les plus utilisés aujourd'hui, c'est parce qu'il s'avère très résistant aux cryptanalyses théoriques dans le cas général. Cependant, de nombreuses attaques mettant en jeu des cas particuliers d'utilisation ou des paramètres vulnérables, ont été publiées. Par exemple, en 1989 Wiener montre, à l'aide d'un développement en fractions continues de N/e, que l'utilisation d'une petite clé secrète d est à bannir [Wie90]. De même, Håstad en 1985 montre qu'en interceptant le même message envoyé à plusieurs destinataires différents, il est possible de retrouver le message originel si la clé publique e est suffisamment petite [Hås85]. D'autres attaques permettant la factorisation de $N = pq$ s'appliquent lorsque le facteur premier p est tel que $p - 1$ ou $p + 1$ est friable (ne possédant que de petits facteurs premiers). À ce sujet, les méthodes $p - 1$ *de Pollard* [Pol74] et $p + 1$ *de Williams* [Wil82] sont les plus connues et l'adoption d'entiers qui ne soient pas friables devient nécessaire. C'est la raison pour

laquelle la norme ANSI X9.31 [ANS98] ou FIPS186-4 [FIP13] de génération de clés RSA s'attarde à générer des entiers dits *premiers forts*, respectant ces propriétés.

Attaques physiques sur système embarqué

Ainsi, les attaques proposées dénotent souvent plus un problème d'utilisation du cryptosystème qu'un problème de fond lié à la sécurité intrinsèque de celui-ci. Plus encore, la sécurité théorique d'un cryptosystème ne garantit pas forcément une sécurité lors de son utilisation dans la pratique. En effet, la mise en œuvre d'un protocole cryptographique dans un système embarqué tel qu'une carte à puce peut facilement être attaquée si elle a été réalisée sans précautions particulières.

Les *analyses par canaux auxiliaires* (*Side-Channel Analysis* en anglais et SCA en abrégé), introduites par les travaux de Paul Kocher en 1996 [Koc96], visent à exploiter les fuites d'informations physiques du système embarqué (voir Chapitre 1.2). Ainsi, certaines valeurs manipulées par le dispositif, portant de l'information secrète, peuvent être retrouvées par un attaquant lorsque ces dernières sont maniées sans précautions. À l'origine, le temps d'exécution était principalement utilisé comme fuite d'information exploitable, mais d'autres paramètres comme la consommation électrique ainsi que le rayonnement électromagnétique sont rapidement devenues les sources d'exploitation les plus efficaces pour attaquer la cryptographie embarquée [KJJ99, QS00].

Les fuites telles que la consommation électrique de la carte, peuvent être exploitées principalement de deux manières : si l'on considère une seule mesure, on peut effectuer une *analyse simple par courant* ou SPA (*Simple Power Analysis* en anglais) ; si l'on en considère plusieurs, une *Analyse différentielle par courant* ou DPA (*Differential Power Analysis* en anglais) peut être réalisée. Ces attaques sont dites passives, en ce sens que les données manipulées par le système embarqué ne sont pas modifiées par l'attaquant, mais seulement observées et analysées par celui-ci, afin d'obtenir des informations sensibles.

Une attaque SPA consiste à analyser les variations et les pics de la consommation électrique du circuit dans le but de découvrir des informations secrètes comme la clé de chiffrement. La signature RSA est typiquement vulnérable à ce type d'attaque si aucune précaution n'est prise. Par exemple, si l'exponentiation modulaire est implantée suivant l'algorithme *Square-and-Multiply*, où l'opération effectuée change selon que le bit traité soit 0 ou 1, l'exposant secret d peut directement être extrait par simple lecture d'une unique courbe car l'opération de mise au carré et celle de la multiplication signent différemment. Une contremesure naturelle consiste à employer des algorithmes dits *réguliers* qui effectuent la même opération peu importe la valeur du bit d'exposant (par exemple, les algorithmes *Square-Always* ou *Montgomery ladder* [JY02, CFG+11]).

Une attaque DPA nécessite quant à elle un grand nombre de mesures extraites de plusieurs exécutions utilisant la même clé. L'idée consiste à identifier une variable intermédiaire dite *sensible* manipulée durant l'exécution de l'algorithme qui dépend d'une petite partie de la clé secrète et d'une donnée connue qui peut être modifiée à chaque exécution de l'algorithme. Ainsi, dès lors qu'une variable sensible est identifiée, une re-

cherche exhaustive va pouvoir être effectuée sur la petite partie de la clé secrète et le choix correct sera validé à l'aide d'un traitement statistique mettant en corrélation la valeur sensible associée au choix du secret, et l'ensemble des courbes de fuites obtenues lors de la manipulation de cette variable : le choix correct est celui pour lequel le niveau de corrélation est le plus élevé. De fait, la signature RSA en mode CRT est vulnérable aux DPA. En effet, une attention particulière portée sur les valeurs intermédiaires des calculs permet de remarquer que la valeur $\lfloor S/q \rfloor$ peut à un moment donné être manipulée. Puisque cette valeur dépend de la clé secrète q ainsi que de la signature S qui peut être modifiée d'une exécution à l'autre, cette valeur est sensible et une DPA permettrait de la retrouver, et donc d'obtenir le secret q. Des contremesures classiques pour résister à la DPA consistent à employer des techniques de masquage, c'est-à-dire à randomiser le module N, le message m et l'exposant d comme décrit dans [AFV07] afin d'introduire une donnée inconnue qui est modifiée à chaque exécution de l'algorithme, ce qui rend l'attaque impraticable.

Les attaques par *injection de fautes* (*Fault Injection* en anglais et FI en abrégé) pourvoient à l'attaquant un autre chemin d'attaque (voir Chapitre 1.2). Ces attaques sont dites actives dans le sens où elles permettent la réalisation de modifications sur le système embarqué, allant d'une simple altération des données manipulées, jusqu'à la détérioration irréversible du matériel. Les attaques par faute visent à perturber les calculs cryptographiques, de sorte qu'une analyse du résultat erroné correspondant permet à l'attaquant de retrouver la clé secrète [GT04]. Le cryptosystème RSA a été le premier d'une longue liste (DES, ElGamal, DSA, etc.) à fléchir face aux attaques par fautes avec la très célèbre attaque dite de *Bellcore* [Bel96, BDL97] qui s'applique sur la signature RSA lorsque le mode CRT est employé. L'idée consiste à injecter une faute durant le calcul modulo p, et à laisser inchangé celui modulo q. Si l'on a accès à un couple de signatures correcte et erronée $(S, \tilde{S})$ du même message, un simple calcul du PGCD de $S - \tilde{S}$ avec N permet de retrouver l'entier secret premier q. Une contre-mesure naturelle consiste à vérifier l'exactitude de la signature S avant de la rendre publique, de sorte que la signature est retournée si et seulement si $S^e \bmod N = m$ et qu'un attaquant ne puisse jamais avoir accès à une signature erronée.

Ainsi, les SCA et FI ont soulevé un intérêt certain aussi bien au sein de la communauté académique qu'industrielle et ont été amplement étudiées au cours des deux dernières décennies. Les impacts dans le domaine de l'industrie de ces deux types d'attaques sont conséquents puisque les produits sécurisés doivent être certifiés afin de prouver leur résistance contre de telles menaces. Aussi, ces dernières années, la communauté cryptographique a également exploré l'éventuelle possibilité de combiner les deux types d'attaques. Ceci a donné lieu à la création d'une nouvelle classe d'attaques appelée *attaques combinées* (*Combined Attacks* en anglais) qui se focalisent particulièrement sur des implémentations supposées résistantes aux attaques par canaux auxiliaires et par fautes.

Attaques physiques combinées

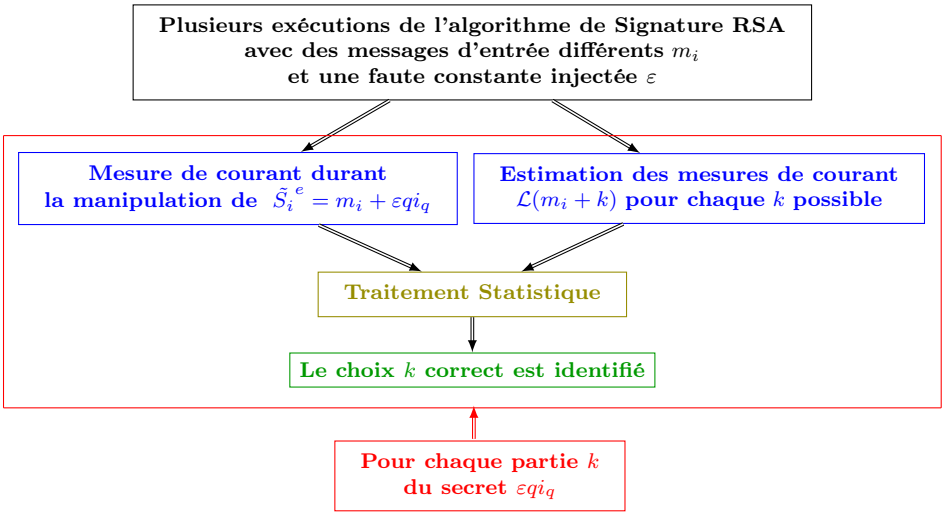
L'idée d'associer SCA et FI est apparue en 2007 avec la publication par Amiel, Feix, Marcel et Villegas d'une attaque combinée sur une implantation de RSA protégée contre les attaques par injection de fautes et les SPA [AFMV07]. Ils remarquèrent qu'en forçant, à l'aide d'une FA, la mise à zéro de l'un des registres temporaires utilisés dans l'algorithme *Montgomery ladder*, sa structure se déséquilibrait, permettant ensuite la révélation de l'exposant secret par SPA. Suite à cette publication, trois autres papiers ont vu le jour, basés sur cette nouvelle manière d'attaquer les systèmes embarqués. Deux d'entre eux présentent une attaque combinée contre une implantation sécurisée de l'AES [RM07, RLK11]. Le troisième est axé sur la multiplication scalaire employée au sein de cryptosystèmes basés sur les courbes elliptiques [FGV11]. Les attaques combinées restent cependant assez peu nombreuses, prouvant ainsi la difficulté de concevoir de telles attaques.

Proposition d'une Attaque Combinée FI/DPA

Une première contribution de cette thèse qui sera détaillée dans le Chapitre 6 consiste en la proposition d'une nouvelle attaque combinée sur une implantation RSA en mode CRT, résistante aux SCA (grâce à l'utilisation de techniques de masquage) et protégée contre les attaques par fautes (grâce à la vérification de la signature en utilisant l'exposant public e). Une telle implantation était connue pour résister à toutes les attaques publiées jusqu'alors. Cependant, nous montrons qu'en injectant une faute durant le calcul de la signature, une variable sensible sera manipulée lors de la vérification publique, de sorte qu'une SCA pourra subséquemment être appliquée afin de la retrouver, et obtenir dans le même temps l'un des facteurs premiers secrets. Plus précisément, si l'on faute le message m pour le calcul de la signature modulo p de sorte que le message fauté $\tilde{m}$ soit tel que $\tilde{m} = m + \varepsilon$ et que l'on laisse inchangé le calcul modulo q, alors la valeur manipulée lors de la vérification publique avec l'exposant e n'est plus $S^e \equiv m \bmod N$, mais $\tilde{S}^e \equiv m + \varepsilon q i_q \bmod N$ où $i_q = q^{-1} \bmod p$. Ainsi, on remarque aisément que cette valeur est sensible puisqu'elle dépend d'une partie secrète $\varepsilon q i_q$ qui ne change pas d'une exécution à l'autre (sur la base de l'hypothèse que ε reste constant) et d'une partie connue m qui peut être modifiée à chaque exécution de l'algorithme. L'application d'une DPA permettrait donc de retrouver $\varepsilon q i_q$ comme illustré en Figure 1. Par suite, un simple calcul du PGCD de $\varepsilon q i_q$ avec N permet de retrouver l'entier secret premier q, et donc la factorisation de N.

Ainsi, le but de la réalisation de cette DPA est de retrouver grâce à des fuites du modèle physique, la valeur de l'inconnue $\varepsilon q i_q$ (afin de factoriser N comme expliqué précédemment). En fait, une analyse théorique permet de modéliser le problème par la recherche des racines entières d'un certain polynôme à deux variables modulo N. En effet, grâce à l'identité de Bézout, nous avons la relation $p i_p + q i_q \equiv 1 \bmod N$ où $i_p = p^{-1} \bmod q$ et $i_q = q^{-1} \bmod p$. Ainsi, en multipliant la relation par $\varepsilon^2 q i_q$, le terme $\varepsilon^2 q i_q \times p i_p$ dis-

FIGURE 1 – CPA durant la recombinaison CRT effectuée lors de la signature RSA.



paraît car $pq \equiv 0 \bmod N$, ce qui donne l'équation finale : $\varepsilon^2 q^2 i_q^2 - \varepsilon^2 q i_q \equiv 0 \bmod N$. Si l'on suppose la faute ε connue ou petite (de sorte qu'elle puisse être recherchée exhaustivement), la valeur de l'inconnue $\varepsilon q i_q$ est solution entière y_0 de l'équation modulaire $y^2 - \varepsilon y \equiv 0 \bmod N$.

De même, le problème plus général de la factorisation RSA peut également être modélisé par la résolution de l'équation polynomiale à deux variables sur les entiers $N - xy = 0$ où $x_0 = p$ et $y_0 = q$ sont les solutions entières recherchées.

De fait, la recherche des secrets de l'ensemble des cryptosystèmes existants peut se modéliser par la résolution d'équations ou de systèmes d'équations à une ou plusieurs variables. La sécurité de ces cryptosystèmes est donc intrinsèquement liée à la difficulté de résoudre de telles équations. Le problème Diophantien consistant à trouver les solutions entières d'équations polynomiales, spécifié comme étant le *$10^{ème}$ problème de Hilbert*, a été prouvé indécidable en 1970 [Mat00].

Problématique

Un sous-problème crucial en cryptanalyse consiste à s'intéresser aux solutions existantes au sein de certains sous-espaces, comme par exemple, chercher l'ensemble des *petites* solutions entières de certaines équations polynomiales. Ce problème est pertinent en cryptologie car si l'ensemble des attaques physiques décrites précédemment (SCA, FI) permettent quelques fois de retrouver la totalité de certaines données secrètes, dans nombre de cas, seulement une partie du secret est dévoilée. L'avantage d'un attaquant connaissant une partie du secret se modélise précisément par le fait qu'il n'ait à rechercher que les solutions entières d'équations polynomiales qui soient petites. La très célèbre méthode de Coppersmith s'attache justement à résoudre ce problème, et notamment à

spécifier la notion du terme « petit » jusqu'alors imprécise.

Méthodes de Coppersmith pour trouver les petites solutions entières d'équations polynomiales

Coppersmith s'est intéressé à deux formes de polynômes en particulier, à savoir les polynômes univariés modulo un entier N de factorisation inconnue $f(x) \equiv 0 \mod N$, ainsi que les polynômes bivariés sur les entiers $f(x, y) = 0$. La résolution de ces deux formes de polynômes est en effet particulièrement intéressante dans le cadre de la cryptanalyse de RSA. Comme il a été précisé précédemment, trouver toutes les solutions entières de ces polynômes est un problème difficile, à savoir, il n'existe pas d'algorithme s'exécutant en temps polynomial permettant d'y arriver (ni de déterminer l'existence de telles solutions !). Cependant, Coppersmith publia en 1996 [Cop96b, Cop96a, Cop97] une méthode pour trouver efficacement l'ensemble des petites solutions de ces équations polynomiales. Son résultat le plus simple et peut-être le plus célèbre concerne le cas univarié modulaire, et est le suivant : étant donnés un entier N de factorisation inconnue et un polynôme unimodulaire $f(x)$ à coefficients entiers, de degré δ, on peut retrouver toutes les solutions entières x_0 telles que $f(x_0) \equiv 0 \pmod N$ et $|x_0| \leq N^{1/\delta}$ en temps polynomial en $\log N$ et δ.

La méthode de Coppersmith s'attache à obtenir, à partir du polynôme modulaire f, un nouveau polynôme v admettant les mêmes solutions, mais ayant la propriété qu'il tienne sur les entiers de sorte qu'il puisse être résolu facilement sur $\mathbb{Z}$. Plus précisément, la méthode construit un polynôme $v(x) \in \mathbb{Z}[x]$ tel que : si $x_0 \in \mathbb{Z}$ est tel que $f(x_0) \equiv 0 \pmod N$ et $|x_0| \leq X$, alors $v(x_0) = 0$ et peut être résolu sur $\mathbb{Z}$.

La méthode de Coppersmith est basée sur des techniques de réseaux. Un réseau est un arrangement régulier et infini de points dans l'espace, défini par une base. Toutefois, il existe une infinité de bases pour représenter un même réseau, et certaines bases ont des propriétés plus avantageuses que d'autres, en particulier celles comportant des vecteurs courts et relativement orthogonaux. Étant donnée une base quelconque, trouver une base contenant le vecteur le plus court du réseau (connu comme le *problème du plus court vecteur* ou SVP) est un problème NP-difficile [Ajt96]. Cependant il existe des algorithmes dits de réduction-LLL, traitant une notion plus allégée de réduction de réseaux (comme les algorithmes LLL, L^2 et $\tilde{L}^1$) permettant de trouver un vecteur relativement court en temps polynomial en la taille des éléments du réseau. La méthode de Coppersmith utilise précisément ces algorithmes afin d'obtenir, à partir d'un vecteur possédant de grands coefficients (correspondant au polynôme original f), un vecteur court (correspondant au nouveau polynôme v) comportant des coefficients plus petits. De par la construction du réseau initial de Coppersmith, ce nouveau polynôme v a la caractéristique qu'il admet les mêmes racines que le polynôme f. Cependant, ayant de petits coefficients, ce polynôme v s'annulera en x_0 sur les entiers, sous réserve que la solution recherchée x_0 soit également petite, d'où l'efficacité de la méthode de Coppersmith pour trouver uniquement les petites solutions.

Quelques applications des méthodes de Coppersmith

De nombreuses applications de la méthode de Coppersmith ont vu le jour dans le domaine de la cryptanalyse à clé publique (par exemple, pour attaquer des cas particuliers de RSA, ou pour factoriser N avec la connaissance de certains indices ou certaines parties des secrets p, q, d, etc.), mais aussi dans quelques preuves de sécurité (comme dans RSA-OAEP [Sho02]). De fait, les travaux de Coppersmith ont donné lieu à des dizaines d'articles introduisant de nouvelles variantes, généralisations et simplifications (notamment celle due à Howgrave-Graham [HG97] devenue une référence dans le domaine). Les applications ont également été nombreuses (se référer à [May10]). Les plus connues sont sans doute la factorisation du module RSA $N = pq$ avec la connaissance de la moitié des bits de p [Cop96a], avec un petit exposant public e (typiquement lorsque $e = 3$) [Cop97], avec des petits exposants-CRT secrets [BM06] ou encore lorsque $d < N^{0.29}$ [BD99].

Application à l'attaque combinée sur CRT-RSA

Dans ce contexte, une analyse de la contribution présentée en Figure 1 nous a permis de proposer une amélioration de la complexité de l'attaque grâce à la méthode de Coppersmith. En effet, la DPA employée au sein de l'attaque combinée qui permet de retrouver la variable sensible $\varepsilon q i_q$ s'effectue partie par partie (typiquement par tranche de 8 bits). Ainsi, dans l'équation modulaire $\varepsilon^2 q^2 i_q^2 - \varepsilon^2 q i_q \equiv 0 \bmod N$ précédemment obtenue, si l'on écrit $\varepsilon q i_q = 2^t k + x$ où $2^t k$ représente la partie haute de $\varepsilon q i_q$ connue grâce à la DPA déjà effectuée sur cette partie, et où x représente la partie basse non encore obtenue, l'on obtient que le secret x est solution d'une équation univariée modulaire de degré 2 (si l'on suppose la faute ε connue ou petite). D'après le théorème de Coppersmith, la solution x peut être retrouvée si sa taille est plus petite que la moitié de celle de N. Ainsi, dans le cas où N est un entier de 2048 bits, il suffit de retrouver les 1024 bits de poids fort de $\varepsilon q i_q$ pour obtenir spontanément les 1024 bits de poids faible, ce qui conduit à une accélération significative de l'attaque. Il est intéressant de noter que les mêmes résultats sont obtenus si la DPA procure en premier lieu les bits de poids faible et non de poids fort. Une extension de l'attaque combinée traitant le cas où la faute ε est inconnue est également proposée. Ces résultats sont détaillés dans le Chapitre 6.

Factorisation de $N = p^r q$ lorsque r est grand : état de l'art

La factorisation des modules de la forme $N = p^r q$ constitue une extension pertinente de la méthode de Coppersmith. De fait, l'utilisation de tels modules a été introduite en cryptographie il y a plusieurs années avec la proposition de certaines applications, notamment pour le cas $r = 2$, avec la conception par Fujioka *et al.* [FOM91] d'un schéma de paiement électronique mettant en avant l'emploi d'un module $N = p^2 q$, ainsi qu'avec la construction d'un cryptosystème à clé publique probabiliste par Okamoto et Ushiyama [OU98]. Plus généralement, il a été souligné par Takagi dans [Tak98] que l'utilisation de modules $N = p^r q$ pour RSA pouvait conduire à un déchiffrement

significativement plus rapide qu'avec l'utilisation de modules classiques $N = pq$.

À Crypto 99, Boneh, Durfee et Howgrave-Graham (BDH) analysèrent la sécurité face à l'utilisation de tels modules, en ce qui concerne les méthodes basées sur les réseaux. Les auteurs aboutirent en la conception d'une méthode pour factoriser les modules $N = p^r q$, grâce à une adaptation de la méthode de Coppersmith pour la factorisation des polynômes univariés modulaires. La condition pour obtenir une factorisation en temps polynomial est que l'exposant r soit grand, à savoir que $r \simeq \log p$ lorsque $q \leqslant p^{\mathcal{O}(1)}$ [BDHG99]. En fait, les auteurs montrent que la connaissance d'une fraction $1/(r+1)$ des bits de p est suffisante pour factoriser $N = p^r q$ en temps polynomial. Ainsi, lorsque $r \simeq \log p$, la connaissance d'un nombre *constant* de bits de p est nécessaire. Par conséquent, ces bits peuvent être retrouvés par recherche exhaustive, ce qui rend la factorisation de $N = p^r q$ réalisable en temps polynomial.

Ainsi, la méthode de BDH met en avant la vulnérabilité de l'utilisation de tels modules $N = p^r q$. On pourrait naturellement être tenté d'utiliser des modules de la forme $N = p^r q^s$ afin d'éviter l'attaque précédente, d'autant plus que la technique employée dans [Tak98] pour un déchiffrement rapide modulo p peut également être appliquée à q, ce qui apporte une accélération supplémentaire comme cela a été montré à Indocrypt 2000 dans [LKYL00] : ainsi, l'utilisation d'un module $N = p^2 q^3$ de 8196 bits permet un déchiffrement 15 fois plus rapide en comparaison à l'emploi d'un module RSA classique $N = pq$ de la même taille. Aussi, dans [BDHG99] les auteurs ont laissé explicitement ouvert le problème de la généralisation de la méthode BDH à des modules de la forme $N = p^r q^s$ lorsque r et s ont approximativement la même taille.

Proposition d'une méthode pour factoriser $N = p^r q^s$

Dans ce contexte, une contribution de cette thèse qui sera détaillée dans le Chapitre 5, consiste en l'apport d'une solution à ce problème ouvert : de tels modules $N = p^r q^s$ devraient également être utilisés avec précaution, puisque lorsque r ou s est grand, factoriser $N = p^r q^s$ peut également se faire en temps polynomial. En effet, nous proposons un nouvel algorithme déterministe pour factoriser $N = p^r q^s$ en temps polynomial lorsque r ou s est plus grand que $(\log p)^3$.

Deux tentatives naturelles pour arriver à ce résultat échouent. La première serait d'écrire $Q := q^s$ et d'appliquer la méthode BDH sur le module $N = p^r Q$, cependant la condition pour une factorisation polynomiale serait $r \simeq \log Q \simeq s \log q$, ce qui n'aboutit pas si r et s ont approximativement la même taille. La deuxième approche consisterait à écrire $N = (P + x)^r (Q + y)^s$ et à appliquer le théorème de Coppersmith pour le cas bivarié modulaire sur les entiers. Cependant, la condition serait $p \cdot q < p^{2/3} q^{2s/(3r)}$, ce qui ne donne jamais lieu à une factorisation en temps polynomial.

La méthode que nous proposons fait appel aux deux techniques suivantes : celle de Coppersmith traitant le cas univarié modulaire, ainsi que son extension proposée par BDH. Nous illustrons en premier lieu notre méthode à l'aide d'un cas particulier : le module de la forme $N = p^r q^{r-1}$. Comme expliqué précédemment, la méthode BDH ne peut pas être appliquée directement à $N = p^r Q$ avec $Q = q^{r-1}$, car la condition pour

une factorisation en temps polynomial serait $r = \Omega(\log Q) = (r-1)\Omega(\log q)$, condition qui n'est jamais satisfaite. Toutefois, il est possible d'écrire N de la façon suivante : $N = (pq)^{r-1}p = P^{r-1}Q$ avec $P := pq$ et $Q := p$. Cette représentation permet d'appliquer la méthode BDH pour retrouver P et Q (et donc p et q), avec $r = \Omega(\log Q) = \Omega(\log p)$ comme condition pour une factorisation en temps polynomial, cette condition étant essentiellement la même que celle obtenue dans la méthode BDH. Par conséquent, cela met en avant le fait que $N = p^r q$ n'est pas la seule classe d'entiers qui peut être factorisée de manière efficace ; il est également possible de factoriser des modules de la forme $N = p^r q^{r-1}$ en temps polynomial lorsque r est suffisamment grand.

Il est aisé de généraliser l'observation précédente à l'ensemble des modules de la forme $N = p^{\alpha \cdot r + a} q^{\beta \cdot r + b}$ lorsque les entiers α, β, a et b sont petits. En effet, on peut écrire $P := p^\alpha q^\beta$ et $Q := p^a q^b$ et appliquer BDH sur $N = P^r Q$ pour retrouver P et Q (donc p et q). La condition pour une factorisation en temps polynomial est de nouveau $r = \Omega(\log Q)$, qui, pour des petites valeurs a et b donne la même condition $r = \Omega(\log p)$ que précédemment (en supposant que p et q ont une taille similaire).

Il est ensuite naturel de se demander si l'on peut généraliser cette méthode pour l'ensemble des modules $N = p^r q^s$. Autrement dit, une question intéressante est de se demander quelles sont les classes d'entiers (r, s) pouvant être représentées ainsi :

$$\begin{cases} r &= u \cdot \alpha + a \\ s &= u \cdot \beta + b \end{cases} \tag{1}$$

où u est un entier suffisamment grand et α, β, a, b des entiers suffisamment petits, pour que la méthode précédente puisse être appliquée (à savoir, le module $N = p^r q^s$ serait représenté par $N = P^u Q$ où $P := p^\alpha q^\beta$ et $Q := p^a q^b$, et la méthode BDH serait appliquée sur $N = P^u Q$ afin de retrouver P et Q, et donc p et q). Aussi, le résultat que nous obtenons est le suivant :

Théorème 1. *Soit $N = p^r q^s$ un entier de factorisation inconnue avec $r > s$ et $\mathrm{pgcd}(r, s) = 1$. Les facteurs premiers p et q peuvent être retrouvés en temps polynomial en $\log N$ si la condition suivante est satisfaite :*

$$r = \Omega(\log^3 \max(p, q)) \quad .$$

En effet, sous cette condition, nous sommes assurés de trouver une « bonne » décomposition de r et s suivant (1), permettant une factorisation en temps polynomial de $N = p^r q^s$. Ainsi, une nouvelle classe d'entiers pouvant être factorisés efficacement est identifiée : celle des modules $N = p^r q^s$ lorsque r ou s est grand.

En outre, pour obtenir la borne $\Omega(\log^3 \max(p, q))$, il est essentiel de considérer également une méthode de factorisation alternative. En effet, si l'on examine de nouveau le module initial $N = p^r q^{r-1}$, nous remarquons que l'on peut également écrire $N = (pq)^r / q$, ce qui conduit à la relation $(pq)^r \equiv 0 \pmod N$. Par conséquent, $P = pq$ est une petite racine d'un polynôme univarié modulo N et de degré r. Ainsi, l'on peut appliquer le premier théorème de Coppersmith pour trouver les petites solutions des polynômes univariés modulaires avec la condition $P < N^{1/r} = P q^{-1/r}$. Cette condition peut être satisfaite

en effectuant une recherche exhaustive sur les $(\log q)/r$ bits de poids fort de P, ce qui reste réalisable en temps polynomial si $r = \Omega(\log q)$. En conséquence, l'on obtient une deuxième méthode (basée sur le théorème de Coppersmith pour le cas univarié modulaire) pour factoriser les modules de la forme $N = p^r q^{r-1}$ sous la condition $r = \Omega(\log q)$, condition identique à celle obtenue par l'utilisation de la première méthode (basée sur BDH). Comme précédemment, cette observation peut se généraliser aisément aux modules de la forme $N = p^{\alpha \cdot r+a} q^{\beta \cdot r+b}$ lorsque α, β, $|a|$ et $|b|$ sont suffisamment petits, et où cette fois-ci, les entiers a et b sont tous deux négatifs.

Ainsi, nous montrons dans cette thèse que l'utilisation alternée des deux méthodes (BDH et Coppersmith) permet la factorisation en temps polynomial des modules de la forme $N = p^r q^s$ lorsque r ou s (le « ou » est non exclusif) est de l'ordre de $\Omega(\log^3 \max(p, q))$. Nous soulignons le fait que les deux méthodes, utilisées de manière alternée selon les modules, sont essentielles pour l'obtention d'une telle condition : dans le cas où une seule méthode (soit BDH, soit Coppersmith) est employée, la condition plus forte $\Omega(\log^5 \max(p, q))$ semble nécessaire.

Généralisation aux modules $N = \prod_{i=1}^{k} p_i^{r_i}$

Nous proposons également dans le Chapitre 5 une généralisation de cette méthode aux modules de la forme $N = \prod_{i=1}^{k} p_i^{r_i}$. En particulier, nous montrons qu'il est toujours possible d'extraire un facteur non trivial de N en temps polynomial si l'un des k exposants r_i est plus grand que $\log^{\theta_k}(\max p_i)$, où les premières valeurs de θ_k sont données en Table 1 (à savoir $\theta_2 = 3$, $\theta_3 = 9$, $\theta_4 = 25$, etc.) et plus généralement $\theta_k \sim \mathcal{O}(k!)$ lorsque k est grand. Ainsi, l'exposant θ_k grandit exponentiellement avec le nombre de facteurs premiers k, cependant, pour une valeur de k fixée, extraire un facteur non trivial de N s'effectue en temps polynomial en $\log N$.

TABLE 1 – Valeurs de θ_k pour un module $N = \prod_{i=1}^{k} p_i^{r_i}$ avec k facteurs premiers. La condition sur le plus grand exposant r_j est $r_j = \Omega(\log^{\theta_k} \max p_i)$.

k	**2**	**3**	**4**	**5**	**6**
$\theta_{\mathbf{k}}$	3	9	25	81	321

Résultats d'implantation

Nous avons implanté notre algorithme en considérant quatre modules $N = p^r q^s$ où $r = 8$, et $s = 1, 3, 5, 7$, avec des premiers p et q de 128 bits. D'après notre analyse pour chaque module N, nous avons indiqué en Table 2, la meilleure méthode à employer (BDH ou Coppersmith) ainsi que la décomposition correspondante. Le temps d'une réduction-LLL est également précisé (réalisé sur un PC 3.20-GHz Intel Xeon), ainsi que le temps total estimé pour factoriser N (obtenu en multipliant le temps d'une réduction-LLL par 2^t où t est le nombre de bits manquants, sur lesquels la recherche exhaustive s'effectue).

TABLE 2 – Module N, méthode employée (BDH ou Coppersmith), nombre de bits manquants, dimension du réseau, temps d'exécution de la réduction-LLL, et temps global estimé.

	Méthode	$(p^\alpha q^\beta)^u p^a q^b$	bits manquants	dim.	LLL	Temps estimé
$\mathbf{N = p^8 q}$	BDH	$p^8 q$	29	68	8.6 s	146 années
$\mathbf{N = p^8 q^3}$	Copp.	$(p^2 q)^4 q^{-1}$	51	61	4.2 s	$3 \cdot 10^8$ années
$\mathbf{N = p^8 q^5}$	BDH	$(p^2 q)^4 q$	55	105	1.3 s	$2 \cdot 10^9$ années
$\mathbf{N = p^8 q^7}$	Copp.	$(pq)^8 q^{-1}$	38	81	26 s	$2 \cdot 10^5$ années

Notre méthode de factorisation de $N = p^r q^s$ (tout comme celle de BDH pour $N = p^r q$) est moins performante que la méthode ECM [Len87] pour des tailles de p et q relativement petites comme c'est le cas dans nos expériences (p et q de 128 bits). Cependant, notre algorithme s'exécute en temps polynomial en la taille de N tandis que ECM est exponentiel, ce qui signifie que notre algorithme devient plus intéressant que ECM pour des tailles de p et q assez grandes.

Performance de ces méthodes basées sur les réseaux en pratique

Tout ces algorithmes permettant de trouver les petites solutions d'équations polynomiales sont basés sur la même idée qui consiste à obtenir de nouveaux polynômes grâce à la réduction de réseau. En théorie, ceci peut être réalisé en temps polynomial grâce aux algorithmes de réduction-LLL, cependant en pratique cela n'est pas tout aussi trivial. En effet, le temps d'exécution asymptotique est un polynôme de haut degré, car le réseau à réduire est gigantesque. Plus précisément, la complexité de la méthode de Coppersmith est $\mathcal{O}((\log^9 N)/\delta^2)$ lorsque l'algorithme L^2 de Nguyen et Stehlé [NS09] est utilisé pour effectuer la réduction-LLL. Les applications courantes de la méthode de Coppersmith comportent des polynômes de faible degré ($\delta \leqslant 9$), toutefois la valeur $\log N$ est la taille d'un module RSA donc elle ne vaut pas moins de 1024, ce qui rend la complexité théorique considérable : la quantité $\log^9 N$ vaut déjà plus de 2^{90}.

Ainsi, le véritable goulot d'étranglement de l'ensemble des algorithmes basés sur la méthode de Coppersmith est la réduction-LLL. Malgré l'attention considérable portée sur ces algorithmes, aucune amélioration conséquente permettant de réduire leur temps d'exécution n'a été publiée, hormis le fait que les algorithmes de réduction-LLL ont connu des avancées depuis la publication de l'article [Cop97] (avec l'apparition de L^2 [NS09] et $\tilde{L}^1$ [NSV11]). Ce problème apparaît dans les expériences (voir [CNS99]) : en pratique on ne peut trouver les petites racines que jusqu'à une borne qui est plus petite que la borne théorique annoncée. Ce point peut être illustré par l'attaque de Boneh-Durfee [BD00] sur RSA lorsqu'un petit exposant secret est utilisé. En particulier, la borne théorique permettant de factoriser N est $d \leq N^{1-1/\sqrt{2}} \approx N^{0.292}$, mais le plus grand d que font apparaître les expériences de Boneh-Durfee est seulement $d \approx N^{0.280}$ et ce, pour un

module N de 1000 bits. Lorsque N grandit, les résultats pratiques s'éloignent plus encore de la borne théorique, avec par exemple $d \approx N^{0.265}$ pour N de 4000 bits.

Proposition d'accélération de la méthode de Coppersmith

Dans ce contexte, où la borne théorique énoncée par la méthode de Coppersmith est souvent difficilement atteignable, voire totalement inaccessible à cause du temps d'exécution fort conséquent en pratique, une contribution de cette thèse (présentée dans le Chapitre 4) consiste en la proposition de deux méthodes permettant l'accélération du temps d'exécution de l'algorithme de Coppersmith pour le cas univarié modulaire ; les deux méthodes pouvant être combinées en pratique.

La première accélération résulte de l'application de l'algorithme de réduction-LLL sur une matrice où les éléments sont tronqués (*Rounding* en anglais). Plus précisément, au lieu de réduire la matrice de Coppersmith contenant des éléments gigantesques, l'idée consiste à tronquer les coefficients de manière appropriée avant d'effectuer la réduction-LLL dans le but de les rendre considérablement plus petits. En procédant de la sorte, nous montrons que la matrice ainsi réduite permet d'obtenir des vecteurs du réseau suffisamment courts. En pratique, cela signifie que pour toute instanciation de l'algorithme de Coppersmith permettant de trouver les petites solutions inférieures à une borne X, il est possible de diminuer considérablement la taille des coefficients de la matrice à réduire (asymptotiquement, la taille des éléments est allégée d'un facteur $(\log N)/\delta$), tout en atteignant quasiment la même borne X sur les solutions retrouvées.

Si cette stratégie consistant à tronquer les éléments de la matrice de Coppersmith avant d'y appliquer la réduction-LLL est plutôt naturelle, il n'est pas avéré qu'elle puisse être employée sur tout type de matrice. En effet, lorsque l'on tronque les éléments d'une matrice arbitraire non-singulière, celle-ci pourrait devenir singulière, ce qui empirerait la situation pour la réduction-LLL. Cependant, nous montrons qu'une stratégie adaptée fonctionne pour le cas particulier des matrices utilisées par l'algorithme de Coppersmith. En effet, l'on exploite le fait que les matrices à réduire sont triangulaires et que les éléments de la diagonale sont relativement équilibrés.

Il est intéressant de noter que cette propriété peut également être utilisée pour améliorer la complexité de la méthode de Coppersmith par une simple analyse, donc sans même modifier la méthode, en remarquant que le nombre d'itérations de l'algorithme de réduction-LLL est fortement lié à l'équilibrage des éléments de la diagonale (voir Chapitre 4).

Ainsi, l'ensemble des complexités obtenues sont présentées en Table 3, en fonction de l'algorithme de réduction de réseau employé (LLL, L^2 ou $\tilde{L}^1$). Plus précisément, la complexité originale de l'algorithme de Coppersmith est fournie en première ligne à titre de comparaison. La deuxième ligne représente la complexité raffinée avec prise en compte du lien entre le nombre d'itérations et l'équilibrage des éléments. Enfin, la troisième ligne met en évidence les complexités obtenues par l'application de la méthode Rounding.

Par exemple, la complexité originale de la méthode de Coppersmith avec l'utilisation de L^2 est $\mathcal{O}((\log^9 N)/\delta^2)$. L'analyse raffinée que l'on propose permet en réalité de la réduire à $\mathcal{O}((\log^8 N)/\delta)$. En outre, l'application de la méthode Rounding apporte une

TABLE 3 – Complexité de l'algorithme de Coppersmith en prenant en compte l'analyse originale, l'analyse raffinée et la méthode Rounding. Ces trois complexités dépendent de l'algorithme de réduction de réseau employé (LLL, L^2 ou $\tilde{L}^1$).

	LLL	L^2	$\tilde{L}^1$
Analyse Originale	$\mathcal{O}((\log^{12} N)/\delta^3)$	$\mathcal{O}((\log^9 N)/\delta^2)$	$\mathcal{O}((\log^{7+\varepsilon} N)/\delta)$
Analyse Raffinée	$\mathcal{O}((\log^{11} N)/\delta^2)$	$\mathcal{O}((\log^8 N)/\delta)$	$\mathcal{O}(\log^{6+\varepsilon} N)$
Méthode Rounding	$\mathcal{O}(\log^9 N)$	$\mathcal{O}(\log^7 N)$	$\mathcal{O}(\log^{6+\varepsilon} N)$

amélioration supplémentaire de la complexité qui devient $\mathcal{O}(\log^7 N)$. Ainsi, l'accélération totale $\Theta((\log^2 N)/\delta^2)$ est quadratique en la taille de la borne sur les petites solutions $N^{1/\delta}$. Le gain est également conséquent avec l'utilisation de LLL où la complexité originale $\mathcal{O}((\log^{12} N)/\delta^3)$ devient $\mathcal{O}(\log^9 N)$ avec une accélération globale de $\Theta((\log^3 N)/\delta^3)$ cubique en la taille des solutions $N^{1/\delta}$. Cependant on remarque que l'application de la méthode Rounding est moins pertinente lorsque l'algorithme $\tilde{L}^1$ est utilisé. En effet, notre analyse permet d'obtenir la complexité $\mathcal{O}(\log^{6+\varepsilon} N)$ au lieu de $\mathcal{O}((\log^{7+\varepsilon} N)/\delta)$ pour tout $\varepsilon > 0$ avec une arithmétique efficace sur les entiers, ce qui apporte une accélération $\Theta((\log N)/\delta)$ qui est linéaire en la taille des solutions $N^{1/\delta}$, cependant la complexité asymptotique reste $\mathcal{O}(\log^{6+\varepsilon} N)$ avec l'emploi de la méthode Rounding. Cela résulte du fait que l'algorithme $\tilde{L}^1$ applique une stratégie similaire consistant à tronquer successivement les éléments de certaines sous-matrices. On peut toutefois ajouter qu'une réelle comparaison des deux approches semble délicate due aux constantes absorbées par le $\mathcal{O}$, une implantation de l'algorithme $\tilde{L}^1$ n'étant pas encore diffusée.

Un point intéressant consiste à remarquer que la méthode Rounding permet de clarifier la complexité asymptotique de l'algorithme de Coppersmith pour le cas de polynômes univariés modulaires. En effet, la dépendance au degré δ n'était jusque-là pas tout à fait claire : par exemple, Coppersmith dans l'article [Cop97] donnait une complexité grandissant exponentiellement en δ, mais il est bien connu qu'il s'agit d'une typo et que la complexité était en réalité polynomiale en δ (voir par exemple [BM05b, Theorem 11]). Ainsi, avec l'utilisation de la méthode Rounding, les complexités obtenues ne dépendent plus du degré δ, mais seulement de la taille du module N.

La deuxième méthode d'accélération que nous proposons est heuristique et s'applique lorsque l'on souhaite effectuer une recherche exhaustive afin d'agrandir la borne supérieure X des solutions retrouvées par l'algorithme de Coppersmith. C'est le cas par exemple si la solution recherchée dépasse la borne théorique réalisée par l'algorithme de Coppersmith $X \leqslant N^{1/\delta}$. Mais c'est également et d'abord le cas si l'on souhaite déjà réellement atteindre la borne $N^{1/\delta}$ car en pratique cette borne ne peut pas être atteinte en appliquant purement la méthode de Coppersmith puisque cela nécessiterait l'emploi de paramètres impraticables par les machines de calcul actuelles. Plus précisément, les éléments de la matrice à réduire seraient de taille $\log^2 N$ bits, et la matrice de dimension

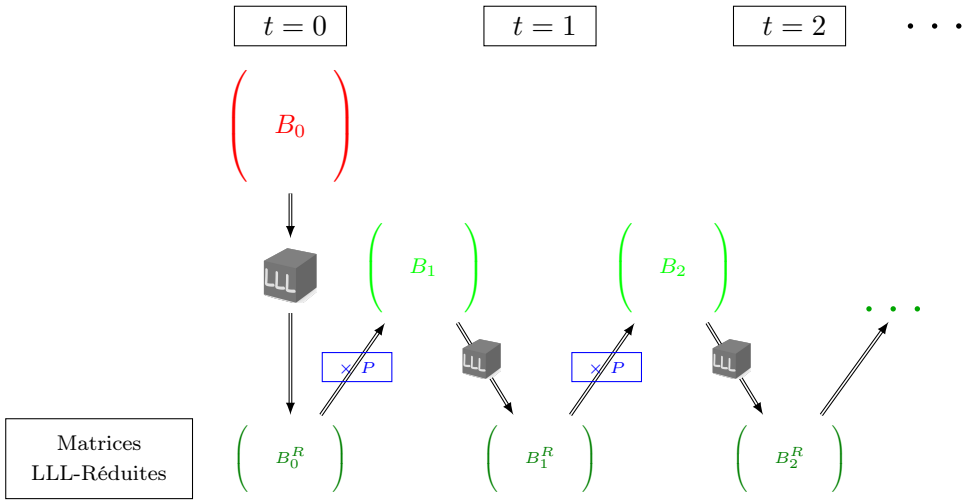
$\log N$; ainsi pour un module N de 2048 bits, la matrice contiendrait $2048^2/2$ soit plus de 2 millions d'éléments (car la matrice est carrée et triangulaire), et la taille de chaque élément serait de plus de 4 millions de bits, ce qui signifie que l'on aurait à réduire une matrice de plus d'un tera-octet.

Ainsi, pour atteindre la borne $N^{1/\delta}$, une recherche exhaustive est préconisée et s'avère considérablement plus efficace. Celle-ci consiste à appliquer l'algorithme de Coppersmith avec le même module N mais sur différents polynômes qui sont tous « décalés » par rapport au pôlynôme initial $f(x)$: $f_t(x) = f(X \cdot t + x)$ où t varie et est tel que $0 \leqslant t < N^{1/\delta}/X$. Dans cette thèse, nous montrons que ce « décalage » permet d'exhiber des relations entre les matrices à LLL-réduire, et que ces relations peuvent être exploitées en pratique. En effet, au lieu d'appliquer des réductions-LLL de manière indépendante, il est possible de les chaîner. Plus précisément, si $B_0, B_1, \ldots, B_n$ sont les différentes matrices à réduire, où B_t est la matrice de Coppersmith correspondant au polynôme $f_t(x)$, notre méthode consiste à effectuer une première réduction-LLL coûteuse de la matrice B_0, ce qui donne la matrice B_0^R. Ensuite, au lieu de réduire B_1, réduction qui serait tout aussi coûteuse que celle de B_0, l'on réduit la matrice $B_0^R \cdot P$ où P est une matrice bien choisie (en l'occurrence, il s'agit de la célèbre matrice de Pascal). Du fait que la matrice $B_0^R \cdot P$ soit le produit d'une matrice réduite B_0^R par une matrice contenant des coefficients relativement petits, on peut s'attendre à ce qu'elle soit déjà presque réduite. Par conséquent, sa réduction-LLL devrait être peu coûteuse.

Ce procédé de chaînage (*Chaining* en anglais) peut ensuite être itéré pour réduire considérablement le temps global de la recherche exhaustive comme illustré en Figure 2. Bien que cette accélération soit conséquente en pratique, elle reste néanmoins heuristique comme nous l'avons précisé plus haut.

Il est judicieux de mentionner que les deux méthodes Rounding et Chaining sont des techniques ayant déjà été auparavant utilisées dans le domaine de la réduction de réseau. En effet pour ce qui est de la technique Rounding, elle a été utilisée par Buchmann [Buc94] pour estimer de manière rigoureuse dans quel cas un calcul avec des réseaux sur les réels pouvait être effectué alternativement à l'aide de réseaux sur les entiers. Comme nous l'avons précisé précédemment, l'algorithme $\tilde{L}^1$ [NSV11] est également basé sur cette stratégie de Rounding. La méthode Chaining quant à elle, a par exemple été utilisée dans le contexte MIMO [NJD11] (avec une technique et analyse toutefois différente de la nôtre). Cependant, malgré ces premiers résultats, les travaux présentés dans cette thèse proposent la première amélioration connue de l'algorithme de Coppersmith. Enfin, plus récemment, un article de Saruchi, Morel, Stehlé et Villard [SMSV14] traite le cas de la technique Rounding sur des matrices plus générales que celles spécifiques à la méthode de Coppersmith. Les bornes qu'ils obtiennent appliquées au cas des matrices de Coppersmith sont très proches de celles que l'on fournit dans notre étude : elles sont légèrement moins avantageuses car leur analyse prend en compte la réduction-LLL de l'ensemble de la matrice, alors que notre approche ne requiert que des bornes sur le premier vecteur de la base réduite.

FIGURE 2 – Nouveau schéma *Chaining* de recherche exhaustive au sein de la méthode de Coppersmith. Une première matrice B_0 est LLL-réduite, puis les matrices sont chaînées par l'application de P et successivement LLL-réduites.



Enfin, nous montrons que les deux méthodes d'accélération peuvent être combinées. Dans ce cas, en pratique, le nouvel algorithme s'exécute des centaines de fois plus rapidement pour des paramètres typiques. Par exemple, si l'on considère le polynôme $f(x) = x^2 + ax + b \equiv 0 \mod N$ de degré $\delta = 2$, où la borne théorique sur les petites solutions est $X \leqslant N^{1/2}$, l'application des méthodes Rounding et Chaining (pour atteindre la borne théorique $N^{1/2}$) permet une accélération de quelques dizaines à milliers de fois suivant les modules, comme illustré en Table 4.

TABLE 4 – Temps global de recherche exhaustive avec l'utilisation de la méthode originale et de la nouvelle méthode (Rounding et Chaining combinés) pour des tailles de N de 512, 1024, 1536 et 2048 bits.

	$\lceil \log_2(N) \rceil = 512$	$\lceil \log_2(N) \rceil = 1024$	$\lceil \log_2(N) \rceil = 1536$	$\lceil \log_2(N) \rceil = 2048$
Méthode Originale	47 minutes	13.1 jours	108.5 jours	7.9 années
Nouvelle Méthode	52 secondes	1.2 heures	5.2 heures	2.6 jours
Accélération	54	262	502	1109

Ainsi, il est pertinent de noter que pour des paramètres typiques, le fait d'atteindre (voire de dépasser !) la borne théorique de Coppersmith, borne qui était jusqu'alors souvent quasiment inaccessible en pratique, peut devenir grâce à ces méthodes une chose tout à fait envisageable (les temps passant parfois de plusieurs années à quelques jours).

Perspectives

Dans cette thèse, la méthode Rounding est appliquée à l'algorithme de Coppersmith pour le cas de polynômes univariés modulaires. Aussi, il serait intéressant d'élargir son emploi à d'autres polynômes, algorithmes, voire à d'autres contextes, avec éventuellement des adaptations à proposer. Ainsi, par exemple, il existe de nombreuses variantes de l'algorithme de Coppersmith sur lesquelles l'obtention d'un gain significatif par l'application de cette méthode ne semble pas tout aussi apparent. En effet, on note par exemple la généralisation au PGCD [HG01, BDHG99] qui consiste à trouver les petites solutions x_0 telles que $f(x_0) \equiv 0 \bmod N$ où $\mathrm{PGCD}(f(x_0), N) \geq N^\alpha$ et $0 < \alpha \leq 1$, mais aussi la méthode de Coppersmith pour le cas de polynômes bivariés sur les entiers, ou encore les généralisations aux polynômes multivariés (modulaires ou sur les entiers). Dans ces variantes, les matrices sur lesquelles la réduction-LLL est effectuée n'ont plus tout à fait la propriété d'équilibre de l'ensemble des éléments de la diagonale, ce qui rend l'application de la méthode Rounding moins directe, ou du moins nécessitant une adaptation. Cela reste donc un problème ouvert intéressant que d'obtenir une accélération significative sur ces différentes variantes.

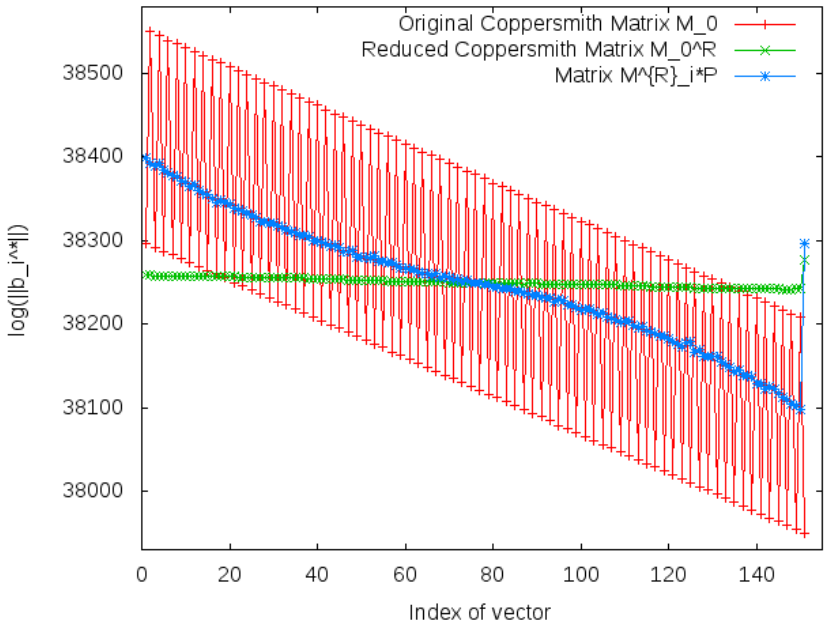
Par ailleurs, s'il est vrai que les applications de la méthode Rounding ici traitées concernent le cas d'anneaux Euclidiens, cette méthode semble s'appliquer tout aussi bien à d'autres types de structures, tels que les anneaux de polynômes, les corps de nombres ou corps de fonctions. On note par exemple l'application à l'algorithme Guruswami-Sudan pour le décodage des codes de Reed-Solomon et sa variante améliorée [CH11b] à laquelle la méthode Rounding pourrait s'adapter naturellement. Ainsi il serait intéressant d'y apporter une analyse plus approfondie.

La méthode Chaining a par ailleurs elle aussi été appliquée à l'algorithme de Coppersmith pour le cas de polynômes univariés modulaires. Toutefois, il serait intéressant d'analyser l'efficacité de son application aux autres variantes. En effet, si l'on observe par exemple le cas de polynômes multivariés creux, l'application de la méthode Chaining pourrait nécessiter l'ajout d'un nombre non négligeable de colonnes (et de lignes) dans la matrice de Coppersmith, ce qui pourrait alourdir les calculs. En effet, afin que la matrice de passage conserve des éléments de petite taille, il est nécessaire que l'ensemble des monômes apparaissant au cours du chaînage figurent également dans la matrice de Coppersmith initiale même s'ils s'avèrent superflus au départ. Ainsi, analyser la pertinence de la méthode Chaining dans ces cas-là pour éventuellement y apporter certaines adaptations, reste un point d'étude ouvert.

Par ailleurs, comme nous l'avons précisé, l'accélération due à l'application de la méthode Chaining sur l'algorithme de Coppersmith pour le cas de polynômes univariés modulaires est heuristique. En effet, une analyse de la taille des éléments des matrices réduites lors du déroulement de la méthode Chaining ne permet pas de mettre en avant une amélioration de la complexité asymptotique des réductions-LLL. Toutefois, le gain est considérable en pratique. Cela s'explique intuitivement par le fait que les matrices

à réduire durant le chaînage sont le produit d'une matrice déjà réduite par une matrice contenant des coefficients relativement petits (matrice de Pascal), et que par suite, le travail à effectuer par l'algorithme de réduction-LLL devrait être moins conséquent. Cela se confirme par une analyse expérimentale illustrée en Figure 3 de la taille des coefficients de Gram-Schmidt de ces matrices (courbe bleue) qui restent relativement proches de ceux de la matrice réduite (courbe verte horizontale) en comparaison de ceux de la matrice de Coppersmith originale (courbe rouge en dents de scie). Cependant une meilleure compréhension théorique de la forme de la courbe bleue reste un point ouvert qui permettrait d'écarter l'heuristique de la méthode et de mieux quantifier le gain obtenu.

FIGURE 3 – Taille des coefficients de Gram-Schmidt $\log_2(||\mathbf{b_i}^\star||)$ de chaque vecteur pour 3 matrices : la courbe rouge (dents de scie) est associée à la matrice originale de Coppersmith (dimension 151) pour un polynôme univarié modulaire de degré $\delta = 2$ et $\lceil \log N \rceil = 512$; la courbe verte (horizontale) représente la matrice LLL-réduite correspondante ; la courbe bleue est associée aux matrices intermédiaires à réduire au sein de la méthode Chaining.



Enfin, si cette thèse propose des attaques algébriques et physiques, elle met aussi en avant le lien existant entre ces deux approches, en particulier le fait qu'elles s'avèrent parfois très complémentaires. En outre, si l'on prend également en compte les accélérations proposées pour trouver les petites racines de polynômes, accélérations permettant en pratique de repousser les bornes existantes, il pourrait être intéressant d'analyser certaines attaques physiques publiées dans le domaine, afin de les améliorer voire de trouver de nouveaux chemins d'attaque grâce à l'utilisation de méthodes algébriques, et en particulier de méthodes basées sur les réseaux.

Part I

State Of The Art

Chapter 1

RSA on Embedded Devices and Physical Attacks

Contents

Since its introduction in 1978, the RSA cryptosystem has become one of the most used public-key cryptosystems. RSA can be used as a ciphering tool, for example to cipher symmetric keys, but also as a signing tool, namely for numerical signatures, to guaranty the integrity of a document and to authenticate its author. The process of both schemes (ciphering and signing) is similar, even so we rather consider the signing scheme in this chapter since it is by far the most used in practice. Furthermore, it is well known that some computations in RSA can be speeded up by using the famous Chinese Remainder Theorem (CRT). Thusly, we commonly speak about two modes of implementation: the Standard mode, and the CRT mode. In embedded systems like smart cards, most RSA implementations use the CRT mode which yields an expected speed-up factor of about four [CQ82]. However, when using this CRT mode on embedded systems, the implementation becomes more vulnerable to *Fault Attacks*, as depicted in [BDL97]. More generally, because embedded systems are left to the consumer's hands, it can be vulnerable to what is called *Side-channel analysis*.

Hence, in this chapter, we recall the RSA Signature accordingly to both modes, the Standard mode and the CRT mode. Then, we recall the most common side-channels and

we describe some basic physical attacks on such an algorithm. Eventually, we highlight that physical attacks can sometimes be combined with *lattice-based techniques* in the sense that physical attacks can possibly allow to recover part of the secret and in some instances, the use of lattice-based techniques can be decisive to recover the whole secret.

1.1 RSA Cryptosystem on Embedded Devices

As previously said, RSA is one of the most used public-key cryptosystems. In practice, it is especially employed in the single framework of electronic signature schemes [RSA78]. In the following we recall how to compute the RSA signature in the Standard mode and the CRT mode.

1.1.1 RSA Signature in Standard mode

If a user wants to sign a message, the following steps should be performed:

1. Creation of the keys:
   — Choose two distinct large prime numbers p and q.
   — Compute their product: $N = p \times q$.
   — Compute Euler's totient function $\varphi(N) = (p-1)(q-1)$.
   — Choose an integer e which is prime with $\varphi(N)$.
   — Compute d such that $ed = 1 \bmod \varphi(N)$.
2. Distribution of the keys:
   — The triplet (p, q, d) is the private key of the user. It is kept secret and used to sign a message.
   — The pair (N, e) is the public key of the user. It will be used by his correspondent to verify the signature of the message.
3. Sending of the signed message:
   — The user who wants to sign a message $m \in \mathbb{Z}_N$ computes the signature $S = m^d \bmod N$ and sends to the correspondent the pair (S, m).
4. Verification of the signature:
   — To verify the signature, the correspondent computes $S^e \bmod N$ by using the public key (N, e) of the user, and checks if the corresponding result is equal to m. Indeed, according to Euler's Theorem, one has $S^e \bmod N = m^{de} \bmod N = m \bmod N$.

1.1.2 RSA Signature in CRT mode

In the CRT mode, most steps are identical to the ones in the Standard mode. Indeed, Steps 1, 2 and 4 remain alike. However, Step 3 which is the signature computation step, is performed differently, by using the Chinese Remainder Theorem (CRT). Thus, instead of directly computing $m^d \bmod N$, one separates the computation into two parts, and recombines both results. More precisely, one performs two exponentiations

$$S_p = m^{d_p} \bmod p \qquad \text{and} \qquad S_q = m^{d_q} \bmod q \quad ,$$

where

$$d_p = d \bmod p - 1 \qquad \text{and} \qquad d_q = d \bmod q - 1 \quad .$$

The signature is then obtained by recombining S_p and S_q, which is usually done by using Garner's formula [Gar59]:

$$S = CRT(S_p, S_q) = S_q + q(i_q(S_p - S_q) \bmod p) \ , \tag{1.1}$$

where $i_q = q^{-1} \bmod p$.

Remark 1. *Note that in both modes, in order to avoid some attacks (reordering of the message, existential forgery, etc.) and to deal with long messages, the message is not signed directly but is previously hashed using a hash function h. Then, the quantity $h(m)$ is signed.*

1.2 Physical Attacks

Physical attacks consists in observing and manipulating the data processed by the embedded system in order to extract some secret information. Depending on whether the device is altered or simply observed, the attack is said to be invasive or non-invasive.

1.2.1 Non-invasive Attacks

In the context of non-invasive attacks, the manipulated data is not modified but only observed and analyzed. Namely, the device is not permanently altered and no evidence of an attack is left behind. Non-invasive attacks typically exploit what is called *side-channels*.

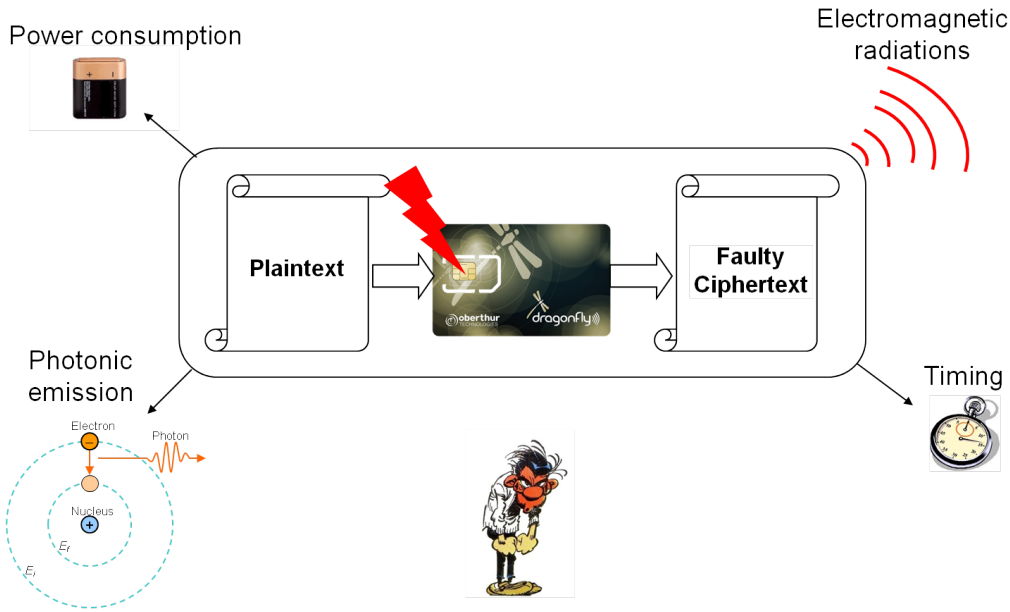
The most famous side-channels:

Side-Channel analysis is a cryptanalytic technique which exploits information leaking from the physical implementation of cryptosystems. It takes advantage of leakages arisen during the execution of an algorithm, in order to extract secret information from a cryptographic device. One of the best examples of cryptographic devices which are subject to side-channel analysis are embedded devices like smart cards.

Side-Channel analysis has been introduced by the publication of the so-called timing attacks in 1996 [Koc96]. By that time, execution timing was the most exploited side-channel. However, other parameters like the power consumption and electromagnetic radiations rapidly became the most efficient side-channels to attack embedded cryptography [KJJ99, QS00].

Besides, the four most famous side-channels which can bring exploitable information to an attacker are the power consumption of the card, the electromagnetic radiations and the photonic emission diffused by the card, and the timing of a computation, as depicted in Figure 1.1.

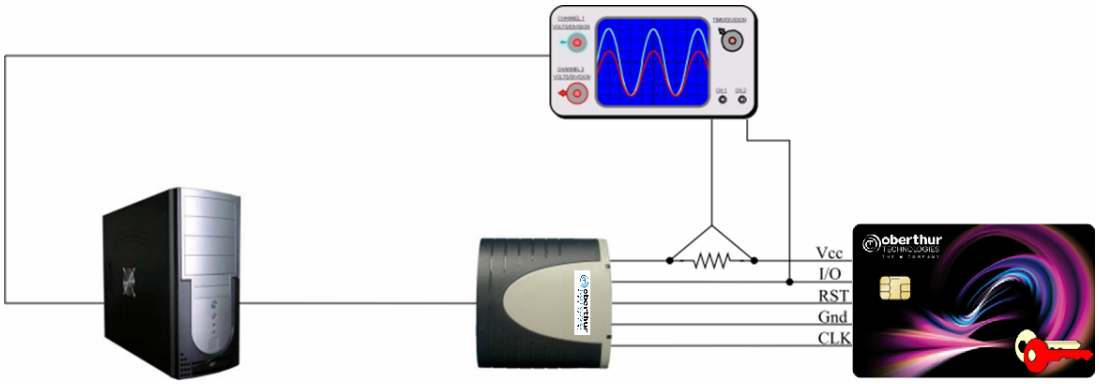
Figure 1.1: The most famous side-channels



Side-channel analysis exploits the dependency between the manipulated data or the executed instruction and the side-channel leakages which can be monitored during the algorithm execution. In the following, we mostly consider the power consumption as a side-channel, but other side-channels like electromagnetic radiations can equally be exploited to mount the same type of attacks.

A typical equipment allowing the exploitation of power consumption involves a computer, a smart card reader, an oscilloscope and a card: the computer sends commands of cryptographic algorithms execution to the card via the smart card reader, and the oscilloscope connected to a small resistor in series with the power supply measures the power consumption as illustrated in Figure 1.2.

Figure 1.2: Typical equipment for a side channel analysis using the power consumption.



The side-channel analysis performed on the obtained information is different whether one considers one measurement only or several measurements. Hence, one draws a distinction between *Simple Power Analysis* and *Differential Power Analysis*, as explained in the sequel.

Simple Power Analysis (SPA)

Simple Power Analysis (SPA) consists in analyzing the variations and peaks of one curve of power consumption during the execution of the cryptographic algorithm, in order to discover some secret information, like the ciphering key [KJJ99]. Historically, this type of analysis was discovered by exploiting the power consumption as a side-channel, and this explains the " P" in the acronym SPA. In fact, other side-channels can be similarly used to perform an SPA: the attacks are performed identically, the only difference is the way the leakage measurement is obtained. For example, when electromagnetic radiations are considered, one can rather employ the acronym SEMA. Yet, in the sequel and as it is frequently done, the acronym SPA will be employed to specify the analysis of all types of side-channels requiring one measure only.

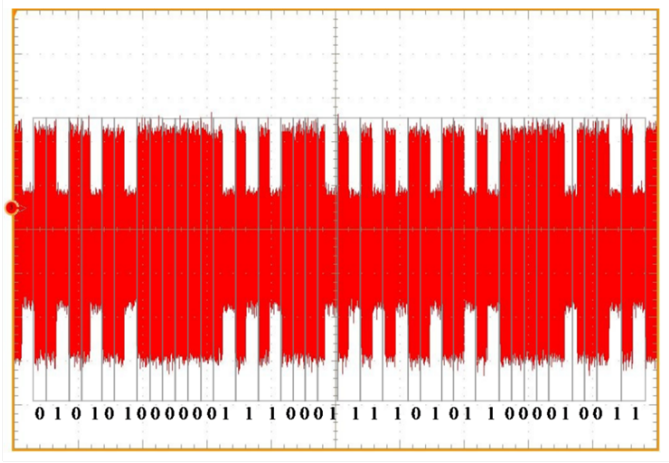
An example of SPA on modular exponentiation during RSA signature: One of the most popular SPA attack applies on the modular exponentiation $m^d \bmod N$ which is performed during the RSA signature in case of a straightforward *Square-and-Multiply* implementation to perform the exponentiation. Namely, the Square-and-Multiply algorithm consists in writing the private key d in a binary basis and for each bit of d, a computation depending on whether the bit is 0 or 1 is performed: if the current bit is 0, then a simple squaring is done, if the bit is 1, one executes a squaring followed by a multiplication. When the squaring and multiplication operations have different patterns in the corresponding side-channel leakages, it is easy to differentiate them [KJJ99]. Hence, the secret exponent d can be directly extracted from one measurement as depicted in Figure 1.3. Thusly, the factorization of N can be easily recovered from the secret key d.

In the literature, a common countermeasure consists in using a so-called *regular* algorithm which performs the same operation whatever the exponent bit value such as the *Square-Always* or *Montgomery ladder* algorithms [JY02, CFG+11].

Differential Power Analysis (DPA)

In contrast to a Simple Power Analysis, where a direct relation linking the secret and the side-channel information could be drawn, a Differential Power Analysis (DPA) exploits side-channel information which is less explicitly linked to the secret. Thus, in a DPA, the secret information is brought out by using a large number of measurements extracted from many executions that use the same secret key. Note that as specified before, one can use measurements obtained from other side channels than power consumption, like electromagnetic radiations; the way the attack is subsequently performed remains

Figure 1.3: Electromagnetic radiations measured during the execution of a modular exponentiation performed with the Square-and-Multiply method.



identical. This type of attack applies a statistical treatment on the curves to recover information on the manipulated values. They consist in identifying some intermediate variables which are manipulated during the execution of the algorithm, and which depend on small parts of the secret key and on some known values. Such variables are said to be *sensitive*. Thus, if one can recover a sensitive value, then one retrieves the corresponding part of the secret key, and vice-versa.

Hence, if a sensitive value is manipulated during the execution of the algorithm, the principle of the attack consists in performing an exhaustive search on the small secret part by making all possible guesses on this secret part. Thusly, on one side, one executes several times the algorithm with different known inputs and one saves the corresponding leakage measurements, and on the other side, for each guess, one predicts the sensitive variables associated to the known inputs (which can be done because the sensitive variables depend on this guess and on the known inputs). For the correct guess, a statistical relation is observed between the predicted values and the leakage measurements, and for all other guesses it is expected that no noticeable relation will be observed.

A classical statistic tool used to perform such a statistical treatment, is the Pearson correlation coefficient:

$$\rho_k = \frac{\mathrm{cov}(\mathcal{L}, H)}{\sigma_{\mathcal{L}} \sigma_H}, \tag{1.2}$$

where $\mathcal{L}$ is the set of curves and H depends on a known value m and on a guess of a small part of a secret k. Such an attack is called Correlation Power Analysis (CPA), as depicted in [EBCO04].

An example of CPA on CRT-recombination during RSA signature: In the literature, many different CPAs have been published to attack the RSA cryptosystem. For instance in the RSA signature, if the CRT mode is implemented then an attacker can mount a CPA to recover the private parameter q during the CRT-recombination specified in Relation (1.1). This CPA attack is depicted in [AFV07] and its principle is recalled in the following.

We use the same notations as before, namely we denote by S the value $m^d \mod N$ where $N = pq$, and i_q is $q^{-1} \mod p$. We also have $S_p = m^{d_p} \mod p$ and $S_q = m^{d_q} \mod q$.

Since by Garner's formula one has $S = S_q + q(i_q(S_p - S_q) \bmod p)$, therefore we deduce that

$$\left\lfloor \frac{S}{q} \right\rfloor = (i_q(S_p - S_q) \bmod p) + \left\lfloor \frac{S_q}{q} \right\rfloor \quad .$$

Furthermore, because by definition, we have $S_q < q$, we deduce that $\left\lfloor \frac{S_q}{q} \right\rfloor = 0$. Therefore, one gets the relation:

$$\left\lfloor \frac{S}{q} \right\rfloor = i_q(S_p - S_q) \bmod p\,.$$

The value $i_q(S_p - S_q) \bmod p$ is manipulated during the CRT-recombination computation. Since this manipulated value is equal to $\lfloor S/q \rfloor$, it depends on a secret value q and on a known value S. Therefore, this is a sensitive value on which one can perform a CPA. More precisely, the value $i_q(S_p - S_q) \bmod p$ is manipulated part by part: typically each byte is processed sequentially. Therefore one can consider each byte of the secret q independently.

Thus, for each byte of q, one makes a guess on the value of this byte and one wants to confirm this guess. To this aim, one launches on the device t executions of the algorithm with input messages $m_1, m_2, \ldots, m_t$, and collect the curves $C_1, C_2, \ldots, C_t$ corresponding to the power consumption of these executions. On the other hand, for all input messages $m_1, m_2, \ldots, m_t$, one predicts the value of the corresponding byte of the sensitive variables $\lfloor S/q \rfloor$ associated to the guess of the byte of q and to the known signatures S.

Then the Pearson correlation coefficient is computed by using the obtained curves and the predictions (cf. Relation (1.2)). If the guess was correct, then a statistical relation is observed between the predicted values and the leakage measurements. In the case where no statistical relation is observed, then one tries other guesses, until recovering the correct byte of the secret q. Hence, an attacker can obtain the whole secret q by performing a CPA for each of its bytes.

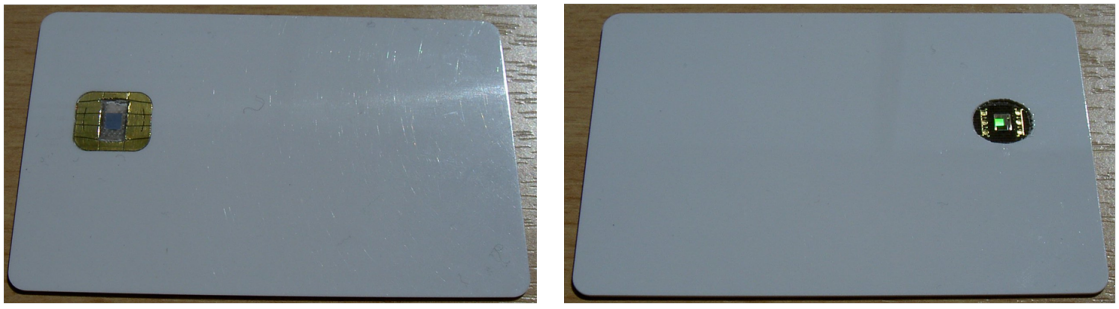
Eventually, classical countermeasures to resist CPA consist in randomizing the modulus N, the message m and the exponent d as depicted in [AFV07] in order to introduce some unknown data that changes at each execution, which makes the attack impossible.

Thus, we have recalled some basics about non-invasive attacks involving side-channels, and we refer the interested reader to the book [MOP07] for more details. In the following, we describe another type of attack which affects the physical integrity of the card and which is said to be *invasive*.

1.2.2 Invasive Attacks

Invasive attacks typically start by the depackaging of the cryptographic device as depicted in Figure 1.4. They allow to introduce modifications on the embedded system, ranging from a simple alteration of the processed data, to an irreversible damage of the material.

Figure 1.4: Smart card depackaging on the back side (on the left) and on the front side (on the right).



A typical example of invasive attacks are *Fault Attacks* which consist in disrupting the cryptographic computation, for instance by injecting light pulses (see Figures 1.5 and 1.6), so that it produces erroneous results.

Figure 1.5: A Diode Laser Station (picture taken from the Riscure Inspector Data Sheet).

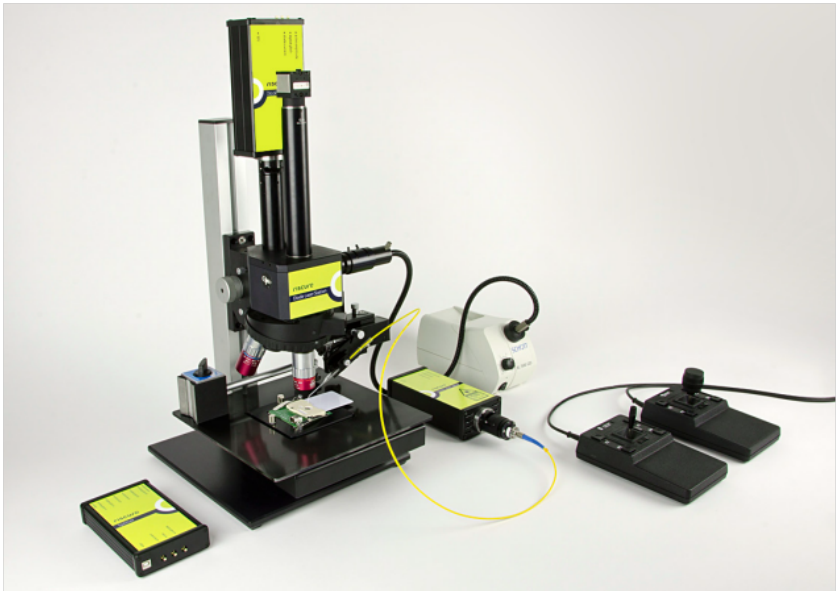
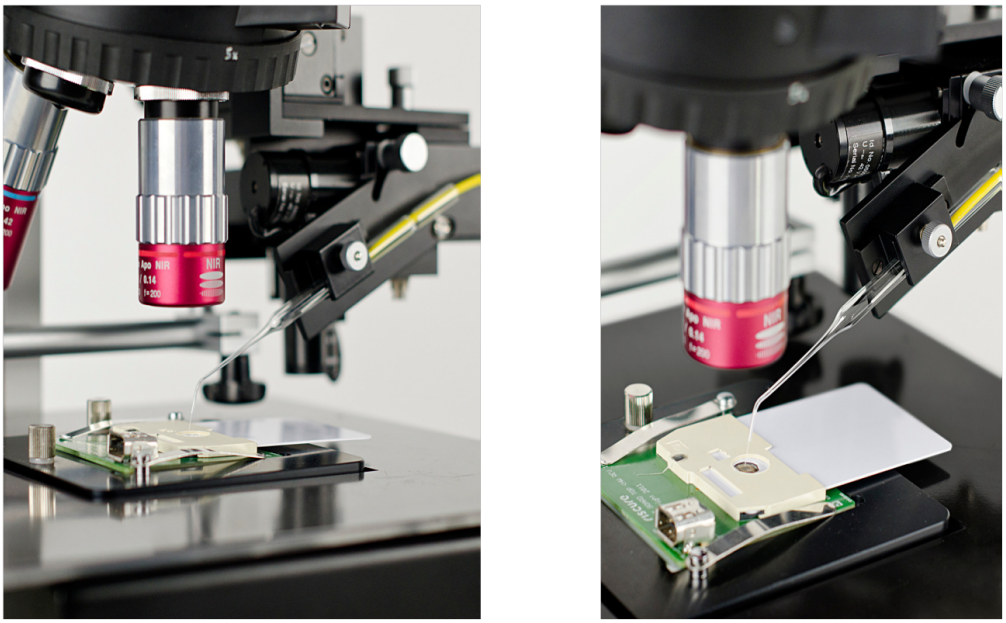


Figure 1.6: Multi-area Diode Laser System: an additional spot is added by a laser beam through a glass fiber (pictures taken from the Riscure Inspector Data Sheet).



Fault Models:

In the literature, four different fault models are generally considered to define the attacker's capabilities [BOS03]:

— the *random* fault model: the bits are changed to a uniformly distributed random value;
— the *bit-flip* fault model: in that case, affected bits are flipped to their complementary value;
— the *stuck-at* fault model: the fault sets the bits to 0 or to 1, depending on the underlying hardware;
— the *unknown constant* fault model: the fault always sets the bits to the same unknown value.

Moreover, these faults do not necessarily modify a whole temporary result. Indeed, it is generally considered that the number of bits affected by the fault is linked to the CPU word-size which is generally 8, 16 or 32 bits.

An Example of Fault Attack on RSA-CRT Signature:

RSA has been the first cryptosystem to succumb to Fault attacks. In the following, we describe the so-called Bellcore attack [BDL97] when the CRT mode is implemented. We recall that in such a mode, the computation of the signature S is performed modulo

p and q separately, which gives S_p and S_q, and both are then recombined to get the final signature $S \bmod N$.

Assume that a fault is injected during the computation of S_p (and not during the one of S_q) leading to a faulty signature $\widetilde{S}$. By definition we have $S \equiv S_p \bmod p$ and $S \equiv S_q \bmod q$, hence one notices that

$$\begin{cases} \widetilde{S} & \equiv & S & \bmod q \\ \widetilde{S} & \not\equiv & S & \bmod p \end{cases}$$

because the computation of S_q was not disturbed and that a fault was injected during S_p computation.

Thusly, when it is possible to sign the message twice: one time correctly, and one time by inducing a fault, one gets a correct signature S and a faulty signature $\widetilde{S}$. The knowledge of both signatures allows to easily recover the secret parameter q by computing the *gcd* of $S - \widetilde{S}$ and N. Indeed, since $S - \widetilde{S} \equiv 0 \bmod q$ and $S - \widetilde{S} \not\equiv 0 \bmod p$ we have that

$$\gcd(S - \widetilde{S}, N) = q \quad .$$

The other factor $p = N/q$ is then straightforwardly deduced.

When it is not possible to sign the same message twice and if the message is known to the attacker, a variant of this attack consists in computing the *gcd* of $\widetilde{S}^e - m$ and N to obtain the secret value q [Len96]. Indeed, since by definition one has $\widetilde{S}^e - m \equiv 0 \bmod N$, one deduces that $\widetilde{S}^e - m \equiv 0 \bmod q$ and $\widetilde{S}^e - m \not\equiv 0 \bmod p$, therefore we have that

$$\gcd(\widetilde{S}^e - m, N) = q \quad .$$

More generally, the effect of fault injections on CRT-RSA is not limited to the disturbance of S_p or S_q. Indeed, a fault injected in any part of the key parameters (i.e. p, q, d_p, d_q or i_q), in the message m at the beginning of either S_p or S_q computation, or even during the CRT-recombination can lead to a useful faulty signature.

The most natural way to counteract fault injection on RSA-type signature is to check the correctness of the signature S before outputting it [BDL97]. More precisely, the signature is returned if and only if $S^e \bmod N = m$. Moreover, such a method requires very little overhead since the public exponent e is usually small in practice (typically 3, 17 or $2^{16} + 1$).

Other methods getting rid of e have also been proposed but they do not offer the same level of security and are generally slower than the public verification [Gir06, Vig08, Riv09].

1.3 Lattices in Physical Attacks

If physical attacks sometimes allow the retrieval of the whole secret, in many cases, only part of the secret is revealed to the attacker. In those cases, lattice-based techniques (see Chapter 2 and Chapter 3 for further details) turn out to be very useful and complementary to physical attacks. Namely, the information brought by physical attacks can account for a substantial input which allows to make possible the discovery of the entire secret thanks to lattice-based techniques. In the following, we reconsider the three examples of physical attacks recalled in this chapter and we provide some extensions based on lattices.

1.3.1 SPA on Modular Exponentiation and Lattices:

It is well-known that the use of a small private key d in RSA, namely an exponent $d < N^{0.292}$ should be prohibited. This result has emerged from a lattice-based technique due to Coppersmith [BD00] which allows to recover the entire secret d if such a condition is fulfilled. Alternatively, if d is large and if one knows a portion of the bits of d, then the same method can be applied to recover the whole secret d. Namely, in [BDF98], the authors show that for low public exponent e, a quarter of the bits of the private key d is sufficient to recover the entire private key. Similar results (though not as strong) are obtained for larger values of e. Furthermore, they also deal with the case where the known bits are the most or the least significant ones, or when a part in both sides is known. Thusly, such methods can be employed in the case where part of the bits of d was recovered thanks to a side-channel analysis such as the SPA described in Section 1.2. The other part of the secret d can indeed be straightforwardly retrieved thanks to these lattice-based techniques.

1.3.2 CPA on CRT-Recombination and Lattices:

It has been shown in [Cop96a] that the knowledge of half of the bits of prime q is sufficient to recover the rest of q by using lattice-based techniques, namely by using Coppersmith's method (which is recalled in Chapter 3). Thusly, such a method can be very useful in the case where half of the bits of q was recovered thanks to a side-channel attack such as the CPA described in Section 1.2. The other part of the secret q is indeed directly retrieved thanks to lattice techniques.

1.3.3 Fault Attacks on RSA and Lattices:

The fault attack on RSA Signature described in Section 1.2 is noteworthy in the sense that a single fault allows to directly recover the entire secret. However, many published fault attacks with a different context of application, allow to recover part of the secret, sometimes with the necessity of many fault injections. Then the use of lattice-based techniques is decisive to recover the whole secret.

One can mention for example randomized RSA encoding/signature schemes (*e.g.* the randomized version of ISO/IEC9796-2 Standard) which were usually considered to be

resistant to traditional fault attacks since a part of the message is unknown to the attacker and varies for each signature computation. However this common assumption was mitigated regarding the works of [CJK+09] and [CNT10] which defeat two randomised RSA encoding schemes. These attacks use lattice-based techniques, and more precisely Coppersmith's method to solve a bivariate polynomial equation whose coefficients are built thanks to the generated faulty signatures.

One can also mention the attack published in [EBNNT11], where the authors take advantage of the disturbance of the public modulus. The generated faulty signatures allow them to build a lattice, which in turn leads to factorize the public modulus.

Chapter 2

Lattice Reduction

Contents

In this chapter, we recall some important definitions and theorems concerning lattices which will be useful for the scope of the manuscript. More precisely, in Section 2.1 we recall some basics on Euclidean lattices. In Section 2.2 we highlight the problem of lattice reduction, together with the weaker notion of LLL-reduction and we depict the most famous existing polynomial-time algorithms that ensure LLL-reduction. Eventually, in Section 2.3, we give an application of LLL-reduction algorithms, related to the problem of simultaneous Diophantine approximation. More generally, we refer the interested reader to [Cas, Coh95] for a more detailed introduction into the theory of lattices.

2.1 Euclidean Lattices

2.1.1 Some Basic Definitions

A lattice is a regular infinite arrangement of points in space. Mathematically, it is a discrete additive subgroup of $\mathbb{R}^n$. More precisely, we acknowledge the following definition.

Definition 2. *Let* $\mathbf{u_1}, \ldots, \mathbf{u_n} \in \mathbb{R}^m$ *be linearly independent vectors with* $n \leqslant m$.

— *A lattice* $\mathcal{L}$ *spanned by* $\{\mathbf{u_1}, \ldots, \mathbf{u_n}\}$ *is the set of all integer linear combinations of* $\mathbf{u_1}, \ldots, \mathbf{u_n}$*:*

$$\mathcal{L} \quad = \quad \{\mathbf{u} \in \mathbb{R}^n \;\big|\; \mathbf{u} = \sum_{i=1}^{n} \lambda_i \mathbf{u_i} \;\; \textit{with} \;\; \lambda_i \in \mathbb{Z}\} \quad .$$

— *The set* $\mathcal{B} = \{\mathbf{u_1}, \ldots, \mathbf{u_n}\}$ *is called a basis of the lattice* $\mathcal{L}$.

— *If* $m = n$*, then the lattice is called a full rank lattice.*

— *The dimension of the lattice* $\dim(\mathcal{L}) = n$ *is the number of vectors in the basis* $\mathcal{B}$.

In this thesis, and as it is often the case in cryptography, we only consider *integer lattices*, that is lattices for which the basis vectors $\{\mathbf{u_1}, \ldots, \mathbf{u_n}\}$ belong to $\mathbb{Z}^n$. Furthermore, the considered lattices will only be full rank lattices.

2.1.2 Volume and Determinant

As soon as $\dim(\mathcal{L}) \geqslant 2$, there exist infinitely many bases which allow to represent the lattice $\mathcal{L}$, but all bases contain the same number of elements which is $\dim(\mathcal{L})$. Another property that connects all those bases lies in the fact that they all admit the same *volume*. In other words, the n-dimensional volume of the parallelepiped spanned by any basis of the same lattice is a geometric invariant. As recalled in the sequel, the volume can be defined by the algebraic notion of *determinant*.

A full rank lattice $\mathcal{L}$ spanned by the basis $\mathcal{B} = \{\mathbf{u_1}, \ldots, \mathbf{u_n}\}$ can be represented by a matrix B of dimension $n \times n$ where each row of the matrix B contains the coordinates of one vector in the basis $\mathcal{B}$.

Thus, the following proposition highlights the link between the volume of a lattice $\mathcal{L}$ and the determinant of a basis of $\mathcal{L}$:

Proposition 3. *Let* B *be a matrix representing a basis of the lattice* $\mathcal{L}$*. The volume of the lattice* $\mathcal{L}$ *is defined as follows:*

$$\mathrm{vol}(\mathcal{L}) = \sqrt{\det(BB^T)} \quad .$$

If further, $\mathcal{L}$ *is a full rank lattice, then we have:*

$$\mathrm{vol}(\mathcal{L}) = |\det(B)| \quad .$$

Thus, in the same way that all bases of a lattice admit the same volume, they all have the same determinant. This leads to the following proposition which gives a link between two matrices of the same lattice.

Proposition 4. *Let* B *and* B' *be two matrices representing a full rank lattice* $\mathcal{L}$*. Then, there exists a unimodular matrix* U *with integer coefficients such that*

$$B' = U \times B \quad \textit{and} \quad \det(U) = \pm 1 \quad .$$

Thus, all bases of the lattice $\mathcal{L}$ *have the same determinant.*

If the basis matrix B is triangular, then the lattice determinant is simply the product of the absolute values of the diagonal coefficients of B. But when it is not the case, Hadamard's inequality still gives a useful upper-bound on the determinant as depicted in the following proposition.

Proposition 5. *Let $\mathcal{L}$ be a full rank lattice and $B = (\mathbf{u_1}, \ldots, \mathbf{u_n})$ be a basis of $\mathcal{L}$, Hadamard's inequality gives:*

$$|\det(B)| \leqslant \prod_{i=1}^{n} \|\mathbf{u_i}\| \quad .$$

Eventually, we provide a last property which allows to write the determinant of the lattice in a convenient way. This relation uses the famous notion of Gram-Schmidt orthogonalization that will be recalled in next section.

Proposition 6. *Let $\mathcal{L}$ be a full rank lattice spanned by $B = \{\mathbf{u_1}, \ldots, \mathbf{u_n}\}$ and let $B^\star = \{\mathbf{u}_1^\star, \ldots, \mathbf{u}_n^\star\}$ be the corresponding Gram-Schmidt orthogonal basis. The determinant of $\mathcal{L}$ is*

$$|\det B| = \prod_{i=1}^{n} \|\mathbf{u}_i^\star\| \quad .$$

2.1.3 Shortest Vector Problem and Orthogonality

As mentioned before, there exist infinitely many bases for the same lattice $\mathcal{L}$. However, some of them hold more interesting properties than others: namely, those which contain vectors that are relatively small (with regard to the Euclidean norm) and orthogonal.

Shortest Vector Problem:

Since a lattice is a discrete subgroup, there exists a non-zero vector $\mathbf{v}$ belonging in the lattice, which has a minimal norm. Thereon, Minkowski gave in [Min12] an upper-bound on the norm of the shortest vector, which is depicted in the following theorem.

Theorem 7 (Minkowski)**.** *Let $\mathcal{L}$ be a lattice of dimension n, then it contains a non-zero vector $\mathbf{v}$ such that*

$$\|\mathbf{v}\| \leqslant \sqrt{n} \det(\mathcal{L})^{1/n} \quad .$$

The Shortest Vector Problem, called SVP, is the most famous lattice problem, and it is the following:

Problem 8 (Shortest Vector Problem (SVP))**.** *Given a basis $B = \{\mathbf{u_1}, \ldots, \mathbf{u_n}\}$ of a lattice $\mathcal{L}$, find a shortest non-zero vector $\mathbf{u}$ in the lattice $\mathcal{L}$.*

This problem is known to be NP-hard under randomized reductions, that is, there is no known polynomial time algorithm that solves it [Ajt96]. Note that in dimension 2, the Gauss Reduction Algorithm allows to find a shortest vector of a lattice in polynomial time. Thus, even if the general problem (for any larger dimension) is NP-hard, some polynomial-time algorithms which allow to approximate a shortest vector have been designed, as depicted in next section.

Gram-Schmidt orthogonalization:

As previously said, we are interested in finding bases which contain relatively small and orthogonal vectors. Thus, we recall the process of the Gram-Schmidt Orthogonalization.

Definition 9 (Gram-Schmidt Orthogonalization)**.** *Let $B = \{\mathbf{u_1}, \ldots, \mathbf{u_n}\}$ be an input basis. The Gram-Schmidt orthogonalization process allows to construct an orthogonal basis $B^\star = \{\mathbf{u}_1^\star, \ldots, \mathbf{u}_n^\star\}$ of the same vector subspace as B.*

— *The process works iteratively on the vectors $\mathbf{u_i}$ for $i \leqslant n$ and consists in computing $\mathbf{u}_i^\star$ which is the projection of $\mathbf{u_i}$, orthogonally to the vector subspace generated by the $i-1$ first vectors of B. More precisely, it is done as follows:*

$$\begin{cases} \mathbf{u}_1^\star = \mathbf{u_1} \quad , \\ \mathbf{u}_i^\star = \mathbf{u_i} - \sum_{j<i} \mu_{i,j}\mathbf{u}_j^\star \quad , \quad \text{where} \quad \mu_{i,j} = \frac{\langle \mathbf{u_i}, \mathbf{u}_j^\star \rangle}{\|\mathbf{u}_j^\star\|^2} \quad \text{for} \quad 2 \leqslant i \leqslant n \quad . \end{cases}$$

— *The orthogonalized matrix $B^\star$ verifies $B^\star = M \times B$ where M is the $n \times n$ lower-triangular matrix defined as follows:*

$$\begin{cases} M_{i,j} = -\mu_{i,j} & \text{if} \quad i > j \\ M_{i,j} = 1 & \text{if} \quad i = j \\ M_{i,j} = 0 & \text{if} \quad i < j \end{cases}$$

Namely, in Algorithm 1, we provide the Gram-Schmidt orthogonalization algorithm which, given an input matrix $B = \{\mathbf{u_1}, \ldots, \mathbf{u_n}\}$, outputs a Gram-Schmidt orthogonalized matrix $B^\star = \{\mathbf{u}_1^\star, \ldots, \mathbf{u}_n^\star\}$ and a lower-triangular transformation matrix M such that $B = MB^\star$.

Algorithm 1 Gram-Schmidt Orthogonalization Algorithm

Input: Initial basis of the vector space $B = \{\mathbf{u_1}, \ldots, \mathbf{u_n}\}$.

Output: A Gram-Schmidt Orthogonalized basis $B^\star = \{\mathbf{u}_1^\star, \ldots, \mathbf{u}_n^\star\}$ and a transformation matrix M such that $B = MB^\star$.

1: **for** i from 1 to n **do**
2: $\quad \mathbf{u_i}^\star := \mathbf{u_i}$
3: $\quad$ **for** j from 1 to $i-1$ **do**
4: $\quad\quad \mu_{i,j} := \frac{\langle \mathbf{u_i}, \mathbf{u}_j^\star \rangle}{\|\mathbf{u}_j^\star\|^2}$
5: $\quad\quad \mathbf{u}_i^\star = \mathbf{u}_i^\star - \mu_{i,j}\mathbf{u}_j^\star$.
6: $\quad$ **end for**
7: **end for**
8: Output matrices $B^\star$ and M.

2.2 *LLL*-Reduction

As highlighted in Section 2.1.3, the problem of finding a shortest vector in a lattice is NP-hard. However, Lenstra, Lenstra and Lovász [LLL82] proposed in 1982 a polynomial time algorithm which is able to approximate a shortest vector. Namely, this famous algorithm called *LLL*, produces a reasonably good basis, using a relaxed notion for reduced basis. This notion, called *LLL*-reduction is defined by two conditions, where the first one, known as *size-reduction*, will be used in Chapter 4 and is defined in the following.

2.2.1 Size-Reduced Basis

Definition 10 (size-reduced)**.** *Let $\mathcal{L}$ be a lattice spanned by $B = \{\mathbf{u_1}, \ldots, \mathbf{u_n}\}$. The basis B is size-reduced if the Gram-Schmidt orthogonalization of B satisfies:*

$$|\mu_{i,j}| \leqslant \frac{1}{2} \quad , \qquad \textit{for all} \quad i < j \quad .$$

There is a classical elementary algorithm which size-reduces a matrix basis $B = \{\mathbf{u_1}, \ldots, \mathbf{u_n}\}$ of an integer lattice $L \subseteq \mathbb{Z}^m$, in polynomial time, without ever modifying the Gram-Schmidt vectors $\mathbf{u}_i^\star$, as depicted in Algorithm 2.

Algorithm 2 A Size-Reduction Algorithm

Input: A basis $B = \{\mathbf{u_1}, \ldots, \mathbf{u_n}\}$ of a lattice $\mathcal{L}$.
Output: A size-reduced basis $B = \{\mathbf{u_1}, \ldots, \mathbf{u_n}\}$.

```
1: Compute all the Gram-Schmidt coefficients μ_{i,j} (using Algorithm 1).
2: for i from 2 to n do
3:    for j from i − 1 downto 1 do
4:       u_i := u_i − ⌈μ_{i,j}⌋ u_j.
5:       for k from 1 to j do
6:          μ_{i,k} := μ_{i,k} − ⌈μ_{i,j}⌋ μ_{j,k}
7:       end for
8:    end for
9: end for
10: Output matrix B = {u_1, ..., u_n}.
```

This algorithm is included in the original *LLL* algorithm [LLL82] (e.g. it is the sub-algorithm *RED* in the description of *LLL* in [Coh93]). In the special case that the input basis is (square) lower-triangular, the running-time of this size-reduction algorithm is $\mathcal{O}(n^3 b^2)$ without fast integer arithmetic, and $n^3\tilde{\mathcal{O}}(b)$ using fast-integer arithmetic, where $b = \max_{1 \leq i \leq n} \log \|\mathbf{b}_i\|$.

2.2.2 *LLL*-Reduced Basis

Eventually, in the following, we give the definition of an *LLL*-reduced basis.

Definition 11 (*LLL*-reduced)**.** *Let $\mathcal{L}$ be a lattice spanned by $B = \{\mathbf{u_1}, \ldots, \mathbf{u_n}\}$ and let $B^\star = \{\mathbf{u}_1^\star, \ldots, \mathbf{u}_n^\star\}$ be the corresponding Gram-Schmidt orthogonal basis. The basis B is LLL-reduced with a parameter $\delta \in (1/4, 1]$ if the following two conditions are satisfied:*

$$\begin{cases} |\mu_{i,j}| \leqslant \frac{1}{2} \ , & \text{for all } i < j \quad \text{(size-reduced condition)} \\ \\ \|\mathbf{u}_{i+1}^\star\|^2 \geqslant (\delta - \mu_{i+1,i}^2)\|\mathbf{u}_i^\star\|^2 \ , & \text{for all } i \quad \text{(Lovász' condition)} \end{cases}$$

The two conditions indicated above that should be satisfied for having an *LLL*-reduced basis are meant to control the two sought properties of the basis (small size and orthogonality). Roughly the first condition which is the size-reduced condition, allows to output a basis with relatively small vectors, and the second one, which is the Lovász' condition, ensures that the vectors remain quite orthogonal relative to one another (in fact this second condition is also crucial to ensure that the vectors are short).

2.2.3 A Basic Version of the LLL Algorithm

In Algorithm 3, we provide a basic version of the LLL algorithm which highlights the principle of the LLL-reduction with the two conditions. We refer the reader to [Coh93, NV10] for a more refined version of the LLL algorithm.

Algorithm 3 A Basic Version of the LLL Algorithm

Input: A basis $B = \{\mathbf{u_1}, \ldots, \mathbf{u_n}\}$ of a lattice $\mathcal{L}$ with a factor $\delta = 3/4$.
Output: An LLL-reduced basis $B = \{\mathbf{u_1}, \ldots, \mathbf{u_n}\}$.
1: Size-reduce the matrix $B = \{\mathbf{u_1}, \ldots, \mathbf{u_n}\}$ (using Algorithm 2).
2: **if** there exists an index j which does not satisfy Lovász' condition **then**
3: Swap $\mathbf{u_j}$ and $\mathbf{u_{j+1}}$.
4: Return to Step 1.
5: **end if**
6: Output matrix B.

2.2.4 Bounds of LLL-Reduced Vectors

The upper-bounds of each vectors achieved for an LLL-reduced basis are highlighted in the following theorem (see [SKKO06]).

Theorem 12 (LLL-reduced vectors bounds)**.** *Let $B^R = \{\mathbf{v_1}, \ldots, \mathbf{v_n}\}$ be an LLL-reduced basis of a lattice $\mathcal{L}$. Then we have:*

$$\|\mathbf{v_i}\| \leqslant 2^{\frac{n(n-1)}{4(n-i+1)}} \det(\mathcal{L})^{\frac{1}{n-i+1}} .$$

In particular, the shortest vector of an LLL-reduced basis $B^R = \{\mathbf{v_1}, \ldots, \mathbf{v_n}\}$ is most likely to be the first vector $\mathbf{v_1}$ and satisfies

$$\|\mathbf{v_1}\| \leq 2^{(n-1)/4} \det(\mathcal{L})^{1/n} .$$

Hence, while the theoretical upper-bound of the shortest vector given in Theorem 7 has a factor $\sqrt{n}$ which is polynomial in the dimension, the upper-bound of the shortest vector of an LLL-reduced basis has a factor $2^{(n-1)/4}$ which is exponential in the dimension. However, LLL-reduced bases give a good approximation of the shortest vector and turn out to be extremely useful in loads of fields, and in particular for cryptanalysis. Also note the bounds obtained in practice by usual LLL-type reduction algorithms are frequently better than the theoretical bounds, which makes them even more interesting.

2.2.5 Complexities of the LLL, L^2 and $\tilde{L}^1$ Algorithms

The LLL Algorithm:

As previously said, Lenstra, Lenstra and Lovász [LLL82] proposed in 1982 a polynomial time algorithm called LLL. When this algorithm was discovered, the authors first applied it for the factorization of polynomials over the rationals, a problem that was firmly established at that time as something that could not be solved in polynomial time. Clearly, this algorithm was originally meant to compute an LLL-reduced basis in polynomial time. Thus, the time complexity of LLL is depicted in the following theorem.

Theorem 13 (LLL)**.** *Given an input basis $B = \{\mathbf{u_1}, \ldots, \mathbf{u_n}\}$ of a lattice $\mathcal{L} \in \mathbb{Z}^m$, where $b = \max_{1 \leqslant i \leqslant n}(\log \|\mathbf{u_i}\|)$, the LLL algorithm outputs an LLL-reduced basis in polynomial time in*

$$\mathcal{O}(n^5 m\, b^3) \quad .$$

Thus, the LLL algorithm has a complexity which is cubic in the size of the input coefficients. While this algorithm is very useful when the size of the elements and the dimension are reasonable, it turns out to be totally inefficient when they become very large, as it is frequently encountered in cryptology.

In order to deal with this problem, many floating-point versions of the LLL algorithm have been proposed, but they usually lead to instability problems, or worse, it may happen that they do not output an LLL-reduced matrix.

The L^2 Algorithm:

Nguyen and Stehlé proposed in 2005 a floating-point algorithm, called L^2 [NS09], which has the advantage to be practical and stable, but even more interesting, it has a better time complexity than the LLL algorithm, as highlighted in the following theorem.

Theorem 14 (L^2)**.** *Given an input basis $B = \{\mathbf{u_1}, \ldots, \mathbf{u_n}\}$ of a lattice $\mathcal{L} \in \mathbb{Z}^m$, where $b = \max_{1 \leqslant i \leqslant n}(\log \|\mathbf{u_i}\|)$, the L^2 algorithm outputs an approximate LLL-reduced basis (cf. Remark 17) in polynomial time in*

$$\mathcal{O}(n^4 m\, b(n+b)) \quad .$$

Accordingly, the L^2 algorithm has a complexity which is quadratic in the size of the input coefficients. Therefore, this algorithm is particularly interesting to reduce bases with large coefficients.

The $\tilde{L}^1$ Algorithm:

Recently in 2011, Novocin, Stehlé and Villard proposed an algorithm called $\tilde{L}^1$ [NSV11], with an improved complexity. This algorithm deals with smaller coefficients by taking advantage of the fact that only the most significant bits of the coefficients are crucial for the reduction. At each step, the least significant bits can therefore be neglected during the costly computations and reconsidered after. Actually, the $\tilde{L}^1$ algorithm can be seen as a generalization of the Knuth-Schönhage fast GCD algorithm [Knu71], from integers to matrices. Thereby, the $\tilde{L}^1$ algorithm can output an LLL-reduced basis for a full-rank lattice with a complexity which is quasi-linear in the size of the input coefficients. The complexity of $\tilde{L}^1$ is given in the following theorem.

Theorem 15 ($\tilde{L}^1$)**.** *Given an input basis $B = \{\mathbf{u_1}, \ldots, \mathbf{u_n}\}$ of a full rank lattice $\mathcal{L}$, where $b = \max_{1 \leqslant i \leqslant n}(\log \|\mathbf{u_i}\|)$, the $\tilde{L}^1$ algorithm outputs an approximate LLL-reduced basis (cf. Remark 17) in polynomial time in*

$$\mathcal{O}(n^{5+\varepsilon}b + n^{\omega+1+\varepsilon}b^{1+\varepsilon}) \quad ,$$

for any $\varepsilon > 0$ using fast integer arithmetic, where $\omega \leq 2.376$ is the matrix multiplication complexity constant.

The $\tilde{L}^1$ algorithm is for the time being the LLL-reduction algorithm holding the best complexity (note that a similar complexity bound can also be obtained by using the 2-dimensional BKZ' algorithm [HPS11, Th. 3]). However, for now the $\tilde{L}^1$ algorithm is still considered mainly theoretical since it is currently not implemented. This is the reason why, in this manuscript it will therefore be meaningful to mention when needed, the complexities associated to both algorithms: L^2 and $\tilde{L}^1$.

Remark 16. *We emphasize that the complexities of LLL and L^2 algorithms given in Theorem 13 and Theorem 14 assume that no fast integer arithmetic is implemented, as opposed to the complexity of $\tilde{L}^1$ given in Theorem 15, which uses fast integer arithmetic and fast linear algebra. The complexities of LLL and L^2 when using fast integer arithmetic are respectively $n^3 m b\tilde{O}(nb)$ and $n^2 m(n+b)b\tilde{O}(n)$, where $b = \max_{1 \leqslant i \leqslant n}(\log \|\mathbf{u_i}\|)$.*

Remark 17. *The L^2 and $\tilde{L}^1$ algorithms do not strictly output LLL-reduced bases following Definition 11, but they compute approximate LLL-reduced bases for a mild modification in the definition of the LLL-reduction. We refer the reader to [NS09] and [NSV11] for more details.*

Complexity Summary: We summarize in Table 4.1 the time complexities of the LLL, L^2 and $\tilde{L}^1$ algorithms, applied to an input basis $B = \{\mathbf{u_1}, \ldots, \mathbf{u_n}\}$ of a lattice $\mathcal{L} \in \mathbb{Z}^m$, where b represents the size of the maximal element in the input basis. As it is well highlighted by their calling designations, the LLL algorithm has a complexity which is cubic in b, while the L^2 is quadratic in b and the $\tilde{L}^1$ is quasi-linear in b.

Table 2.1: Time complexities of the LLL, L^2 and $\tilde{L}^1$ algorithms.

	LLL	L^2	$\tilde{L}^1$
Complexity	$\mathcal{O}(n^5 m\, b^3)$	$\mathcal{O}(n^4 m\, b(n+b))$	$\mathcal{O}(n^{5+\varepsilon} b + n^{\omega+1+\varepsilon} b^{1+\varepsilon})$

2.2.6 Number of Iterations of LLL-Reduction Algorithms

The number of loop iterations performed by LLL-reduction algorithms (LLL, L^2 and $\tilde{L}^1$) is upper bounded by $\mathcal{O}(n^2 b)$ [LLL82]. However, when the Gram-Schmidt norms of the input basis are balanced, the LLL-reduction algorithms require fewer loop iterations than in the worst case. Namely, it has been shown in [DV94] that the b term in the number of iterations $\mathcal{O}(n^2 b)$ can be replaced by the more refined term $\max \|\mathbf{b_i}^\star\| / \min \|\mathbf{b_i}^\star\|$. More precisely, [DV94] showed the following theorem:

Theorem 18 (Number of iterations)**.** *Let $\mathcal{L}$ be a lattice spanned by $B = \{\mathbf{b_1}, \ldots, \mathbf{b_n}\}$ and let $B^\star = \{\mathbf{b_1}^\star, \ldots, \mathbf{b_n}^\star\}$ be the corresponding Gram-Schmidt orthogonal basis. The upper bound on the number of loop iterations of LLL-reduction algorithms is:*

$$O\left(n^2 \log \frac{\max \|\mathbf{b_i}^\star\|}{\min \|\mathbf{b_i}^\star\|}\right) \quad .$$

Since the complexity of the LLL-reduction algorithms depends on the number of loop iterations, by using Theorem 18, the complexity of LLL-reduction can sometimes be decreased by some polynomial factor.

Remark 19. *The property of Gram-Schmidt norms balancedness which is highlighted in Theorem 18 will be used in Chapter 4 in order to improve the complexity of Coppersmith's method.*

2.3 Diophantine Problem and LLL-Reduction

The Diophantine problem consists in searching or studying, integer solutions for a system of equations given by:

$$f_i(x_1, \ldots, x_n) = 0 \ , \quad \text{where} \quad i = 1, \ldots, m \ .$$

Diophantine problems typically hold fewer equations than unknown variables and they involve finding integers that work correctly for all equations (if there are more equations than variables, then the problem amounts to solving an overdetermined system that can be done using classical tools like Gröbner bases, and thereby extracting the integer roots among all roots found). The problem of deciding whether such solutions exist has been formalized in the 10-th Hilbert Problem and has been shown in 1970 to be NP-hard in its general form.

Still, LLL-reduction in polynomial time has brought a non-negligible impact to the Diophantine problem. Thus, LLL-reduction has been used to find small integer solutions to Diophantine equations, but also to find simultaneous diophantine approximation, as recalled below.

2.3.1 Finding Small Integer Solutions

As previously said, the 10-th Hilbert problem is difficult in its general form. However, with the publication of LLL-reduction algorithms in polynomial time, one of its subproblems consisting in finding small integer solutions, has been shown to be solvable in polynomial time. Namely, Coppersmith in [Cop96b, Cop96a, Cop97] showed that if the searched integer roots are small enough, then lattice-based techniques can allow to recover them. The core idea involves finding new polynomial equations thanks to lattice-reduction in order to get as many equations as variables, and then solve the system easily. We describe in more details those techniques in Chapter 3.

2.3.2 Simultaneous Diophantine Approximation

Another famous application of lattice reduction algorithms involves the theory of Diophantine approximation. Apart from its own interest, being able to have good simultaneous approximations is a very useful building block for many algorithms. This theory deals with the approximation of numbers (rational or irrational), by rational numbers with special properties. Namely, let n be a positive integer, $e_1, e_2, \ldots, e_n$ be rational numbers and $\varepsilon \in \mathbb{R}$ satisfying $0 < \varepsilon < 1$. A theorem from Cassels [Cas, Sec.V.10] specifies that there exist integers $p_1, p_2, \ldots, p_n, q$ such that $|p_i - qe_i| \leqslant \varepsilon$ for $1 \leqslant i \leqslant n$, and $1 \leqslant q \leqslant \varepsilon^{-n}$.

However, even if it is proven that such integers exist, no known polynomial-time algorithm can find them. Yet, in [LLL82] it is shown that the LLL algorithm can be used to recover integers that satisfy a slightly weaker condition. Namely, they show the following theorem:

Theorem 20 (Simultaneous Diophantine Approximation)**.** *There exists a polynomial time algorithm (LLL) that, given a positive integer n and rational numbers $e_1, e_2, \ldots, e_n, \varepsilon$ satisfying $0 < \varepsilon < 1$, finds integers $p_1, p_2, \ldots, p_n, q$ for which*

$$|p_i - qe_i| \leqslant \varepsilon \qquad \textit{for} \quad 1 \leqslant i \leqslant n \quad ,$$

$$1 \leqslant q \leqslant 2^{\frac{n(n+1)}{4}} \varepsilon^{-n} \quad .$$

Proof. The idea consists in applying LLL on the following $(n+1) \times (n+1)$ matrix B:

$$B \;=\; \begin{pmatrix} 1 & 0 & \ldots & 0 & -e_1 \\ 0 & 1 & \ldots & 0 & -e_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & -e_n \\ 0 & 0 & \ldots & 0 & 2^{\frac{-n(n+1)}{4}} \varepsilon^{n+1} \end{pmatrix} \quad .$$

We denote by $\mathbf{v}$ the first vector of the LLL-reduced matrix and by $\mathbf{v}' = (p_1, p_2, \ldots, p_n, q)^T$ the vector of the transformation matrix such that $B\mathbf{v}' = \mathbf{v}$. Thus the vector $\mathbf{v}$ is such that

$$\mathbf{v} = (p_1 - qe_1,\ p_2 - qe_2,\ \ldots\ ,\ p_n - qe_n,\ q\,2^{\frac{-n(n+1)}{4}} \varepsilon^{n+1})^T \quad .$$

Since matrix B is upper triangular, one has

$$\det B = 2^{\frac{-n(n+1)}{4}} \varepsilon^{n+1} \quad .$$

Therefore, the first vector v of the LLL-reduced matrix satisfies:

$$\|\mathbf{v}\| \leqslant 2^{\frac{n}{4}} (\det B)^{\frac{1}{n+1}} = \varepsilon \quad .$$

Thus, it straightforwardly follows that $|p_i - qe_i| \leqslant \varepsilon$ for $1 \leqslant i \leqslant n$. It remains to show that $1 \leqslant q \leqslant 2^{\frac{n(n+1)}{4}} \varepsilon^{-n}$. Indeed, since the last component $q\,2^{\frac{-n(n+1)}{4}} \varepsilon^{n+1}$ of vector $\mathbf{v}$ is also $\leqslant \varepsilon$, one gets the condition $q \leqslant 2^{\frac{n(n+1)}{4}} \varepsilon^{-n}$. Eventually, since $\varepsilon < 1$ and $\mathbf{v} \neq 0$, one gets $q \neq 0$. Thus, if q is negative, then one rather considers vector $-\mathbf{v}$. This ensures that $q \geqslant 1$ because q is an integer, and it concludes the proof.

□

Hence, as depicted in Theorem 20, LLL-reduction algorithms have made effective a relaxed version of the simultaneous Diophantine approximation problem.

Remark 21. *Theorem 20 will be used in Chapter 5 in order to find a decomposition of the exponents r_i for moduli of the form $N = \prod_{i=1}^{k} p_i^{r_i}$, which satisfies some good conditions.*

Chapter 3

Finding Small Solutions to Polynomial Equations

Contents

The famous Diophantine problem which consists in finding integer roots of polynomial equations is hard in general. The problem of deciding whether such solutions exist is known as the 10^{th} Hilbert Problem, whose insolvability has been proven in 1970 by Yuri Matiyasevich [Mat00]. In cryptology, many security assumptions are based on the ability to solve specific Diophantine equations. For example, the RSA cryptosystem (see Chapter 1) is defeated if one can solve the bivariate polynomial equation over the integers $N - xy = 0$ where $x_0 = p$ and $y_0 = q$ are the searched integer solutions.

More generally, the following two problems are particularly interesting with regard to the cryptanalysis of RSA:

Problem 1: Find the integer roots of polynomial equations with integer coefficients. That is, we look for

$$(x_1, \ldots, x_d) \in \mathbb{Z} \quad \text{such that} \quad f(x_1, \ldots, x_d) = 0 \ .$$

Problem 2: Find the modular roots of modular polynomial equations with integer coefficients. That is, we look for

$$(x_1, \ldots, x_d) \in \mathbb{Z}/N\mathbb{Z} \quad \text{such that} \quad f(x_1, \ldots, x_d) \equiv 0 \mod N \ .$$

While those problems are hard in general, Coppersmith showed at EUROCRYPT '96 that if the roots are small enough, then lattice-based techniques can allow to recover them in polynomial time in the size of the coefficients [Cop96b, Cop96a, Cop97].

The global idea of these techniques consists in recovering, thanks to lattice reduction, some new polynomials which admit the same roots as the polynomial f, but with better properties than f, namely properties which allow to solve them easily.

For example, in the case where we look for integer roots of $f(x_1, \ldots, x_d)$ over the integers, the idea is to find d algebraically independent polynomial equations which all admit the same integer roots as the polynomial f (in fact one has to find $d-1$ equations since one already has one polynomial which is f). Then, from these d equations with d variables, one can easily retrieve those roots by using classical tools for solving systems of polynomial equations, like Gröbner bases [CLO07]. In other words, the method consists in finding a set of polynomials such that the obtained system forms an ideal of dimension 0, *i.e.* such that the computation of a Gröbner basis from this ideal allows to recover the searched solutions.

In the case where we look for modular roots of $f(x_1, \ldots, x_d)$, the principle is the same: one finds d algebraically independent polynomials which all admit the same modular roots as the polynomial f, but this time those equations hold over the integers. Yet as before, one can solve the corresponding system. Note that this time, one actually needs to find d equations because the modular polynomial f cannot be used into the system.

We highlight that the d required polynomials are obtained by LLL-reducing (cf Chapter 2) a particular matrix and by taking the polynomials corresponding to the d first vectors of the LLL-reduced matrix. Obviously, we will specify in the sequel the matrix used, and we will justify why the method works.

Heuristic: Before that, one should notify that in fact, Coppersmith's root finding algorithms for multivariate polynomial equations (*i.e.* $d \geqslant 2$ for modular equations and $d \geqslant 3$ for equations over the integers) are heuristic. This comes from the fact that one cannot be sure that the d equations found will be algebraically independent. It is interesting to note that in practice, Coppersmith's algorithms for multivariate polynomial equations have proven to be of great use and above all, to work generally well, despite

the heuristic. Besides, some work has been done with the aim of removing the heuristic in some cases, namely in [Bau08], the authors give a construction to make Coppersmith's methods rigorous for some multivariate polynomials.

Yet, Coppersmith's algorithms for multivariate polynomial equations are natural generalizations of the univariate modular case ($d = 1$) and of the bivariate integer case ($d = 2$) in the sense that the method is similar no matter the number d of variables, the main difference being the number of equations finally taken from the LLL-reduced matrix (which is d for the modular case and $d-1$ for the integer case). Thus, it is interesting to note that for the univariate modular case and for the bivariate integer case, since one makes use of one vector of the LLL-reduced matrix only, the result is not heuristic.

Complexity: When the number of variables is small, the bottleneck of Coppersmith's algorithms is the LLL-reduction. Indeed, once the LLL-reduced basis is computed, the retrieval of the roots is easily done by using classical tools for solving systems of polynomial equations, whose complexity is smaller than the one corresponding to an LLL-reduction (see proof of Corollary 24). As a consequence, the complexity of Coppersmith's algorithms depends on the LLL-reduction algorithm used (typically LLL, L^2 or $\tilde{L}^1$). Hence, this complexity depends on the dimension n of the matrix to be reduced, and on the maximal size b of the matrix coefficients. More precisely, in Coppersmith's matrices, the size b depends on the dimension n, on the degree δ of the polynomial and on the size of the module $\log N$. Thus the complexity depends on these three parameters: n, δ and $\log N$. We emphasize that contrarily to δ and $\log N$ which are fixed inputs of Coppersmith's algorithm, the dimension n is a parameter which can be chosen. Therefore at this stage, it is interesting to differentiate the asymptotical complexity and the practical complexity of Coppersmith's algorithm.

Indeed, in theory, in order to reach the bound on the solutions found by Coppersmith's algorithm, it is necessary to consider a matrix of huge dimension. As a consequence, May's survey [May10] gives for Coppersmith's lattice-based algorithm, the complexity upper bound $\mathcal{O}(\delta^5 \log^9 N)$ using the L^2 algorithm [NS09] as the reduction algorithm. A careful look gives a somewhat better upper bound: asymptotically, one may take a matrix of dimension $n = \mathcal{O}(\log N)$, and bit-size $\mathcal{O}((\log^2 N)/\delta)$, resulting in a complexity upper bound $\mathcal{O}((\log^9 N)/\delta^2)$ using L^2 .

In practice, though, the dimension of the matrix used cannot be that large. Therefore a combination of LLL-reduction and exhaustive search is generally employed to reach Coppersmith's bound. While this exhaustive search is asymptotically carried out in constant time, in reality, it can be, by all means, impractical.

Hence, in this chapter, we provide a new detailed analysis which allows to determine the asymptotical complexity of Coppersmith's algorithm. However, the exhaustive search difficulties which occur in practice are rather studied and improved in Chapter 4.

Roadmap. In Section 3.1, we highlight Coppersmith's main result (Theorem 22) for finding small solutions to univariate modular equations, and we review the associated method allowing to get to that result. We also provide a new analysis to determine

the asymptotical complexity of such an algorithm. In Section 3.2 we briefly recall the main result for the bivariate integer case. We refer the reader to [JM06] for more details about the multivariate integer and modular cases. Eventually, in Section 3.3, we provide a reminder on the Boneh-Durfee-Howgrave-Graham method for factoring moduli of the form $N = p^r q$, which is an extension of Coppersmith's method for the univariate modular case.

3.1 Coppersmith's Method for Finding Small Roots to Univariate Modular Equations

3.1.1 The Main Result

Coppersmith in [Cop96b, Cop97] showed how to find efficiently all small roots of univariate modular polynomial equations. In the following, we recall Coppersmith's theorem which gives a condition on the size of the small roots and the complexity of the method.

Theorem 22 (Coppersmith)**.** *Let $f(x)$ be a monic polynomial of degree δ in one variable, modulo an integer N of unknown factorization. Let X be such that $X < N^{\frac{1}{\delta}}$. One can find all integers x_0 with $f(x_0) \equiv 0 \mod N$ and $|x_0| < X$ in time polynomial in $\log N$ and δ.*

The technique designed by Coppersmith to obtain this result was later simplified by Howgrave-Graham in [HG97]. Coppersmith and Howgrave-Graham's methods have the same asymptotical complexity, but since the latter one holds a more natural approach and is easier to implement (in fact both methods lie in dual vectorial spaces relative to one another), it is commonly adopted. Therefore we describe in the sequel the method, following the classical Howgrave-Graham's approach and we refer the reader to [Cop96b, Cop97] or to [Rit10, Bau08] for descriptions of the original Coppersmith's method. For the sake of simplicity, one will adopt the calling " Coppersmith's method" even when Howgrave-Graham's approach is employed.

3.1.2 The Method

The core idea of the method consists in reducing the problem to solving univariate polynomial equations over the integers, by transforming modular roots into integral roots. More precisely, it constructs a polynomial $v(x) \in \mathbb{Z}[x]$ such that: if $x_0 \in \mathbb{Z}$ is such that $f(x_0) \equiv 0 \pmod N$ and $|x_0| \leq X$, then $v(x_0) = 0$ over $\mathbb{Z}$. Then, solving this univariate polynomial equation $v(x)$ over the integers can efficiently be done by using Schönhage's root isolation algorithm [Sch82, Sec. 5.2].

Thus, in order to obtain such a polynomial $v(x)$, one considers many polynomials which correspond to some specific multiples of polynomial f and which all admit x_0 as a root modulo N^m, where m is a given parameter. By applying LLL on the matrix containing all those polynomials, one retrieves a polynomial v which also admits x_0 as

a root modulo N^m, but with small coefficients. Accordingly, if the solution x_0 is small enough, the polynomial v will be such that $v(x_0) = 0$ over the integers, and can easily be solved.

More precisely, the method goes as follows. Let $m \geq 1$ be an integer parameter and define the following family of $n = \delta m + 1$ polynomials where δ is the degree of polynomial f:

$$g_{i,j}(x) = x^j N^{m-i} f^i(x) \tag{3.1}$$

for all i and j such that $0 \leq i < m$ and $0 \leq j < \delta$, and $j = 0$ for $i = m$.

These n polynomials satisfy:

$$\text{if } f(x_0) \equiv 0 \pmod{N} \text{ for some } x_0 \in \mathbb{Z}, \text{ then } g_{i,j}(x_0) \equiv 0 \pmod{N^m}.$$

Indeed, since $f(x_0) \equiv 0 \pmod{N}$, one has

$$g_{i,j}(x_0) \equiv x_0^j N^{m-i} f^i(x_0) \equiv x_0^j N^{m-i} N^i \equiv x_0^j N^m \equiv 0 \pmod{N^m} \quad .$$

Then one constructs the n-dimensional lattice L spanned by the rows of the $n \times n$ matrix B formed by the n coefficient vectors of $g_{i,j}(xX)$ where X is a known upper-bound on the solution x_0. These polynomials are ordered by increasing degree (*e.g.* in the order $(i,j) = (0,0), (0,1), \cdots, (0,\delta-1), (1,0), \cdots, (m-1,\delta-1), (m,0)$) and the coefficients are ordered by increasing monomial degree: the first coefficient is thus the constant term of the polynomial. The matrix B is lower triangular, and its n diagonal entries are:

$$\left(N^m, N^m X, \ldots, N^m X^{\delta-1}, \ldots, N^1 X^{\delta m-\delta}, \ldots, N^1 X^{\delta m-2}, N^1 X^{\delta m-1}, N^0 X^{\delta m}\right), \tag{3.2}$$

because $f(x)$ is monic (that is, the coefficient associated to the leading monomial x^δ is 1). In other words, the exponent of X increases by one at each row, while the exponent of N decreases by one every δ rows. Therefore, the matrix B has a block structure as depicted in the following, where $f(x) = a_0 + a_1 x + \cdots + a_\delta x^\delta$.

$$B = \begin{pmatrix}
N^m & & & & & & & & & \\
 & XN^m & & & & & & & & \\
 & & \ddots & & & & & & & \\
 & & & X^{\delta-1}N^m & & & & & & \\
a_0N^{m-1} & \dots & & & X^{\delta}N^{m-1} & & & & & \\
 & a_0XN^{m-1} & \dots & & & X^{\delta+1}N^{m-1} & & & & \\
 & & \ddots & & & & \ddots & & & \\
 & & & a_0X^{\delta-1}N^{m-1} & \dots & & X^{2\delta-1}N^{m-1} & & & \\
 & & \dots & & \dots & & & \ddots & & \\
a_0^{m-1} & & \dots & & \dots & & X^{\delta(m-1)}N & & & \\
 & a_0^{m-1}X & & \dots & & \dots & & X^{\delta(m-1)+1}N & & \\
 & & \ddots & & & & & & \ddots & \\
 & & & a_0^{m-1}X^{\delta-1} & \dots & & \dots & & & X^{\delta m-1}N \\
a_0^m & & \dots & & \dots & & \dots & & & X^{\delta m}
\end{pmatrix}$$

Since matrix B is diagonal, it follows that the determinant of B is the product of its diagonal elements, that is

$$\det(B) = \left(\prod_{i=1}^{m} N^{\delta i}\right)\left(\prod_{i=1}^{n-1} X^{i}\right) = N^{\frac{1}{2}(n-1)(m+1)}X^{\frac{1}{2}n(n-1)} \quad .$$

Then the LLL algorithm (or an improved algorithm with similar output - see Section 2.2) is applied to the matrix B. An LLL-reduced matrix B^R is therefore output. Since in the original matrix B, all polynomials where such that $g_{i,j}(x_0) \equiv 0 \mod N^m$, the polynomials retrieved after LLL-reduction of B keep this same property. In particular, the polynomial $v(x)$ associated to the first vector of B^R satisfies:

$$v(x_0) \equiv 0 \mod N^m \quad .$$

At a guess, the problem of solving $v(x) \equiv 0 \mod N^m$ is as difficult as the original problem of solving $f(x) \equiv 0 \mod N^m$. However, there is a crucial difference between both problems: the coefficients of the polynomial $v(x)$ are small. Namely, according to Theorem 12, the first polynomial v of B^R is a non-zero polynomial $\in \mathbb{Z}[x]$ such that

$$\|v(xX)\| \leq 2^{\frac{n-1}{4}} \det(B)^{\frac{1}{n}} = 2^{\frac{n-1}{4}} N^{\frac{(n-1)(m+1)}{2n}} X^{\frac{n-1}{2}} \quad . \tag{3.3}$$

In fact, one would like this polynomial to be such that

$$v(x_0) < N^m \quad \text{so that} \quad v(x_0) = 0 \text{ over } \mathbb{Z} \quad .$$

Indeed, if it is the case, one can solve it easily over the integers. To this aim, one uses the following elementary criterion:

Lemma 23 (Howgrave-Graham [HG97]). *Let $v(x) \in \mathbb{Z}[x]$ be a polynomial with at most n non-zero coefficients. Let N be an integer ≥ 1. Assume that $\|v(xX)\| < \frac{N^m}{\sqrt{n}}$ for some $X \in \mathbb{R}$. If $x_0 \in \mathbb{Z}$ is such that $v(x_0) \equiv 0 \mod N^m$ and $|x_0| \leq X$, then $v(x_0) = 0$.*

Proof. We have:

$$\begin{aligned} |v(x_0)| &= \left|\sum_{i=1}^{n} v_i x_0^i\right| = \left|\sum_{i=1}^{n} v_i X^i \left(\frac{x_0}{X}\right)^i\right| \leq \sum_{i=1}^{n} \left|v_i X^i \left(\frac{x_0}{X}\right)^i\right| \\ &\leq \sum_{i=1}^{n} \left|v_i X^i\right| \leq \sqrt{n}\|v(xX)\| < N^m \,. \end{aligned}$$

Therefore we have $|v(x_0)| < N^m$, and since $v(x_0) \equiv 0 \mod N^m$, this gives $v(x_0) = 0$. □

It follows from Lemma 23 that the polynomial $v(x)$ holds over the integers if $\|v(xX)\| < N^m/\sqrt{n}$. By using inequality 3.3, this gives the following condition:

$$2^{\frac{n-1}{4}} N^{\frac{(n-1)(m+1)}{2n}} X^{\frac{n-1}{2}} < \frac{N^m}{\sqrt{n}} \quad .$$

Therefore, in order for this inequality to be accurate, one gets the following condition on the bound X:

$$X^{\frac{n-1}{2}} < 2^{-\frac{n-1}{4}} N^{m-\frac{(n-1)(m+1)}{2n}} n^{-\frac{1}{2}}$$

which gives:

$$X < 2^{-\frac{1}{2}} N^{\frac{2m}{n-1}-\frac{m+1}{n}} n^{-\frac{1}{n-1}} \quad .$$

Since we have:

$$\frac{2m}{n-1} - \frac{m+1}{n} = \frac{2}{\delta} - \frac{m+1}{n} = \frac{2n - \delta(m+1)}{\delta n} = \frac{2n-(n-1)-\delta}{\delta n} = \frac{n-\delta+1}{\delta n} \quad ,$$

we get the condition:

$$X < 2^{-\frac{1}{2}} N^{\frac{n-\delta+1}{\delta n}} n^{-\frac{1}{n-1}}. \tag{3.4}$$

Thus, Coppersmith's method allows to find all small integers roots x_0 such that $|x_0| < X$ where X satisfies (3.4).

About the need of the bound X in matrix B: As previously explained, the polynomials $g_{i,j}(x)$ are such that $g_{i,j}(x_0) \equiv 0 \pmod{N^m}$, but in matrix B we set the vectors associated to the polynomials $g_{i,j}(xX)$, that is, each coefficient is multiplied by X^i where $0 \leqslant i \leqslant d-1$ represents the index of the column. Obviously, it is not true that $g_{i,j}(x_0X) \equiv 0 \pmod{N^m}$. This is the reason why, once the short polynomial $v(xX)$ is obtained by LLL-reduction, one has to perform the reverse step: divide each coefficient of $v(xX)$ by X^i, to get polynomial $v(x)$. Hence, it is natural to justify the need of such a

bound X in matrix B, especially by considering that it makes the determinant increase, together with the upper bound of the shortest vector $v(xX)$. In fact, if one considers polynomials $g_{i,j}(x)$ instead of $g_{i,j}(xX)$ in matrix B, then the LLL-reduction ensures that the first polynomial $v(x)$ evaluated for $x = 1$ is small, but not necessarily $v(x_0)$. Therefore, the polynomial $v(x_0)$ may not hold over the integers as required.

Exhaustive search: As mentioned before, the upper bound X on the solutions that are found by Coppersmith's method satisfies (3.4). Therefore, Coppersmith's method does not directly achieve the bound $N^{1/\delta}$ given in Theorem 22. Indeed, it finds efficiently all roots up to some bound X ($< N^{1/\delta}$) depending on the dimension n of the lattice used. But when n is sufficiently large, then X becomes sufficiently close to $N^{1/\delta}$ so that one can find all roots up to $N^{1/\delta}$. Indeed, in condition (3.4) the two terms $2^{-\frac{1}{2}}$ and $n^{-\frac{1}{n-1}}$ are asymptotically in $\mathcal{O}(1)$. Thus, the main term is $N^{\frac{n+\delta-1}{\delta n}}$ and by taking $n = \mathcal{O}(\log N)$, one has:

$$N^{\frac{n-\delta+1}{\delta n}} = N^{\frac{1}{\delta}+\frac{1-\delta}{\delta n}} = \mathcal{O}(N^{\frac{1}{\delta}}) \quad .$$

However, in practice the bound $X = N^{1/\delta}$ should not be reached by using such a large dimension n. Instead, it is faster to use a lattice of reasonable dimension, and perform exhaustive search on the most significant bits of the solutions. Namely, we consider polynomials:

$$f_t(x) = f(X \cdot t + x) \quad ,$$

where

$$0 \leqslant t < \frac{N^{1/\delta}}{X}$$

and

$$X = \lfloor 2^{-1/2} N^{\frac{n-\delta+1}{\delta n}} n^{-\frac{1}{n-1}} \rfloor \quad . \tag{3.5}$$

Thus, an initial solution x_0 that can be written $x_0 = X \cdot t_0 + x_0'$ is obtained by finding the solution x_0' of the polynomial f_{t_0}. In this case, this solution satisfies $|x_0'| < X$ and it has a correct size for LLL to find it using a lattice of dimension n. For each polynomial f_t, one runs LLL on the matrix B associated to f_t. The solution x_0 is then found for the right value t_0, *i.e.* for the right polynomial f_{t_0}. As it will be shown later, the exhaustive search is performed in constant time.

Eventually, the whole Coppersmith's method is depicted in Algorithm 4.

It is straightforward that when the degree of the polynomial is sufficiently large in comparison to N (it is sufficient that $\delta + 1 \geq (\log N)/2$ in our analysis) then a brute force search of the solutions provides a comparable complexity result. In a similar way, if the degree of the polynomial is one, then a direct approach naturally allows to recover the solutions. Thus, this explains why we implicitly consider polynomials not in these cases in Algorithm 4, that is to say polynomials verifying $2 < \delta + 1 < (\log N)/2$.

Algorithm 4 Coppersmith's Method

Input: Two integers $N \geq 1$ and $m \geq 1$, a univariate degree-δ monic polynomial $f(x) \in \mathbb{Z}[x]$ with coefficients in $\{0, \ldots, N-1\}$ and $2 < \delta + 1 < (\log N)/2$.

Output: All $x_0 \in \mathbb{Z}$ s.t. $|x_0| \leq N^{1/\delta}$ and $f(x_0) \equiv 0 \mod N$.

1: Let $n = m\delta + 1$, X the bound given in (3.5), and $t = 0$.
2: **while** $Xt < N^{1/\delta}$ **do**
3: $f_t(x) = f(Xt + x) \in \mathbb{Z}[x]$.
4: Build the $n \times n$ lower-triangular matrix B whose rows are the $g_{i,j}(xX)$'s defined by (3.1).
5: Run the L^2 algorithm [NS09] on the matrix B.
6: The first vector of the reduced basis corresponds to a polynomial of the form $v(xX)$ for some $v(x) \in \mathbb{Z}[x]$.
7: Compute all the roots x_0' of the polynomial $v(x) \in \mathbb{Z}[x]$ over $\mathbb{Z}$.
8: Output $x_0 = Xt + x_0'$ for each root x_0' which satisfies $f_t(x_0') \equiv 0 \pmod N$ and $|x_0'| \leq X$.
9: $t \leftarrow t + 1$.
10: **end while**

3.1.3 Complexity

In the following, the complexity of Coppersmith's method is enlightened. Namely, as shown in the proof of Corollary 24, since the exhaustive search is performed in constant time, and since the bottleneck of Algorithm 4 is the LLL-reduction, the asymptotical complexity of Coppersmith's method is the cost of one LLL-reduction. Thus, the complexity depends on the lattice reduction algorithm which is used in Step 5 of Algorithm 4. This is why in Corollary 24 we give two complexities: one associated to L^2 , and the other, to $\tilde{L}^1$.

Corollary 24. *Algorithm 4 of Theorem 22 with* $n = \lfloor \log N \rfloor$ *and* $X = \lfloor 2^{-1/2} N^{\frac{n+\delta-1}{\delta n}} n^{-\frac{1}{n-1}} \rfloor$ *runs in time* $\mathcal{O}((\log^9 N)/\delta^2)$ *without fast integer arithmetic using* L^2 *in Step 5, or* $\mathcal{O}((\log^{7+\varepsilon} N)/\delta)$ *for any* $\varepsilon > 0$ *using fast integer arithmetic and* $\tilde{L}^1$ *in Step 5.*

Proof. As a first step, we show that Algorithm 4 performs a constant number of loop iterations. Indeed, consider the bound $X = \lfloor 2^{-1/2} N^{\frac{n+\delta-1}{\delta n}} n^{-\frac{1}{n-1}} \rfloor$ achieved by one loop of Algorithm 4. By definition, Algorithm 4 performs at most $\mathcal{O}(N^{1/\delta}/X)$ loop iterations. Recall that we have $2 < \delta + 1 < (\log N)/2$, since in the other cases a direct approach provides a much better complexity. We have $\mathcal{O}(X) = \mathcal{O}(N^{\frac{n+\delta-1}{\delta n}})$ and:

$$N^{\frac{1}{\delta}} / N^{\frac{n+\delta-1}{\delta n}} = N^{\frac{1}{\delta} - \frac{n+\delta-1}{\delta n}} = N^{\frac{\delta-1}{\delta n}} \leq N^{\frac{1}{n}} .$$

Therefore, for $n = \lfloor \log N \rfloor$, we conclude that $N^{\frac{1}{\delta}} / N^{\frac{n+\delta-1}{\delta n}} = \mathcal{O}(1)$, that is, the number of loop iterations is constant.

As a second step, we provide the running time for L^2 and $\tilde{L}^1$ algorithms in Step 5 of Algorithm 4. The dimension of B is $n = \delta m + 1$, and the entries of the matrix B have

bit-size $\mathcal{O}(m \log N)$. Therefore the running time of L^2 in Step 5 without fast integer arithmetic is

$$\mathcal{O}(\delta^6 m^7 \log N + \delta^5 m^7 \log^2 N) = \mathcal{O}(\delta^5 m^7 \log^2 N) \ ,$$

because $\delta + 1 < (\log N)/2$. Furthermore, the running time of $\tilde{L}^1$ in Step 5 is

$$\mathcal{O}(m^{6+\varepsilon} \delta^{5+\varepsilon} \log N + m^{\omega+2+2\varepsilon} \delta^{\omega+1+\varepsilon} \log^{1+\varepsilon} N)$$

for any $\varepsilon > 0$ using fast integer arithmetic, where $\omega \leq 2.376$ is the matrix multiplication complexity constant. Yet, with $m = \lfloor \log N/\delta \rfloor$, one gets the complexities $\mathcal{O}((\log^9 N)/\delta^2)$ for L^2 and $\mathcal{O}((\log^{7+\varepsilon} N)/\delta)$ for $\tilde{L}^1$.

As a last step, we emphasize that the bottleneck of Coppersmith's algorithm is the reduction step (Step 5). Indeed, let ℓ be the maximal bit-size of the coefficients of $v(x) \in \mathbb{Z}[x]$ in Step 7: we know that $\ell \leq m \log N$, and the degree of $v(x)$ is $\leq n$. Then Step 7 can be performed in time $\mathcal{O}(n^3(\ell + \log n)) = \mathcal{O}(m \log^4 N) = \mathcal{O}((\log^5 N)/\delta)$ using Schönhage's root isolation algorithm [Sch82, Sec. 5.2]. Hence, the cost of Step 7 is less than the one of Step 5.

Thus, the asymptotic complexity of Coppersmith's algorithm is the one of one call to LLL (L^2 or $\tilde{L}^1$). This allows to conclude the proof.

□

Remark 25. *The complexity* $(\log N, \delta)$ *highlighted in Theorem 22 is the complexity stated in [Cop96b], but surprisingly in [Cop97], the enunciated complexity is polynomial in* $(\log N, 2^\delta)$ *where* δ *is the degree of the polynomial equation. It seems that there is no reason that it should be* 2^δ *rather than* δ *since the exhaustive search is performed in constant time. In any case, one refers the reader to Chapter 4 for a complexity which is polynomial in* $\log N$*, i.e. which is independent of* δ*. And in any case, we can assume that* $\delta \leqslant \log N$ *since otherwise we would get the condition* $X < N^{1/\delta} \leq \exp(1)$*, hence only a constant number of possible roots* x_0*. Thus, the complexity is polynomial in* $\log N$ *only.*

Thus, in Chapter 4, we propose a new method called *Rounding and Chaining* which allows to improve the complexities given in Corollary 24. In particular, as it was just specified in previous remark, the complexities will be independent of the degree δ.

3.1.4 Applications

Many applications have risen from Coppersmith's method for finding small solutions to univariate modular polynomial equations (see May's survey [May10]). Perhaps the most famous one is the vulnerability of RSA when using a public key e which is small, namely for $e = 3$. Indeed, in this case, one has the equivalence $m^3 \equiv c \mod N$, where m is the secret message and c is the known ciphered message. Then by decomposing the message m as $m = M + x$, and by assuming that M is a known part of m, one gets the polynomial equation $f(x) = (M + x)^3 - c \equiv 0 \mod N$. This is a univariate modular polynomial equations of degree 3. According to Coppersmith's theorem, one can recover

x if it is smaller than $N^{1/3}$. This means that one can recover the entire message if one knows 2/3 of the message m.

Another application which requires an extension of this method, is the factorization of moduli of the form $N = p^r q$ for large r. Such an application will be discussed in more details in Section 3.3.

3.2 Coppersmith's Method for Finding Small Roots to Bivariate Equations Over the Integers

3.2.1 The Main Result

As specified before, Coppersmith also proposed a method that allows to find small integer roots of bivariate polynomials over $\mathbb{Z}$ [Cop96b, Cop97]. Namely, one would like to find small (x_0, y_0) such that $f(x_0, y_0) = 0$. In the following, we recall Coppersmith's main result for the bivariate integer case.

Theorem 26 (Coppersmith)**.** *Let $f(x, y)$ be an irreducible polynomial in two variables over $\mathbb{Z}$, of maximum degree δ in each variable separately. Let X and Y be upper bounds on the desired integer solution (x_0, y_0), and let $W = \max_{i,j} |f_{ij}|X^i Y^j$. If $XY < W^{2/(3\delta)}$, then in time polynomial in $(\log W, 2^\delta)$, one can find all integer pairs (x_0, y_0) such that $f(x_0, y_0) = 0$, $|x_0| \leq X$, and $|y_0| \leq Y$.*

3.2.2 Core Idea of the Method

In the scope of this thesis, a simple overview of the method is sufficient, however, we refer the interested reader to [Cor07] for more details.

Coppersmith's approach for the bivariate integer case is very similar to the one used to solve univariate modular polynomials. The idea consists in constructing, thanks to lattice reduction, a new equation $v(x, y)$ which admits the same roots as the original polynomial equation f, and which is algebraically independent with f, so that one can easily solve the system containing the two polynomial equations f and v with two variables. As for the univariate modular case, a simplification of the method to construct the polynomial v has been proposed by Coron at CRYPTO 2007 [Cor07]. This method involves an adaptation of Howgrave-Graham's version for the univariate modular case. The core idea consists in considering many polynomials which correspond to some specific multiples of polynomial f and which all admit x_0 as a root modulo N, where N is a well chosen integer. By applying *LLL* on the matrix containing all those polynomials (in fact we apply *LLL* on a submatrix obtained after triangularization), one retrieves a polynomial v which also admits (x_0, y_0) as a root modulo N, but with small coefficients. Accordingly, if the solution (x_0, y_0) is small enough, the polynomial v will be such that $v(x_0, y_0) = 0$ over the integers, which corresponds to the sought equation.

3.2.3 Applications

Undoubtedly, the most famous application of Coppersmith's method for finding small roots of bivariate polynomial equations over the integers is the factorization of RSA moduli $N = pq$ when half of p is known. Indeed, by writing $p = P + x$ where P is the known upper part of p, one can retrieve half of q as follows: compute $Q = N/P$ and write $q = Q + x$. Thus, one gets the equation $N - (P + x)(Q + y) = 0$. This is a bivariate polynomial equation over the integers, of maximum degree 1 in each variable x and y separately. According to Coppersmith's theorem, one can recover the solutions x and y if the product is smaller than $N^{1/2}$. This leads to the aforementioned result: one can factorize N with the knowledge of half of p.

3.3 The Boneh-Durfee-Howgrave-Graham Method for Factoring $N = p^r q$

3.3.1 Motivations

The use of moduli of the form $N = p^r q$ has been introduced in cryptography many years ago and studied ever since. In particular, for the case $r = 2$, at least two applications have been proposed: one from Fujioka *et al.* [FOM91] where a modulus $N = p^2 q$ is used for the design of an electronic cash scheme, and one from Okamoto and Ushiyama [OU98] for the construction of a practical public key cryptosystem which is proven to be as secure as factoring $N = p^2 q$. More generally, it has been shown by Takagi in [Tak98] that the use of moduli $N = p^r q$ in RSA could lead to a decryption which is significantly faster than that with classical RSA moduli $N = pq$. The idea consists in noticing that in order to preclude both the number field sieve and the elliptic curve method, one can use a smaller private key d with $N = p^r q$ than with $N = pq$, for similar sizes of N.

3.3.2 The Main Result

At Crypto 99 [BDHG99], Boneh, Durfee and Howgrave-Graham analyzed the security of the use of such moduli as far as lattice-based methods are concerned. As a result, they adapted Coppersmith's method for univariate modular polynomials, in order to design a method which factorizes moduli $N = p^r q$ in polynomial time, under the condition that r is large, namely that $r \simeq \log p$ when $q \leqslant p^{\mathcal{O}(1)}$. In the following, we recall their main theorem.

Theorem 27 (BDH). *Let $N = p^r q$ where $q < p^c$ for some c. The factor p can be recovered from N, r, and c by an algorithm with a running time of:*

$$exp\left(\frac{c+1}{r+c} \cdot \log p\right) \cdot \mathcal{O}(\gamma),$$

where γ is the time it takes to run LLL on a lattice of dimension $\mathcal{O}(r^2)$ with entries of size $\mathcal{O}(r \log N)$. The algorithm is deterministic, and runs in polynomial space.

When p and q have similar bitsize one can take $c = 1$. Thus in that case we have $(c+1)/(r+c) = \mathcal{O}(1/r)$. Therefore, knowing a fraction $1/r$ of the bits of p is enough for a polynomial-time factorization of $N = p^r q$. Alternatively, if no part of p is known, the algorithm is still polynomial time when r is large, namely, when $r = \Omega(\log p)$. Indeed, in this case, only a constant number of bits of p must be known, hence those bits can be recovered by exhaustive search, and factoring $N = p^r q$ becomes polynomial-time.

More generally, for arbitrary c, that is for all sizes of p and q, if $(r + c)/(c + 1) = \mathcal{O}(\log p)$, then the running time becomes $\exp(\mathcal{O}(1)) \cdot \mathcal{O}(\gamma)$, which is polynomial in $\log N$.

3.3.3 The Method

In the sequel we recall the main steps of the BDH method that allow to get to Theorem 27. We refer the reader to [BDHG99] for more details.

Let $N = p^r q$. Assume that we are also given an integer P such that $p = P + x_0$ where the high-order bits of P are the same as the high-order bits of p, and x_0 is a small unknown. One considers the polynomial $f(x) = (P + x)^r$ which satisfies:

$$f(x_0) \equiv (P + x_0)^r \equiv 0 \pmod{p^r} \ .$$

This polynomial equation is a univariate equation of degree r and modulo p^r. Thus the main difference between such an equation and the one used in the univariate modular case (Section 3.1) is that the present modulus p^r is unknown, contrarily to the previous modulus N which was public. On the face of it, this makes a direct application of Coppersmith's method inconvenient.

However, since $N = p^r q$ one has:

$$N \equiv 0 \pmod{p^r} \ .$$

Therefore, even if the modulus p^r is unknown, one can create, as in Section 3.1, some polynomials with "good" properties thanks to the knowledge of N.

More precisely, for a given integer m one considers the polynomials

$$g_{i,k}(x) = N^{m-k} x^i f^k(x) \tag{3.6}$$

for $0 \leq k \leq m$ and $i \geq 0$. Thus, for all k and i, we have:

$$g_{i,k}(x_0) \equiv N^{m-k} \cdot x_0^i \cdot f^k(x_0) \equiv 0 \pmod{p^{rm}}$$

Let X be a bound on x_0. One considers the lattice spanned by the coefficient vectors of $g_{i,k}(xX)$ for $0 \leq k \leq m-1$ and $0 \leq i \leq r-1$, and also $g_{i,k}(xX)$ for $k = m$ and $0 \leq i \leq n - mr - 1$, where n is a parameter which is actually the lattice dimension. As depicted below, the matrix B has a block structure. All blocks contain r rows, apart from the last one which somehow completes the matrix so that it has a dimension n.

$$B = \begin{pmatrix}
N^m \\
 & XN^m \\
 & & \ddots \\
 & & & X^{r-1}N^m \\
P^rN^{m-1} & & \dots & & X^rN^{m-1} \\
 & P^rXN^{m-1} & & \dots & & X^{r+1}N^{m-1} \\
 & & \ddots & & & & \ddots \\
 & & & P^rX^{r-1}N^{m-1} & & \dots & & X^{2r-1}N^{m-1} \\
 & & \dots & & \dots & & \ddots \\
P^{r(m-1)}N & & \dots & & \dots & & X^{r(m-1)}N \\
 & P^{r(m-1)}XN & & \dots & & \dots & & X^{r(m-1)+1}N \\
 & & \ddots & & & & & & \ddots \\
 & & & P^{r(m-1)}X^{r-1}N & & \dots & & \dots & & X^{rm-1}N \\
P^{rm} & & \dots & & \dots & \dots & & \dots & X^{rm} \\
 & P^{rm}X & & \dots & & \dots & & \dots & & X^{rm+1} \\
 & & \ddots & & & & & \ddots \\
 & & & P^{rm}X^{n-mr-1} & & \dots & & \dots & \dots & & X^{n-1}
\end{pmatrix}$$

Since the matrix basis B of the lattice is triangular, the determinant of the lattice is the product of the diagonal entries, which gives:

$$\det B = \left(\prod_{k=0}^{m-1} \prod_{i=0}^{r-1} N^{m-k} \right) \left(\prod_{j=0}^{n-1} X^j \right) = N^{rm(m+1)/2} X^{n(n-1)/2} < N^{rm(m+1)/2} X^{n^2/2} \quad .$$

We apply the LLL algorithm on the matrix B. Again, since in the original matrix B, all polynomials where such that $g_{i,k}(x_0) \equiv 0 \bmod p^{rm}$, the polynomials retrieved after LLL-reduction of B keep this same property. In particular, the polynomial $v(x)$ associated to the first vector of the reduced matrix satisfies:

$$v(x_0) \equiv 0 \bmod p^{rm} \quad .$$

Furthermore, according to Theorem 12, this first polynomial v is such that

$$\|v(xX)\| \leqslant 2^{(n-1)/4} (\det B)^{1/n} < 2^{(n-1)/4} N^{rm(m+1)/(2n)} X^{n/2} < N^{rm(m+1)/(2n)} (2X)^{n/2} \quad .$$

In fact, one would like this polynomial to be such that

$$v(x_0) < p^{rm} \quad \text{so that} \quad v(x_0) = 0 \text{ over } \mathbb{Z} \quad .$$

From Lemma 23 and omitting the $\sqrt{n}$ factor, this property is satisfied if $\|v(xX)\| < p^{rm}$, which gives the condition:

$$N^{rm(m+1)/(2n)} (2X)^{n/2} < p^{rm} \quad ,$$

or equivalently

$$(2X)^{n^2/2} < p^{nrm} N^{-rm(m+1)/2} \quad .$$

We assume that $q < p^c$ for some $c > 0$. This gives $N < p^{r+c}$, which gives the condition:

$$(2X)^{n^2/2} < p^{nrm - r(r+c)m(m+1)/2} \ .$$

The larger the value $nrm - r(r+c)m(m+1)/2$, the larger the solution x_0 with $|x_0| < X$ that can be found. Therefore, for a fixed r, we wish to maximize the value of $nm - (r+c)m(m+1)/2$, so that one can use weaker approximations P. One can rewrite this value as follows:

$$nm - \frac{(r+c)m(m+1)}{2} = -\frac{r+c}{2}m^2 + \left(n - \frac{r+c}{2}\right)m \quad .$$

A polynomial $ax^2 + bx + c$ is maximized at $x = -b/(2a)$ with maximum $-\Delta/(4a)$ where $\Delta = b^2 - 4ac$. Therefore in our case, the previous value is maximized at:

$$m = \left(n - \frac{r+c}{2}\right) \times \frac{2}{2(r+c)} = \frac{n}{r+c} - \frac{1}{2} \ .$$

Therefore, we set the positive integer parameter m to :

$$m = \left\lfloor \frac{n}{r+c} - \frac{1}{2} \right\rfloor \quad ,$$

and one may choose the dimension n such that $\frac{n}{r+c} - \frac{1}{2}$ is within $\frac{1}{2r+c}$ of an integer. The maximum of the value $nm - (r+c)m(m+1)/2$ is then:

$$\left(n - \frac{r+c}{2}\right)^2 \times \frac{2}{4(r+c)} = \frac{\left(n - \frac{r+c}{2}\right)^2}{2(r+c)} > \frac{n^2 - n \cdot (r+c)}{2(r+c)} = \frac{n^2}{2(r+c)}\left(1 - \frac{r+c}{n}\right) \ .$$

Therefore, we get the condition:

$$(2X)^{\frac{n^2}{2}} < p^{r \times \frac{n^2}{2(r+c)}\left(1 - \frac{r+c}{n}\right)} \ ,$$

which gives the bound (by neglecting the factor 2):

$$X < p^{\frac{r}{r+c}\left(1 - \frac{r+c}{n}\right)} \quad .$$

Therefore we deduce the following condition on X under which the solution x_0 can be retrieved in polynomial time in $\log(p)$ by using Coppersmith's method:

$$X < p^{1 - \frac{c}{r+c} - 2\frac{r}{n}}, \tag{3.7}$$

which proves Lemma 3.3 from [BDHG99].

Lemma 28 (Lemma 3.3 from [BDHG99]). *Let $N = p^r q$ be given, and assume $q < p^c$ for some c. Furthermore assume that P is an integer satisfying*

$$|P - p| < p^{1-\frac{c}{r+c}-2\frac{r}{n}}$$

Then the factor p may be computed from N, r, c and P by an algorithm whose running time is dominated by the time it takes to run LLL on a lattice of dimension n.

In [BDHG99] the authors take $n = 2r(r + c)$, which gives:

$$|P - p| < p^{1-\frac{c+1}{r+c}} \quad .$$

Therefore, in order to factor moduli of the form $N = p^r q$ it suffices to perform exhaustive search on a fraction $(c + 1)/(r + c)$ of the bits of p, and the running time becomes:

$$\exp\left(\frac{c+1}{r+c} \cdot \log p\right) \cdot \mathsf{poly}(\log N)$$

which proves Theorem 27.

Yet, as for Coppersmith's method, in practice instead of using such a large dimension $n = 2r(r + c)$, it is faster to use a lattice of reasonable dimension, and perform an exhaustive search in constant time on the most significant bits of the solutions. Namely, we consider polynomials $f_t(x) = f(X \cdot t + x)$ where

$$0 \leqslant t < p^{1-\frac{c+1}{r+c}}/X \quad \text{and} \quad X = \lfloor 2^{\lceil \log p \rceil (1-\frac{c}{r+c}-2\frac{r}{n})} \rfloor \quad .$$

Thus one has

$$0 \leqslant t \leqslant \lfloor 2^{\frac{2r}{n}-\frac{1}{r+c}} \rfloor \quad .$$

Yet, as before, the solution x_0 such that $x_0 = X \cdot t_0 + x_0'$ is obtained by finding the solution x_0' of the polynomial f_{t_0}. Therefore, the *LLL* algorithm is applied on each matrix B associated to each polynomial f_t, and the solution x_0 is found for the right polynomial f_{t_0}.

In outline, the whole BDH method is depicted in Algorithm 5.

Algorithm 5 BDH's Method for factoring $N = p^r q$

Input: An integer modulus N and a degree r such that $N = p^r q$. A rational c such that $q < p^c$. An approximation P of p such that $P = p + x_0$.

Output: All $x_0 \in \mathbb{Z}$ s.t. $|x_0| < p^{1-\frac{c+1}{r+c}}$ and $f(x_0) \equiv 0 \mod N$.

1: Let f be a univariate degree-r monic polynomial such that $f(x) = (P + x)^r$.
2: Let the dimension n be such that $\frac{n}{r+c} - \frac{1}{2}$ is within $\frac{1}{2r+c}$ of an integer.
3: Let $m \geq 1$ be an integer defined as $m = \left\lfloor \frac{n}{r+c} - \frac{1}{2} \right\rfloor$.
4: Let X be the bound $X = \lfloor 2^{\lceil \log p \rceil (1 - \frac{c}{r+c} - 2\frac{r}{n})} \rfloor$
5: **while** $Xt < 2^{\lceil \log p \rceil (1 - \frac{c+1}{r+c})}$ **do**
6: $f_t(x) = f(Xt + x) = (P + Xt + x)^r \in \mathbb{Z}[x]$.
7: Build the $n \times n$ lower-triangular matrix B whose rows are the $g_{i,k}(xX)$'s defined by (3.6).
8: Run the L^2 algorithm [NS09] on the matrix B.
9: The first vector of the reduced basis corresponds to a polynomial of the form $v(xX)$ for some $v(x) \in \mathbb{Z}[x]$.
10: Compute all the roots x_0' of the polynomial $v(x) \in \mathbb{Z}[x]$ over $\mathbb{Z}$.
11: Output $x_0 = Xt + x_0'$ for each root x_0' which satisfies $f_t(x_0') \equiv 0 \pmod N$ and $|x_0'| \leq X$.
12: $t \leftarrow t + 1$.
13: **end while**

As highlighted in Theorem 27, if none of the bits of p is known, then the exhaustive search can still be performed in constant time with a running time which is polynomial in $\log N$ if the degree r and the rational c are such that $r + c/c + 1 = \mathcal{O}(\log p)$. Thus, in this case, one simply replaces P by 0 in Algorithm 5, which leads to the use of the polynomial $f(x) = x^r$, where $|x_0| = p$ is the searched solution.

In summary, in this section we have recalled the BDH's method which allows to factorize moduli of the form $N = p^r q$ in polynomial time, if $r \simeq \log p$ and $q \leqslant p^{\mathcal{O}(1)}$.

A generalized Takagi-Cryptosystem with moduli of the form $N = p^r q^s$ has also been proposed at Indocrypt' 2000 in [LKYL00]. Namely, the authors show that the use of such moduli allows to considerably speed up the computation: for example using an 8196-bit modulus $N = p^2 q^3$ leads to a decryption which is 15 times faster than with a classical RSA modulus $N = pq$ of the same size. In the BDH paper the generalization of the attack to moduli of the form $N = p^r q^s$ where r and s are approximately the same size, is explicitly left as an open problem. Yet, in Chapter 5, we solve this open problem and we also propose a generalization to moduli $N = \prod_{i=1}^{k} p_i^{r_i}$ with more that two prime factors.

Part II

Contributions

Chapter 4

Rounding and Chaining LLL: Finding Faster Small Roots of Univariate Polynomial Congruences

Contents

The results presented in this chapter are from a joint work with Jingguo Bi, Jean-Sébastien Coron, Jean-Charles Faugère, Phong Q. Nguyen and Guénaël Renault. It was published in the proceedings of the PKC' 14 conference [BCF+14].

In Chapter 3.1, we have recalled Coppersmith's method which allows to find all small roots of univariate polynomial congruences of degree δ modulo an integer N of unknown factorization, in polynomial time in $(\log N, \delta)$. This has found many applications in public-key cryptanalysis and in a few security proofs. However, the running

time of the algorithm is a high-degree polynomial, which limits experiments. Indeed, as explained in Chapter 3.1.3, the bottleneck of Coppersmith's algorithm is the LLL-reduction of a high-dimensional matrix with extra-large coefficients. Namely, the complexity is $\mathcal{O}((\log^9 N)/\delta^2)$ if one uses the L^2 algorithm for the LLL-reduction step and it is $\mathcal{O}((\log^{7+\varepsilon} N)/\delta)$ if $\tilde{L}^1$ is rather used. We present in this chapter two speedups which are the first significant speedups over Coppersmith's algorithm.

The first speedup is based on a special property of the matrices used by Coppersmith's algorithm, namely, the diagonal elements in the matrix are all balanced. This property allows to provably speed up the LLL-reduction by rounding, that is by keeping only the most significant bits of all coefficients in the matrix before LLL-reducing it, so that the matrix contains much smaller elements. Once the LLL-reduction is done, the result is then rectified in order to reintegrate the least significant bits which were previously neglected. As we will show, this property of balancedness can also be solely used to improve the complexity analysis of Coppersmith's original algorithm. The exact speedup depends on the LLL-reduction algorithm used. Namely, the complexity becomes $\mathcal{O}(\log^7 N)$ if one uses the L^2 algorithm for the LLL-reduction step and it is $\mathcal{O}(\log^{6+\varepsilon} N)$ if $\tilde{L}^1$ is rather used. Thus, the speedup is asymptotically quadratic (resp. linear) in the bit-size of the small-root bound if one uses the L^2 (resp. the $\tilde{L}^1$) algorithm. Yet, a new feature of the complexity bound is that it becomes now independent of the degree δ. Indeed, the complexity only depends on the bit-size of the modulus N.

The second speedup is heuristic and applies whenever one wants to enlarge the root size of Coppersmith's algorithm by exhaustive search. Instead of performing several LLL-reductions independently, we exhibit relationships between these matrices so that the LLL-reductions can somewhat be chained to decrease the global running time.

As depicted in the sequel, when both speedups are combined, the new algorithm is in practice hundreds of times faster for typical parameters.

Roadmap. In Section 4.1, we present and analyze our first speedup of Coppersmith's algorithm: rounding LLL. In Section 4.2, we present and analyze our second speedup of Coppersmith's algorithm: chaining LLL. In Section 4.3, we provide experimental results with both speedups. Finally, we discuss the case of other small-root algorithms in Section 4.4.

State of the Art. In this Chapter, we make use of the reminder on lattice reduction given in Chapter 2. We will also frequently refer the reader to the original Coppersmith's method provided in Chapter 3.

4.1 Speeding up Coppersmith's Algorithm by Rounding

The original Coppersmith's method which allows to find small roots of univariate modular congruencies is recalled in Section 3.1. As shown there, the costliest step in the method is the LLL-reduction of the well-designed matrix B. Therefore, the complexity

of the method strongly depends on the LLL-reduction algorithm used. Thus, the complexity of Coppersmith's method is $\mathcal{O}((\log^9 N)/\delta^2)$ if one uses the L^2 algorithm and it is $\mathcal{O}((\log^{7+\varepsilon} N)/\delta)$ if $\tilde{L}^1$ is rather used (see Corollary 24).

Our first main result is the following complexity improvement over Coppersmith's algorithm:

Theorem 29. *There is an algorithm (namely, Algorithm 6) which, given as input an integer N of unknown factorization and a monic polynomial $f(x) \in \mathbb{Z}[x]$ of degree δ and coefficients in $\{0, \ldots, N-1\}$, outputs all integers $x_0 \in \mathbb{Z}$ such that $f(x_0) \equiv 0 \,(mod\, N)$ and $|x_0| \leq N^{1/\delta}$ in time $\mathcal{O}(\log^7 N)$ without fast integer arithmetic using the L^2 algorithm [NS09], or $\mathcal{O}(\log^{6+\varepsilon} N)$ for any $\varepsilon > 0$ using fast integer arithmetic and the $\tilde{L}^1$ algorithm [NSV11] in Step 7.*

As explained in Section 3.1, we recall that only the non trivial case where the degree δ is such that $2 < \delta + 1 < (\log N)/2$ will be implicitly considered in the sequel.

4.1.1 Rounding for Coppersmith's Algorithm

In Coppersmith's algorithm (Algorithm 4) an LLL-reduction of the matrix B is done in Step 5. This matrix has a dimension $n = \delta m + 1$, and its entries have bit-size $\mathcal{O}(m \log N)$. We explained that asymptotically, the dimension n is $\mathcal{O}(\log N)$ which gives $m = \mathcal{O}(\log N/\delta)$. Therefore, the bit-size of the elements is $\mathcal{O}((\log^2 N)/\delta)$. In our method, we will modify Coppersmith's algorithm in such a way that we only need to LLL-reduce a matrix of the same dimension but with much smaller entries. Namely the bit-length will be in $\mathcal{O}(\log N)$ instead of $\mathcal{O}((\log^2 N)/\delta)$.

To explain the intuition behind our method, let us first take a closer look at the matrix B and uncover some of its special properties. In particular, in the following lemma, we show that the maximal and the minimal diagonal coefficients are relatively close, which means that the diagonal elements are well-balanced.

Lemma 30. *Let $X \leq N^{1/\delta}$, $n = \delta m + 1$ with $m \geqslant 1$ and B be the Coppersmith's matrix defined in Step. 4 of Algorithm 4.*

- *The maximal diagonal coefficient of matrix B is $N^m X^{\delta-1} < N^{m+1}$.*
- *The minimal diagonal coefficient is $X^{\delta m} \leq N^m$.*
- *The ratio between the maximal and the minimal diagonal coefficient satisfies*

$$\frac{N^m X^{\delta-1}}{X^{\delta m}} \geq N^{1-1/\delta} \quad .$$

- *Furthermore, if $X \geq \Omega(N^{\frac{n+\delta-1}{\delta n}})$ and $n = \mathcal{O}(\log N)$, then the minimal diagonal coefficient is such that:*

$$X^{\delta m} \geq N^{m-\mathcal{O}(1)}. \tag{4.1}$$

Proof. The $n = \delta m + 1$ diagonal coefficients of B are naturally split into h blocks of δ coefficients with a last additional row. The i-th block is formed by the leading coefficients of the polynomials $g_{i,j}(xX)$ for $0 \leq j < \delta$ and the last row is formed by the leading coefficients of the polynomials $g_{m,0}(xX)$. Since the leading coefficient of $g_{i,j}(xX)$ is $X^j N^{m-i} X^{\delta i}$, it follows that the maximal and minimal coefficients in the i-th block are located respectively at the end (*i.e.* $j = 0$) and at the beginning (*i.e.* $j = \delta - 1$): their values are respectively

$$X^{\delta(i+1)-1} N^{m-i} = N^m (X^\delta/N)^i X^{\delta-1} \quad \text{and} \quad N^{m-i} X^{\delta i} = N^m (X^\delta/N)^i \quad .$$

If $X \leq N^{1/\delta}$, we obtain that the maximal diagonal coefficient is $N^m X^{\delta-1} < N^{m+1}$ reached in the 0-th block *i.e.* for $i = 0$, and the minimal diagonal coefficient is $X^{\delta m} \leqslant N^m$ reached in the last row, *i.e.* for $i = m$.

Furthermore, the ratio $N^m X^{\delta-1}/X^{\delta m}$ is greater than $N^{1-1/\delta}$ for $m \geqslant 1$. Indeed, since $X \leq N^{1/\delta}$, we have

$$\frac{N^m X^{\delta-1}}{X^{\delta m}} = \frac{N^m}{X^{\delta(m-1)+1}} \geqslant \frac{N^m}{N^{\frac{\delta(m-1)+1}{\delta}}} = \frac{N^m}{N^{(m-1)+\frac{1}{\delta}}} = N^{1-\frac{1}{\delta}} \quad .$$

Now, let $X_0 = N^{\frac{n-\delta+1}{\delta n}}$ so that $X = \Omega(X_0)$. One has

$$N^{\frac{n-\delta+1}{\delta n}} \;=\; N^{\frac{1}{\delta}-\frac{\delta-1}{\delta n}} \;\geqslant\; N^{\frac{1}{\delta}-\frac{\delta}{\delta n}} \;=\; N^{\frac{1}{\delta}-\frac{1}{n}} \quad .$$

Therefore we have $X_0 \geqslant N^{1/\delta - 1/n}$. Hence $X_0^\delta \geqslant N^{1-\delta/n}$ and thus $X_0^{\delta m}$ is such that

$$X_0^{\delta m} \geqslant N^{m-\frac{\delta m}{n}} = N^{m-\frac{n-1}{n}} > N^{m-1} \quad .$$

Since $X = \Omega(X_0)$ and $\delta m = \mathcal{O}(\log N)$, we obtain $X^{\delta m} \geqslant N^{m-\mathcal{O}(1)}$ which is (4.1). □

Lemma 30 implies that the diagonal coefficients of B are somewhat balanced. This means that the matrix B is not far from being reduced. In fact, the first row of B has norm N^m which is extremely close to the bound $N^m/\sqrt{n}$ required by Lemma 23. Intuitively, this means that it should not be too difficult to find a lattice vector shorter than $N^m/\sqrt{n}$.

To take advantage of the structure of B, we first size-reduce B (see Chapter 2, Definition 10) to make sure that in each column, all subdiagonal coefficients are smaller than the diagonal coefficient.

Then we round the entries of B so that the smallest diagonal coefficient becomes $\lfloor c \rfloor$ where $c > 1$ is a parameter. More precisely, we create a new $n \times n$ triangular matrix $\tilde{B} = (\tilde{b}_{i,j})$ defined by:

$$\tilde{B} = \left\lfloor cB/X^{n-1} \right\rfloor \quad . \tag{4.2}$$

This means that the new matrix $\tilde{B}$ is made of matrix B where all of its coefficients are divided by X^{n-1}/c.

By Lemma 30, the diagonal coefficients $b_{i,i}$ of matrices B are such that:

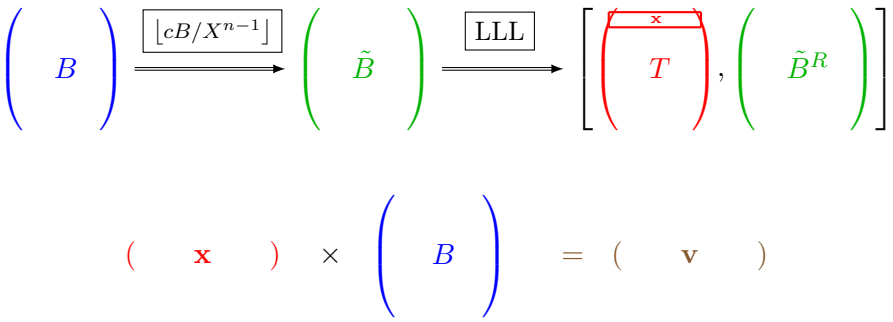
$$b_{i,i} \geqslant X^{\delta m} = X^{n-1} \quad ,$$

which gives that the diagonal coefficients $\tilde{b}_{i,i}$ of matrix $\tilde{B}$ are such that:

$$\tilde{b}_{i,i} \geqslant \lfloor cX^{n-1}/X^{n-1} \rfloor = \lfloor c \rfloor \, .$$

Hence, we LLL-reduce the rounded matrix $\tilde{B}$ instead of B. Let $\tilde{\mathbf{v}}$ be the first vector of the reduced basis obtained $\tilde{B}^R$. This vector $\tilde{\mathbf{v}}$ is in fact the result of the product of a transformation vector $\mathbf{x}$ (which is the first vector of the transformation matrix output by LLL) with the matrix $\tilde{B}$. This means that we have $\tilde{\mathbf{v}} = \mathbf{x}\tilde{B}$. We then apply the transformation vector $\mathbf{x}$ to the matrix B in order to obtain a short vector $\mathbf{v}$. Those steps are illustrated in Figure 4.1 and they correspond to Steps 6 to 9 in Algorithm 6.

Figure 4.1: Rounding-LLL: the rounded matrix $\tilde{B} = \lfloor cB/X^{n-1} \rfloor$ is LLL-reduced. Then the transformation vector $\mathbf{x}$ is applied to the matrix B, which gives a short vector $\mathbf{v}$.



More generally, if we applied to B the unimodular transformation T that LLL-reduces $\tilde{B}$, we may not even obtain an LLL-reduced basis in general. However, because of the special structure of B, it turns out that by applying the transformation vector $\mathbf{x}$ to the matrix B, the vector $\mathbf{v} = \mathbf{x}B$ is still a short non-zero vector of L, as shown below:

Lemma 31. *Let $B = (b_{i,j})$ be an $n \times n$ lower-triangular matrix over $\mathbb{Z}$ with strictly positive diagonal. Let $c > 1$. If $\tilde{B} = \lfloor cB/\min_{i=1}^{n} b_{i,i} \rfloor$ and $\mathbf{x}\tilde{B}$ is the first vector of an LLL-reduced basis of $\tilde{B}$, then:*

$$0 < \|\mathbf{x}B\| < \left(n\|\tilde{B}^{-1}\|_2 + 1\right) 2^{\frac{n-1}{4}} \det(B)^{\frac{1}{n}} \, .$$

Proof. Let $\alpha = \min_{i=1}^{n} b_{i,i}/c$, so that $\tilde{B} = \lfloor B/\alpha \rfloor$. Define the matrix $\bar{B} = \alpha\tilde{B}$ whose entries are $\bar{b}_{i,j} = \alpha\tilde{b}_{i,j}$. Therefore the elements $b_{i,j}$ are relatively close to the elements $\bar{b}_{i,j}$. Namely we have $0 \leqslant b_{i,j} - \bar{b}_{i,j} < \alpha$. This gives the relation

$$\|B - \bar{B}\|_2 < n\alpha \quad .$$

Thus we have:

$$\|\mathbf{x}B\| \leqslant \|\mathbf{x}(B-\bar{B})\| + \|\mathbf{x}\bar{B}\| \leqslant \|\mathbf{x}\| \times \|B-\bar{B}\|_2 + \alpha\|\mathbf{x}\tilde{B}\| < n\|\mathbf{x}\|\alpha + \alpha\|\mathbf{x}\tilde{B}\|.$$

Let $\tilde{\mathbf{v}} = \mathbf{x}\tilde{B}$. Then $\|\mathbf{x}\| \leqslant \|\tilde{\mathbf{v}}\|\|\tilde{B}^{-1}\|_2$, and we obtain

$$\|\mathbf{x}B\| < \left(n\|\tilde{B}^{-1}\|_2 + 1\right)\alpha\|\tilde{\mathbf{v}}\| \quad .$$

The matrix $\tilde{B}$ is lower-triangular with all diagonal coefficients strictly positive because $c > 1$. Since $\tilde{\mathbf{v}} = \mathbf{x}\tilde{B}$ is the first vector of an LLL-reduced basis of $\tilde{B}$, and $\tilde{B}$ is non-singular, we have $\mathbf{x}B \neq 0$, which gives $\|\mathbf{x}B\| > 0$p and:

$$\|\tilde{\mathbf{v}}\| \leqslant 2^{\frac{n-1}{4}} \det(\tilde{B})^{\frac{1}{n}} \quad .$$

Therefore we deduce that

$$\alpha\|\tilde{\mathbf{v}}\| \leqslant \alpha 2^{\frac{n-1}{4}} \det(\tilde{B})^{\frac{1}{n}} = 2^{\frac{n-1}{4}} \det(\bar{B})^{\frac{1}{n}} \leqslant 2^{\frac{n-1}{4}} \det(B)^{\frac{1}{n}},$$

where we used the fact that matrices $\tilde{B}$, $\bar{B}$ and B are lower-triangular. To conclude, the result follows by combining both inequalities:

$$\|\mathbf{x}B\| < \left(n\|\tilde{B}^{-1}\|_2 + 1\right) 2^{\frac{n-1}{4}} \det(B)^{\frac{1}{n}} \quad .$$

□

If $\mathbf{x}B$ is sufficiently short, then it corresponds to a polynomial of the form $v(xX)$ for some $v(x) \in \mathbb{Z}[x]$ satisfying Lemma 23, and the rest proceeds as in Algorithm 4.

The whole rounding algorithm is given in Algorithm 6, which will be shown to admit a lower complexity upper-bound than Algorithm 4 to compute all roots up to $N^{1/\delta}$.

We now justify the bound X given in Algorithm 6. In order for Lemma 31 to be useful, we need to exhibit an upper bound for $\|\tilde{B}^{-1}\|_2$. Later on, we will see that the upper bound for $\|\tilde{B}^{-1}\|_2$ depends on the upper-bound of the inverse of a triangular matrix. Therefore, we already need to prove the following elementary lemma from which one can derive an upper bound on inverses of triangular matrices.

Lemma 32. *Let $t > 0$ and $T = (t_{i,j})$ be an $n \times n$ lower-triangular matrix over $\mathbb{R}$, with unit diagonal (*i.e. *$t_{i,i} = 1$ for $1 \leq i \leq n$), and such that $|t_{i,j}| \leq t$ for $1 \leq j < i \leq n$. Then the matrix T satisfies*

$$\|T^{-1}\|_\infty \leq (1+t)^{n-1} \quad .$$

Proof. Let $S = T^{-1}$. Then for $1 \leq i, j \leq n$ we have $\sum_{k=j}^{n} s_{i,k}t_{k,j} = \delta_{i,j}$, where $\delta_{i,j}$ is Kronecker's symbol. Therefore one has

$$s_{i,j} = \delta_{i,j} - \sum_{k=j+1}^{n} s_{i,k}t_{k,j} \quad .$$

Algorithm 6 Coppersmith's Method with Rounding

Input: Two integers $N \geq 1$ and $m \geqslant 1$, a univariate degree-δ monic polynomial $f(x) \in \mathbb{Z}[x]$ with coefficients in $\{0, \ldots, N-1\}$ and $2 < \delta + 1 < (\log N)/2$.

Output: All $x_0 \in \mathbb{Z}$ s.t. $|x_0| \leq N^{1/\delta}$ and $f(x_0) \equiv 0 \mod N$.

1: Let $n = \delta m + 1$, X the bound given in Theorem 34, $c = (3/2)^n$ and $t = 0$.
2: **while** $Xt < N^{1/\delta}$ **do**
3: $f_t(x) = f(Xt + x) \in \mathbb{Z}[x]$.
4: Build the $n \times n$ matrix B whose rows are the $g_{i,j}(xX)$'s defined by (3.1).
5: Size-reduce B without modifying its diagonal coefficients.
6: Compute the matrix $\tilde{B} = \lfloor cB/X^{n-1} \rfloor$ obtained by rounding B.
7: Run the L^2 algorithm [NS09] on the matrix $\tilde{B}$.
8: Let $\tilde{\mathbf{v}} = \mathbf{x}\tilde{B}$ be the first vector of the reduced basis obtained.
9: The vector $\mathbf{v} = \mathbf{x}B$ corresponds to a polynomial of the form $v(xX)$ for some $v(x) \in \mathbb{Z}[x]$.
10: Compute all the roots x_0' of the polynomial $v(x) \in \mathbb{Z}[x]$ over $\mathbb{Z}$.
11: Output $x_0 = x_0' + Xt$ for each root x_0' which satisfies $f_t(x_0') \equiv 0 \pmod N$ and $|x_0'| \leq X$.
12: $t \leftarrow t + 1$.
13: **end while**

This implies that S is lower-triangular and for $1 \leq j < i \leq n$ one has:

$$|s_{i,j}| \leq t \left(1 + \sum_{k=j+1}^{i-1} |s_{i,k}| \right). \tag{4.3}$$

Let us prove that for all $j < i$, $|s_{i,j}| \leq t(1+t)^{i-j-1}$, by induction over $i-j$. Since $|t_{i,j}| \leq t$ for $1 \leq j < i \leq n$, one has $|s_{i,i-1}| \leq t|s_{i,i}| = t$, which starts the induction for $i - j = 1$. Now, assume by induction that $|s_{i,k}| \leq t(1+t)^{i-k-1}$ for all k such that $i - k < i - j$ for some $1 \leq j < i \leq n$. Then (4.3) implies:

$$\begin{aligned} |s_{i,j}| &\leq t \left(1 + \sum_{k=j+1}^{i-1} t(1+t)^{i-k-1} \right) = t \left(1 + t \sum_{k=0}^{i-j-2} (1+t)^k \right) \\ &= t \left(1 + t \frac{(1+t)^{i-j-1} - 1}{1+t-1} \right) = t(1+t)^{i-j-1} \end{aligned}$$

which completes the induction. Hence, we have:

$$\|S\|_\infty \leq 1 + \sum_{i=2}^{n} t(1+t)^{i-2} = 1 + t \sum_{i=0}^{n-2} (1+t)^i = (1+t)^{n-1}.$$

□

One can now exhibit an upper bound for $\|\tilde{B}^{-1}\|_2$, as given in the following lemma.

Lemma 33. *Let $B = (b_{i,j})$ be an $n \times n$ size-reduced lower-triangular matrix over $\mathbb{Z}$ with strictly positive diagonal. Let $c > 1$. If $\tilde{B} = \lfloor cB / \min_{i=1}^{n} b_{i,i} \rfloor$, then:*

$$\|\tilde{B}^{-1}\|_2 \leq \sqrt{n} \left(\frac{3c-2}{2c-2}\right)^{n-1} / \lfloor c \rfloor \,.$$

Proof. The matrix $\tilde{B}$ is lower-triangular like B. Because B is size-reduced, the entries of $\tilde{B}$ satisfy, for $1 \leq j < i \leq n$:

$$\frac{|\tilde{b}_{i,j}|}{\tilde{b}_{j,j}} < \frac{|b_{i,j}|/2^k}{b_{j,j}/2^k - 1} \leq \frac{1}{2} \times \frac{1}{1 - 2^k/b_{j,j}} \leq \frac{1}{2} \times \frac{1}{1 - 1/c}$$

This means that $\tilde{B}$ is almost size-reduced. Let Δ be the $n \times n$ diagonal matrix whose i-th diagonal entry is $1/\tilde{b}_{i,i}$. Then $T = \Delta\tilde{B}$ satisfies the conditions of Lemma 32 with $t = 1/(2(1-1/c))$. Therefore we get:

$$\|T^{-1}\|_\infty \;\; \leq \;\; \left(\frac{1}{2} \times \frac{1}{1-1/c} + 1\right)^{n-1} = \left(\frac{3c-2}{2c-2}\right)^{n-1} .$$

Furthermore, we have

$$\|\tilde{B}^{-1}\|_2 \;\leq\; \sqrt{n}\|\tilde{B}^{-1}\|_\infty \;\leq\; \sqrt{n}\|T^{-1}\|_\infty \|\Delta\|_\infty \;\; .$$

Therefore we deduce that:

$$\|\tilde{B}^{-1}\|_2 \;\leq\; \sqrt{n}\left(\frac{3c-2}{2c-2}\right)^{n-1} \times \frac{1}{\min_{1\leq i\leq n} \tilde{b}_{i,i}} \;\leq\; \sqrt{n}\left(\frac{3c-2}{2c-2}\right)^{n-1} / \lfloor c \rfloor,$$

which concludes the proof.

□

By combining Lemmas 31 and 33, we obtain the following small-root bound X for Algorithm 6:

Theorem 34. *Given as input two integers $N \geq 1$ and $m \geqslant 1$, a rational $c > 1$, and a univariate degree-δ monic polynomial $f(x) \in \mathbb{Z}[x]$ with coefficients in $\{0, \ldots, N-1\}$, one loop of Algorithm 6, corresponding to $t < N^{1/\delta}/X$, outputs all $x_0 = Xt + x_0' \in \mathbb{Z}$ s.t. $|x_0'| \leq X$ and $f(x_0) = 0 \mod N$, where $n = \delta m + 1$ and*

$$X = \left\lfloor \frac{N^{\frac{n+\delta-1}{\delta n}} \kappa_1^{-2/(n-1)}}{\sqrt{2}\, n^{1/(n-1)}} \right\rfloor \quad \textit{with} \quad \kappa_1 = n^{3/2} \left(\frac{3c-2}{2c-2}\right)^{n-1} \lfloor c \rfloor^{-1} + 1 \,. \tag{4.4}$$

Proof. Combining Lemma 33 with Lemma 31 where $\det(B)^{1/n} = N^{\frac{(n-1)(m+1)}{2n}} X^{\frac{n-1}{2}}$, we get

$$0 < \|\mathbf{x}B\| < \kappa_1 2^{\frac{n-1}{4}} N^{\frac{(n-1)(m+1)}{2n}} X^{\frac{n-1}{2}} \;\; .$$

It follows from Lemma 23 that the polynomial $v(x)$ holds over the integers if $\|v(xX)\| = \|\mathbf{x}B\| < N^m/\sqrt{n}$. This gives the following condition on X:

$$\kappa_1 2^{\frac{n-1}{4}} N^{\frac{(n-1)(m+1)}{2n}} X^{\frac{n-1}{2}} < N^m/\sqrt{n}\,.$$

which can be rewritten

$$X < \frac{N^{\frac{2m}{n-1}-\frac{m+1}{n}} \kappa_1^{-2/(n-1)}}{\sqrt{2}\, n^{1/(n-1)}}\,.$$

As already seen in Chapter 3, one has:

$$\frac{2m}{n-1} - \frac{m+1}{n} = \frac{2}{\delta} - \frac{m+1}{n} = \frac{2n-\delta(m+1)}{\delta n} = \frac{2n-(n-1)-\delta}{\delta n} = \frac{n-\delta+1}{\delta n}\quad .$$

Therefore, the bound given in (4.4) is straightforwardly obtained from the last inequality. □

The bound X of Theorem 34 is never larger than that of Corollary 24. However, if one selects $c \geqslant (3/2)^n$, then the two bounds are asymptotically equivalent. This is why Algorithm 6 uses $c = (3/2)^n$.

4.1.2 Running time: proof of Theorem 29

The original matrix B had entries whose bit-size was $\mathcal{O}(m \log N)$. Let $\beta = \frac{N^m X^{\delta-1}}{X^{n-1}}$ be the ratio between the maximal diagonal coefficient and the minimal diagonal coefficient of $\tilde{B}$. If B is size-reduced, the entries of the new matrix $\tilde{B} = \lfloor cB/X^{n-1} \rfloor$ are upper bounded by $c\beta$.

By Lemma 30, we know that if $m \geqslant 1$, then $\beta \geq N^{1-1/\delta}$, and if further $X \geq \Omega(N^{\frac{2m}{n-1}-\frac{m+1}{n}})$ and $\delta m = \mathcal{O}(\log N)$, then $\beta = N^{\mathcal{O}(1)}$. Hence, the bit-size b of $\tilde{B}$'s entries is such that

$$b \leqslant \log c + \mathcal{O}(\log N)\quad .$$

Furthermore, the dimension of $\tilde{B}$ is the same as B, *i.e.* $n = \delta m + 1$. It follows that the running time of L^2 in Step 7 is $\mathcal{O}(\delta^6 m^6 (\log c + \log N) + \delta^5 m^5 (\log c + \log N)^2)$ without fast integer arithmetic, which is

$$L^2: \quad \mathcal{O}((\delta m)^5 (\log c + \log N)^2) = \mathcal{O}((\log c + \log N)^7)$$

because $\delta < (\log N)/2 - 1$ and $\delta m = \mathcal{O}(\log N)$. The running time using $\tilde{L}^1$ in Step 7 is $\mathcal{O}((\delta m)^{5+\varepsilon}(\log c + \log N) + (\delta m)^{\omega+1+\varepsilon}(\log c + \log N)^{1+\varepsilon})$ for any $\varepsilon > 0$ using fast integer arithmetic, where $\omega \leq 2.376$ is the matrix multiplication complexity constant, which gives the complexity:

$$\tilde{L}^1: \quad \mathcal{O}((\delta m)^{5+\varepsilon}(\log c + \log N) + (\delta m)^{\omega+1+\varepsilon}(\log c + \log N)^{1+\varepsilon}) = \mathcal{O}((\log c + \log N)^{6+\varepsilon})\quad .$$

This leads to our main result (Theorem 29), which is a variant of Coppersmith's algorithm with improved complexity upper bound. More precisely, as in Coppersmith's algorithm,

one can easily prove that the number of loops performed in Algorithm 6 is at most constant. Indeed, when $c = (3/2)^n$, then $\kappa_1^{-2/n-1}$ converges to 1. This means that the bound X achieved by Theorem 34 is asymptotically equivalent to the one achieved by Corollary 24, which completes the proof of Theorem 29, because $\log c = \mathcal{O}(\log N)$ when $c = (3/2)^n$.

4.1.3 A Remark on the Original Coppersmith's Complexity

By simply analyzing Coppersmith's matrix, and without even performing the Rounding-LLL improvement, the complexity upper bounds of Corollary 24 with L^2 and $\tilde{L}^1$ can already actually be decreased. Indeed, Lemma 30 uncovers the special property of balancedness of Coppersmith's matrix, which implies that

$$\mathcal{O}\left(\frac{max\|\mathbf{b}_i^\star\|}{min\|\mathbf{b}_i^\star\|}\right) = \mathcal{O}(N) \quad .$$

Therefore, according to Theorem 18, the number τ of loop iterations of the LLL-reduction algorithm on the input basis used by Coppersmith's algorithm is

$$\tau = \mathcal{O}\left(n^2 \log \frac{max\|\mathbf{b}_i^\star\|}{min\|\mathbf{b}_i^\star\|}\right) = \mathcal{O}(n^2 \log N) = \mathcal{O}(\log^3 N) \quad ,$$

by using that $n = \mathcal{O}(\log N)$. This number τ of loop iterations replaces the all-purpose bound $\mathcal{O}(n^2 b) = \mathcal{O}(n^2 m \log N) = \mathcal{O}(\log^4 N/\delta)$ [DV94].

L^2 Algorithm:

By taking this observation into account, the complexity of the L^2 algorithm becomes: $\mathcal{O}(n^3\tau(n+b)) = \mathcal{O}(n^3 \log^3 N(n+b)) = \mathcal{O}(\log^6 N(n+b))$ instead of $\mathcal{O}(n^3 \log^4 N(n+b)/\delta) = \mathcal{O}(\log^7 N(n+b)/\delta)$. Since according to Lemma 30 the maximal diagonal element is bounded by N^{m+1}, the bit-size b of the elements in matrix B is bounded by $\mathcal{O}(m \log N) = \mathcal{O}(\log^2 N/\delta)$. This yields the complexity $\mathcal{O}(\log^6 N(n+b)) = \mathcal{O}(\log^8 N/\delta)$, instead of the previous $\mathcal{O}((\log^9 N)/\delta^2)$. Yet the Rounding-$LLL$ improvement, which can be easily implemented, is based on the same special property of Coppersmith's matrix and it allows to continue decreasing the complexity down to $\mathcal{O}(\log^7 N)$ as highlighted in Theorem 29.

$\tilde{L}^1$ Algorithm:

Surprisingly, Lemma 30 also allows to prove that the $\tilde{L}^1$ algorithm, when carefully analyzed using the balancedness of the Gram-Schmidt norms, already achieves the complexity bound $\mathcal{O}(\log^{6+\varepsilon} N)$ given in Theorem 29. Indeed, using Theorem 6 from [NSV11] which gives the $\tilde{L}^1$ complexity upper bound $\mathcal{O}(n^{3+\varepsilon}\tau) = \mathcal{O}(\log^{3+\varepsilon} N\tau)$ where τ is the total number of iterations, and combining it with [DV94] applied to Coppersmith's matrix (Lemma 30), which gives $\tau = \mathcal{O}(n^2 \log N) = \mathcal{O}(\log^3 N)$, allows to retrieve the above

complexity $\mathcal{O}(\log^{6+\varepsilon} N)$. However, we have proposed in this section a direct improvement of Coppersmith's method based on elementary tools and which can therefore be easily implemented on usual computer algebra systems (*e.g.* Sage, Magma, NTL) with immediate practical impact on cryptanalyses. Furthermore, we are not aware of any implementation of the $\tilde{L}^1$ algorithm for the time being, which makes a practical comparison tricky.

4.1.4 A Summary of the Complexities

As a summary, the obtained asymptotical complexities are provided in Table 4.1, as a function of the LLL-reduction algorithm employed (L^2 or $\tilde{L}^1$). More precisely, the original complexity of Coppersmith's algorithm is given in the first row for comparison. The second row represents the refined complexity by taking into account the link between the number of iterations of the LLL-reduction algorithm and the balancedness of the Gram-Schmidt norms. Eventually, the third row highlights the complexities obtained by the application of the Rounding method. Thus, when using the L^2 algorithm, the global asymptotical speed up $\Theta((\log^2 N)/\delta^2)$ is quadratic in the bit-size of the small-root bound $N^{1/\delta}$ and the speed up $\Theta((\log N)/\delta)$ is linear when $\tilde{L}^1$ is rather employed.

Table 4.1: Coppersmith algorithm complexity by taking into account the original analysis, the refined analysis and the Rounding method. The complexities depend on the LLL-reduction algorithm employed (L^2 or $\tilde{L}^1$).

	L^2	$\tilde{L}^1$
Original Analysis	$\mathcal{O}((\log^9 N)/\delta^2)$	$\mathcal{O}((\log^{7+\varepsilon} N)/\delta)$
Refined Analysis	$\mathcal{O}((\log^8 N)/\delta)$	$\mathcal{O}(\log^{6+\varepsilon} N)$
Rounding Method	$\mathcal{O}(\log^7 N)$	$\mathcal{O}(\log^{6+\varepsilon} N)$

Eventually, we emphasize that this work helps to clarify the asymptotical complexity of Coppersmith's algorithm for univariate polynomial congruences regarding the dependence on the polynomial degree δ. In the original Coppersmith's paper [Cop97] the complexity is stated as polynomial in $(\log N, 2^\delta)$, but it is well known that the 2^δ is a typo and the complexity is polynomial in δ only (see our analysis in Chapter 3). However, our final complexity upper bound using the Rounding method becomes independent of δ: it only depends on the bit-size of the modulus N.

In next section, we present a method that allows to speed up the exhaustive search which is performed to reach Coppersmith's bound $N^{1/\delta}$.

4.2 Chaining LLL

As recalled in Section 3.1, in order to find all solutions which are close to the bound $N^{1/\delta}$, one should not use a very large lattice dimension (*i.e.* $n = \mathcal{O}(\log N)$). Instead, it is better to use a lattice of reasonable dimension and to perform exhaustive search on the most significant bits of x until finding all solutions. Namely, we consider polynomials:

$$f_t(x) = f(X \cdot t + x) \quad \text{where} \quad 0 \leqslant t < \frac{N^{1/\delta}}{X} \quad \text{and} \quad X = \lfloor 2^{-1/2} N^{\frac{n-\delta+1}{\delta n}} n^{-\frac{1}{n-1}} \rfloor \ .$$

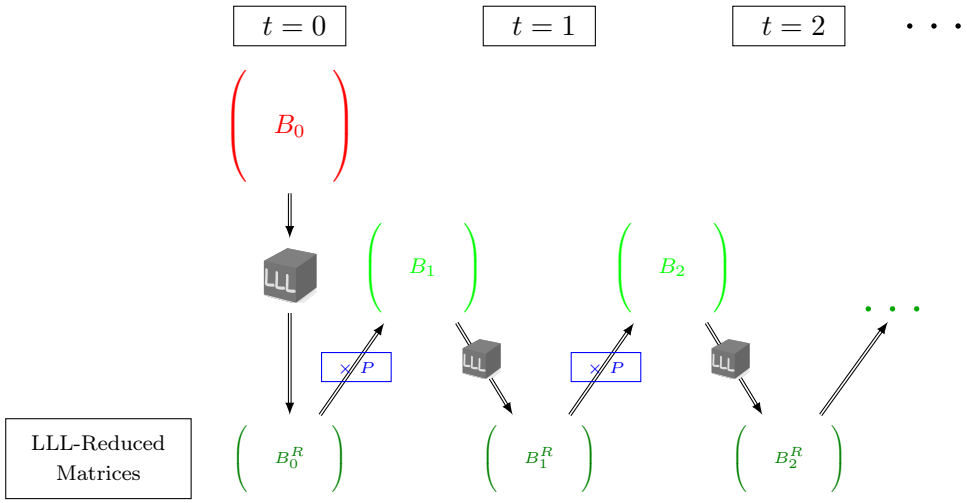
Thus, an initial solution x_0 that can be written $x_0 = X \cdot t_0 + x'_0$ is obtained by finding the solution x'_0 of the polynomial f_{t_0}. In this case, this solution satisfies $|x'_0| < X$ and it has a correct size for LLL to find it using a lattice of dimension n. For each polynomial f_t, one runs LLL on a certain matrix (Step 4 of Algorithm 6). The solution x_0 is then found for the right value t_0 , *i.e.* for the right polynomial f_{t_0}.

In Section 4.2.1, we describe a method that allows to take advantage of the LLL performed for the case $t = i$ to reduce (in practice) the complexity of the LLL performed for the case $t = i + 1$. The method is based on a hidden relationship between Coppersmith's lattices. More precisely, we show how to easily construct a matrix which is equivalent to the Coppersmith's one for the case $t = i + 1$ from a matrix of the case $t = i$. One enlightens the fact that in order to solve polynomial f_i, one can use matrix $B_0 \cdot P^i$ (instead of matrix used in Step 4 of Algorithm 6), for different instances $t = 0, \ldots, i$ those matrices can be linked one to another by the relation

$$B_i \ = \ B_{i-1} \cdot P \ = \ B_{i-2} \cdot P^2 \ = \ \ldots \ = \ B_0 \cdot P^i \ ,$$

where P is a well-known structured matrix. Our method consists in LLL-reducing B_0, which gives B_0^R. Then, instead of LLL-reducing $B_0 \cdot P$, we apply LLL on $B_1 = B_0^R \cdot P$. We expect this matrix to be almost reduced already since it is the product of an LLL-reduced matrix B_0^R with a matrix P containing small coefficients. This gives matrix B_1^R. Next step consists in applying LLL on $B_2 = B_1^R \cdot P$ instead of $B_0 \cdot P^2$. Thus, we perform this process incrementally until all solutions are found, as illustrated in Figure 4.2.

Figure 4.2: Chaining-LLL: New exhaustive search scheme within Coppersmith's method. A first matrix B_0 is LLL-reduced, then the matrices are chained by the application of P and successively LLL-reduced.



Thereafter, in Section 4.2.2 we combine this improvement with the rounding approach described in Section 4.1.

4.2.1 Exploiting Relations Between Consecutive Lattices

The following proposition discloses a connection between the lattice used for the case $t = i$ and the next lattice used for $t = i + 1$. This connection is based on the well-known *Pascal matrix* $P = (p_{s,j})$ defined as the $n \times n$ lower-triangular matrix whose non-zero coefficients are the binomials: $p_{s,j} = \binom{s}{j}$ for $0 \leq j \leq s \leq n - 1$. Namely, we have:

$$P = \begin{pmatrix} 1 & & & & & & \\ 1 & 1 & & & & & \\ 1 & 2 & 1 & & & & \\ 1 & 3 & 3 & 1 & & & \\ 1 & 4 & 6 & 4 & 1 & & \\ 1 & 5 & 10 & 10 & 5 & 1 & \\ \vdots & & \cdots & & \cdots & & \ddots \\ 1 & n-1 & \binom{n-1}{2} & \cdots & \binom{n-1}{j} & \cdots & \binom{n-1}{2} & n-1 & 1 \end{pmatrix}.$$

Proposition 35. *Let B be a basis of the n-dimensional lattice used by Coppersmith's algorithm to find all small roots of the polynomial $f_i(x) = f(X \cdot i + x)$, where X is the small-root bound. Then $B \cdot P$ is a basis of the "next" lattice used for the polynomial $f_{i+1}(x)$.*

Proof. Because all lattice bases are related by some unimodular matrix, it suffices to prove the statement for a special basis B. We thus only consider the special basis $B = B_i$ formed by the n shifted polynomials constructed from $f_i(x)$ and written in the basis

$$\mathcal{B} = (1, xX^{-1}, (xX^{-1})^2, \ldots, (xX^{-1})^{n-1}) \quad .$$

For the case $t = i + 1$, one tries to solve the polynomial

$$f_{i+1}(x) = f(X \cdot (i+1) + x) = f(X \cdot i + x + X) = f_i(x + X)\,.$$

Therefore, the shifted polynomials constructed from f_{i+1} are the same as for the case $t = i$, but written in the different basis

$$\mathcal{B}' = (1, xX^{-1} + 1, (xX^{-1} + 1)^2, \ldots, (xX^{-1} + 1)^{n-1}) \quad .$$

Yet, we need to return to the original representation of the polynomials, *i.e.* in the basis $\mathcal{B}$. To this end, we use the following property regarding the lower triangular Pascal matrix P:

$$\mathcal{B}'^T = P \cdot \mathcal{B}^T\,.$$

Namely, we have:

$$\begin{pmatrix} 1 \\ \frac{x+X}{X} \\ \left(\frac{x+X}{X}\right)^2 \\ \vdots \\ \left(\frac{x+X}{X}\right)^{n-1} \end{pmatrix} \quad = \quad P \quad \times \quad \begin{pmatrix} 1 \\ \frac{x}{X} \\ \left(\frac{x}{X}\right)^2 \\ \vdots \\ \left(\frac{x}{X}\right)^{n-1} \end{pmatrix}$$

As a consequence, left-multiplying each side of this equality by the matrix B_i proves that the matrix $B_i \cdot P$ is a basis of the lattice used for finding small roots of the polynomial $f_{i+1}(x)$.

□

The proposition allows us to use different matrices to tackle the polynomial $f_{i+1}(x)$ than the one initially used by Coppersmith's method. In particular, we can use a matrix of the form $B^R \cdot P$ where B^R is an LLL-reduced basis of the previous lattice used to solve $f_i(x)$: intuitively, it might be faster to LLL-reduce such matrices than the initial Coppersmith's matrix. Although we are unable to prove the lattice reduction will be faster, we can show that the vectors of such a matrix are not much longer than that of B^R:

Corollary 36. *Let B_i^R be the LLL-reduced matrix used for solving f_t for $t = i$ and P be the Pascal matrix. The matrix*

$$B_{i+1} = B_i^R \cdot P$$

spans the same lattice used for solving the case $t = i + 1$. This matrix consists of vectors $\mathbf{b_{i+1},}_j$ whose norms are close to vector norms of the LLL-reduced matrix B_i^R. Namely, for all $1 \leqslant j \leqslant n$ we have:

$$||\mathbf{b_{i+1},}_j|| < \sqrt{n} \cdot 2^{n-1} \cdot ||\mathbf{b_i^R}_{,j}|| \,.$$

In particular, for the case $i = t_0$ the first vector of B_{i+1} has a norm bounded by $2^{n-1} \cdot N^m$.

Proof. The previous proposition immediately gives the first statement. Since the matrix B_{i+1} is the product of B_i^R with a matrix P composed of relatively short elements, the elements in B_{i+1} remain close to those in the reduced matrix B_i^R. Indeed, the largest element in P is $\binom{n-1}{\lceil (n-1)/2 \rceil}$. By a property of binomial coefficients, we have $\binom{a}{\lceil a/2 \rceil} \leqslant 2^{a-1}$ for $a \geqslant 1$. Therefore the largest element in P is smaller than 2^{n-2}. More precisely, the maximal norm of column vectors in P is reached in the $\lfloor (n-1)/2 \rfloor$-th column and is smaller than

$$\sqrt{2^{2\cdot(\frac{n-2}{2}-1)} + 2^{2\cdot(\frac{n-2}{2})} + \cdots + 2^{2\cdot(n-2)}} < \sqrt{2^{2n-3}} < 2^{n-1} \,.$$

Therefore the norm of each row vector of B_{i+1} is at most enlarged by a factor $\sqrt{n} \cdot 2^{n-1}$ compared to the norm of the corresponding vector in B_i^R, *i.e.* for all $1 \leqslant j \leqslant n$ we have $||\mathbf{b_{i+1},}_j|| < \sqrt{n} \cdot 2^{n-1} \cdot ||\mathbf{b_i^R}_{,j}||$. In particular, for $i = t_0$, since the first vector of B_i^R has a norm bounded by $N^m/\sqrt{n}$, the norm of the first vector of $B_i^R \cdot P$ is bounded by $2^{n-1} \cdot N^m$ which is relatively close to $N^m/\sqrt{n}$.

□

Corollary 36 shows us that vectors of B_{i+1} are relatively close to the ones in the LLL-reduced matrix B_i^R. Thus, we intuitively expect the LLL-reduction of B_{i+1} to be less costly than the one of the original Coppersmith's matrix. However, our bounds are too weak to rigorously prove this. Yet, one can use this property iteratively to elaborate a new method which *chains* all LLL-reductions as follows. First, one LLL-reduces B_0 for the case $t = 0$. This gives a reduced matrix B_0^R. Then, one performs a multiplication by P and an LLL-reduction of $B_1 = B_0^R \cdot P$, which gives B_1^R. We then iterate this process by performing LLL on $B_{i+1} = B_i^R \cdot P$ (for $i \geq 0$) to obtain B_{i+1}^R and so forth until all solutions are found (each time by solving the polynomial corresponding to the first vector of B_i^R).

In the sequel, we study this *chaining method* by performing similar roundings as in Section 4.1 before each call of LLL-reduction.

4.2.2 Rounding and Chaining LLL

During the exhaustive search described in Section 4.2.1, we perform the LLL algorithm on the matrix $B_{i+1} = B_i^R \cdot P$ for $0 \leqslant i < N^{1/\delta}/X$, where B_i^R is LLL-reduced.

It is worth noticing that the structure of B_i^R and thereby of B_{i+1}, is different from the original Coppersmith's matrix B_0 (in particular, it is not triangular anymore). Yet, we are able to show that under certain conditions on B_{i+1} verified experimentally, one can combine the rounding technique of Section 4.1 with the chaining technique of Section 4.2.1. Indeed, we show that during the chaining loop, one can size-reduce B_{i+1} and then round its elements for all $i \geq 0$ as follows:

$$\tilde{B}_{i+1} = \left\lfloor cB_{i+1} / \min_{1 \leq i \leq n} \|\mathbf{b}_i^\star\| \right\rfloor \ , \tag{4.5}$$

where $\mathbf{b}_i^\star$ are Gram-Schmidt vectors of B_{i+1} and c is a rational that will be determined later. Then, one applies LLL on the rounded matrix $\tilde{B}_{i+1}$ as performed in Section 4.1. We obtain an LLL-reduced matrix $\tilde{B}_{i+1}^R$ and a unimodular matrix $\tilde{U}_{i+1}$ such that

$$\tilde{U}_{i+1} \cdot \tilde{B}_{i+1} = \tilde{B}_{i+1}^R \ .$$

Then one shows that by applying $\tilde{U}_{i+1}$ on B_{i+1}, the first vector of this matrix $\tilde{U}_{i+1} \cdot B_{i+1}$ is a short vector that allows to find the solutions provided that they are smaller than a bound X that will be determined latter.

For the sake of clarity, in the sequel we denote by B the matrix B_{i+1}, and by $\mathbf{x}B$, the first vector of matrix $\tilde{U}_{i+1} \cdot B_{i+1}$.

We would like to exhibit an upper-bound on $\|\mathbf{x}B\|$. To this end, we will need, as in Section 4.1, to upper-bound the value $\|\tilde{B}^{-1}\|_2$. This is done in the following lemma:

Lemma 37. *Let $B = (b_{i,j})$ be an $n \times n$ non-singular integral matrix and $\alpha \geq 1$ such that $n\alpha\|B^{-1}\|_2 < 1$. Then the matrix $\tilde{B} = \lfloor B/\alpha \rfloor$ is invertible with:*

$$\|\tilde{B}^{-1}\|_2 \leq \frac{\alpha\|B^{-1}\|_2}{1 - n\alpha\|B^{-1}\|_2} \, .$$

Proof. Let again $\bar{B} = \alpha\tilde{B}$, which implies that $\|B - \bar{B}\|_2 < n\alpha$. Since $n\alpha\|B^{-1}\|_2 < 1$, we have $\rho = \|B^{-1}\Delta B\|_2 < 1$, where $\Delta B = \bar{B} - B$.

Theorem 2.3.4 in the book [GVL13] by Golub and Loan gives a bound on inverses for perturbed matrices. Namely, it states that if we have $\rho = \|B^{-1}\Delta B\|_2 < 1$, then $B + \Delta B$ is non-singular with:

$$\|(B + \Delta B)^{-1} - B^{-1}\|_2 \leq \frac{\|\Delta B\|_2 \|B^{-1}\|_2^2}{1 - \rho} \tag{4.6}$$

Therefore, in our case, one deduces that $\bar{B}$ is invertible with:

$$\|\bar{B}^{-1} - B^{-1}\|_2 \leq \frac{\|\Delta B\|_2 \|B^{-1}\|_2^2}{1 - \rho} \leq \frac{n\alpha\|B^{-1}\|_2^2}{1 - n\alpha\|B^{-1}\|_2} \ .$$

Hence one deduces the following upper-bound on $\|\bar{B}^{-1}\|_2$:

$$\|\bar{B}^{-1}\|_2 \leq \|\bar{B}^{-1} - B^{-1}\|_2 + \|B^{-1}\|_2 \leq \|B^{-1}\|_2 + \frac{n\alpha\|B^{-1}\|_2^2}{1 - n\alpha\|B^{-1}\|_2} = \frac{\|B^{-1}\|_2}{1 - n\alpha\|B^{-1}\|_2}$$

Since $\bar{B} = \alpha\tilde{B}$, we have $\tilde{B}^{-1} = \alpha\bar{B}^{-1}$. Therefore $\|\tilde{B}^{-1}\|_2 = \alpha\|\bar{B}^{-1}\|_2$, which concludes the proof.

□

As one can see, this value depends on $\|B^{-1}\|_2$ which is given in Lemma 38.

Lemma 38. *Let B be an $n \times n$ non-singular size-reduced matrix, with Gram-Schmidt vectors $\mathbf{b}_i^\star$. Then:*

$$\|B^{-1}\|_2 \leq \frac{\sqrt{n}(3/2)^{n-1}}{\min_{1\leq i\leq n}\|\mathbf{b}_i^\star\|}\,.$$

Proof. Using the Gram-Schmidt factorization, we have $B = \mu DQ$. Therefore $B^{-1} = Q^{-1}D^{-1}\mu^{-1}$ and

$$\|B^{-1}\|_2 \leq \|D^{-1}\|_2\|\mu^{-1}\|_2 \quad .$$

Because B is size-reduced, we can apply Lemma 32 on μ and $t = 1/2$, which proves that $\|\mu^{-1}\|_\infty \leq (3/2)^{n-1}$. Hence:

$$\|B^{-1}\|_2 \leq \|D^{-1}\|_2\sqrt{n}(3/2)^{n-1},$$

where $\|D^{-1}\|_2 = 1/\min_{1\leq i\leq n}\|\mathbf{b}_i^\star\|$.

□

One can now give an upper-bound on $\|\mathbf{x}B\|$:

Corollary 39. *Let $B = (b_{i,j})$ be an $n \times n$ size-reduced non-singular matrix over $\mathbb{Z}$. Let $\alpha \geq 1$ such that $n^2\alpha\|B^{-1}\|_2 < 1$. Then $\tilde{B} = \lfloor cB/\min_{1\leq i\leq n}\|\mathbf{b}_i^\star\|\rfloor = \lfloor B/\alpha\rfloor$ is non-singular. And if $\mathbf{x}\tilde{B}$ is the first vector of an LLL-reduced basis of $\tilde{B}$, then:*

$$0 \;<\; \|\mathbf{x}B\| \;<\; \kappa_2\; 2^{\frac{n-1}{4}}\;\det(B)^{\frac{1}{n}},$$

where

$$\kappa_2 = \frac{c^{\frac{n+1}{n}}}{(c - n^{3/2}(3/2)^{n-1})(c - n^{5/2}(3/2)^{n-1})^{1/n}} \quad .$$

Proof. The proof follows Lemma 31 proof. The major differences being that the bounds on $\|B^{-1}\|_2$ and $\|\tilde{B}^{-1}\|_2$ differ (see Lemmas 37 and 38), and that $\det(\tilde{B})$ is not straightforwardly predictable anymore since matrix B is no more triangular.

Here starts the proof. We have $\tilde{B} = \lfloor B/\alpha\rfloor$. Define the matrix $\bar{B} = \alpha\tilde{B}$ whose entries are $\bar{b}_{i,j} = \alpha\tilde{b}_{i,j}$. Since $n^2\alpha\|B^{-1}\|_2 < 1$, Lemma 37 implies that $\tilde{B}$ is non-singular. Furthermore, we have:

$$\|\mathbf{x}B\| \leq \|\mathbf{x}(B - \bar{B})\| + \|\mathbf{x}\bar{B}\| \leq \|\mathbf{x}\| \times \|B - \bar{B}\|_2 + \alpha\|\mathbf{x}\tilde{B}\| < n\|\mathbf{x}\|\alpha + \alpha\|\mathbf{x}\tilde{B}\|.$$

Let $\tilde{\mathbf{v}} = \mathbf{x}\tilde{B}$. Then $\|\mathbf{x}\| \leq \|\tilde{\mathbf{v}}\|\|\tilde{B}^{-1}\|_2$, and we obtain:

$$\|\mathbf{x}B\| < \left(n\|\tilde{B}^{-1}\|_2 + 1\right)\alpha\|\tilde{\mathbf{v}}\|.$$

Since $\tilde{\mathbf{v}} = \mathbf{x}\tilde{B}$ is the first vector of an LLL-reduced basis of $\tilde{B}$, and $\tilde{B}$ is non-singular, $\mathbf{x}B \neq 0$ and we have:

$$\alpha\|\tilde{\mathbf{v}}\| \leq \alpha 2^{\frac{n-1}{4}} \det(\tilde{B})^{\frac{1}{n}} = 2^{\frac{n-1}{4}} \det(\bar{B})^{\frac{1}{n}}$$

The condition $n^2\alpha\|B^{-1}\|_2 < 1$ implies that $n\|B^{-1}\|_2\|\bar{B} - B\|_2 < 1$.

Godunov *et al.* [GAKK93] give a bound on determinants for perturbed matrices. Namely, they show that if $n\|B^{-1}\|_2\|\Delta B\|_2 < 1$, where B is an $n \times n$ non-singular matrix and ΔB is a "small" $n \times n$ perturbation, then one has:

$$\frac{|\det(B + \Delta B) - \det B|}{|\det B|} \leq \frac{n\|B^{-1}\|_2\|\Delta B\|_2}{1 - n\|B^{-1}\|_2\|\Delta B\|_2} \tag{4.7}$$

In our case, since $\bar{B} = B + (\bar{B} - B)$, this implies that:

$$\frac{|\det(\bar{B}) - \det B|}{|\det B|} \leq \frac{n\|B^{-1}\|_2\|\bar{B} - B\|_2}{1 - n\|B^{-1}\|_2\|\bar{B} - B\|_2} \leq \frac{n^2\alpha\|B^{-1}\|_2}{1 - n^2\alpha\|B^{-1}\|_2} \quad .$$

It follows that:

$$|\det(\bar{B})| \leq |\det B|\left(1 + \frac{n^2\alpha\|B^{-1}\|_2}{1 - n^2\alpha\|B^{-1}\|_2}\right) = \frac{|\det B|}{1 - n^2\alpha\|B^{-1}\|_2} \quad .$$

Therefore, we get the following inequality:

$$\|\mathbf{x}B\| < \left(n\|\tilde{B}^{-1}\|_2 + 1\right) 2^{\frac{n-1}{4}} (1 - n^2\alpha\|B^{-1}\|_2)^{-1/n} \det(B)^{\frac{1}{n}} \quad . \tag{4.8}$$

Using Lemmas 38 and 37, which respectively give the upper bounds

$$\|B^{-1}\|_2 \;\leq\; \frac{\sqrt{n}(3/2)^{n-1}}{\min_{1\leq i\leq n}\|\mathbf{b}_i^\star\|} \qquad \text{and} \qquad \|\tilde{B}^{-1}\|_2 \;\leq\; \frac{\alpha\|B^{-1}\|_2}{1 - n\alpha\|B^{-1}\|_2} \quad ,$$

and combining them in (4.8), one gets

$$\|\mathbf{x}B\| < \frac{\min_{1\leq i\leq n}\|\mathbf{b}_i^\star\|^{\frac{n+1}{n}}\; 2^{\frac{n-1}{4}}\; \det(B)^{\frac{1}{n}}}{(\min_{1\leq i\leq n}\|\mathbf{b}_i^\star\| - n^{3/2}\alpha(3/2)^{n-1})(\min_{1\leq i\leq n}\|\mathbf{b}_i^\star\| - n^{5/2}\alpha(3/2)^{n-1})^{1/n}} \quad .$$

Eventually, one replaces α by $\min_{1\leq i\leq n}\|\mathbf{b}_i^\star\|/c$ in previous inequality. This allows to get the bound stated in Corollary 39.

□

Again, if $\|\mathbf{x}B\|$ is sufficiently short, then it corresponds to a polynomial of the form $v(xX)$ for some $v(x) \in \mathbb{Z}[x]$ satisfying Lemma 23. In particular, for the case $t = t_0$, solving this polynomial equation would allow to retrieve the solution x_0. Note that the condition $n^2\alpha\|B^{-1}\|_2 < 1$ specified in Corollary 39 gives a condition on the rational c. Indeed, since $\alpha = \min_{1\leq i\leq n} \|\mathbf{b}_i^\star\|/c$ and using Lemma 38, one gets:

$$n^2\alpha\|B^{-1}\|_2 \leqslant n^2 \frac{\min_{1\leq i\leq n} \|\mathbf{b}_i^\star\|}{c} \frac{\sqrt{n}(3/2)^{n-1}}{\min_{1\leq i\leq n} \|\mathbf{b}_i^\star\|} \leqslant \frac{n^{5/2}(3/2)^{n-1}}{c} < 1\,,$$

that is c should be such that $c > n^{5/2}(3/2)^{n-1}$.

The whole chaining and rounding algorithm is depicted in Algorithm 7. Note that in practice, we do not need to perform Step 8 of Algorithm 7 and that $\min_{1\leq i\leq n} \|\mathbf{b}_{\mathbf{t}+\mathbf{1}_i}^\star\|$ can be estimated instead of being computed in Step 9 (see Section 4.2.3 for more details).

In the following, we give a small-root bound X on the solution x_0' sufficient to guarantee success:

Theorem 40. *Given as input two integers $N \geq 1$ and $m \geqslant 1$, a rational $c > n^{5/2}(3/2)^{n-1}$, and a univariate degree-δ monic polynomial $f(x) \in \mathbb{Z}[x]$ with coefficients in $\{0, \ldots, N-1\}$, one loop of Algorithm 7, corresponding to $t < N^{1/\delta}/X$, outputs all $x_0 = Xt + x_0' \in \mathbb{Z}$ s.t. $|x_0'| \leq X$ and $f(x_0) = 0 \mod N$, and $n = \delta m + 1$, where*

$$X = \left\lfloor \frac{N^{\frac{n-\delta+1}{\delta n}} \kappa_2^{\frac{-2}{n-1}}}{\sqrt{2}\, n^{1/(n-1)}} \right\rfloor\,, \tag{4.9}$$

and κ_2 is the value defined in Corollary 39.

Proof. During the chaining-LLL method, all matrices B satisfy the property

$$\det(B)^{1/n} = N^{\frac{(n-1)(m+1)}{2n}} X^{\frac{n-1}{2}}\,,$$

since the Pascal matrix has determinant equal to 1.

Furthermore, it follows from Lemma 23 that the polynomial $v(x)$ holds over the integers if $\|v(xX)\| = \|\mathbf{x}B\| < N^m/\sqrt{n}$. Therefore, using Corollary 39, one gets the condition:

$$\kappa_2\, 2^{\frac{n-1}{4}}\, N^{\frac{(n-1)(m+1)}{2n}} X^{\frac{n-1}{2}} < N^m/\sqrt{n}\,,$$

which can be rewritten

$$X < \frac{N^{\frac{2m}{n-1} - \frac{m+1}{n}} \kappa_2^{-2/(n-1)}}{\sqrt{2}\, n^{1/(n-1)}}\,.$$

As before, since we have $\frac{2m}{n-1} - \frac{m+1}{n} = \frac{n-\delta+1}{\delta n}$, the bound given in (4.9) follows.

□

The bound X of Theorem 40 is never larger than that of Corollary 24. However, if one selects $c > n^{5/2}(3/2)^{n-1}$, then the two bounds are asymptotically equivalent. This is why Algorithm 7 uses $c = n^{5/2}(3/2)^n$.

Algorithm 7 Coppersmith's Method with Chaining and Rounding

Input: Two integers $N \geq 1$ and $m \geqslant 1$, a univariate degree-δ monic polynomial $f(x) \in \mathbb{Z}[x]$ with coefficients in $\{0, \ldots, N-1\}$ and $2 < \delta + 1 < (\log N)/2$.

Output: All $x_0 \in \mathbb{Z}$ s.t. $|x_0| \leq N^{1/\delta}$ and $f(x_0) \equiv 0 \mod N$.

1: Perform Step 1 and Steps 3 to 7 of Algorithm 6. Step 7 returns $\tilde{B}_0^R$ and $\tilde{U}_0$ such that $\tilde{U}_0 \cdot \tilde{B}_0 = \tilde{B}_0^R$.
2: Let $n = \delta m + 1$ and X be the bound given in Theorem 40. Let the rationnal c be such that $c = n^{\frac{5}{2}}(\frac{3}{2})^n$ and the loop iteration counter $t = 0$. Let P be the $n \times n$ lower triangular Pascal matrix.
3: Compute the matrix $\tilde{U}_0 \cdot B_0$, where B_0 is the matrix computed in Step 5 of Algorithm 6.
4: The first vector of $\tilde{U}_0 \cdot B_0$ corresponds to a polynomial of the form $v(xX)$ for some $v(x) \in \mathbb{Z}[x]$.
5: Compute and output all roots $x_0 \in \mathbb{Z}$ of $v(x)$ satisfying $f(x_0) \equiv 0 \,(\mathrm{mod}\, N)$ and $|x_0| \leq X$.
6: **while** $Xt < N^{1/\delta}$ **do**
7: Compute the matrix $B_{t+1} = \tilde{U}_t \cdot B_t \cdot P$.
8: Size-reduce B_{t+1}.
9: Compute the matrix $\tilde{B}_{t+1} = \lfloor cB_{t+1} / \min_{1 \leq i \leq n} \|\mathbf{b_{t+1}}_i^\star\| \rfloor$ obtained by rounding B_{t+1}.
10: Run L^2 algorithm on matrix $\tilde{B}_{t+1}$ which returns $\tilde{B}_{t+1}^R$ and $\tilde{U}_{t+1}$ s.t. $\tilde{U}_{t+1} \cdot \tilde{B}_{t+1} = \tilde{B}_{t+1}^R$.
11: Compute the matrix $\tilde{U}_{t+1} \cdot B_{t+1}$.
12: The first vector of $\tilde{U}_{t+1} \cdot B_{t+1}$ corresponds to a polynomial of the form $v(xX)$.
13: Compute all the roots x_0' of the polynomial $v(x) \in \mathbb{Z}[x]$ over $\mathbb{Z}$.
14: Output $x_0 = x_0' + Xt$ for each root x_0' which satisfies $f(x_0' + Xt) \equiv 0 \,(\mathrm{mod}\, N)$ and $|x_0'| \leq X$.
15: $t \leftarrow t + 1$.
16: **end while**

4.2.3 Complexity Analysis: A Heuristic Approach

The complexity of Algorithm 7 relies on the complexity of the LLL-reduction performed in Step 10. The cost of this reduction depends on the size of coefficients in matrix $B = \tilde{B}_{t+1}$, which itself depends on the value $\min_{1 \leq i \leq n} \|\mathbf{b}_i^\star\|$. The exact knowledge of this value does not seem straightforward to obtain without computing the Gram-Schmidt matrix explicitly. However, experiments show that the Gram-Schmidt curve is roughly decreasing, *i.e.* $\min_{1 \leq i \leq n} \|\mathbf{b}_i^\star\| \approx \|\mathbf{b}_n^\star\|$ and is roughly symmetric, *i.e.*

$$\log \|\mathbf{b}_i^\star\| - \log \|\mathbf{b}_{\lfloor n/2 \rfloor}^\star\| \approx \log \|\mathbf{b}_{\lfloor n/2 \rfloor}^\star\| - \log \|\mathbf{b}_{n-i+1}^\star\| \quad .$$

If we assume these two experimental facts, we deduce that $\|\mathbf{b}_{\lfloor n/2 \rfloor}^\star\| \approx |\det(B)|^{1/n}$. By duality, this means that $\|\mathbf{b}_n^\star\| \approx |\det(B)|^{2/n} / \|\mathbf{b}_1^\star\|$. Furthermore, from the definition of

the GSO, we know that $\|\mathbf{b}_1^\star\| = \|\mathbf{b}_1\|$, where $\mathbf{b}_1$ is the first vector of matrix B. Therefore we have:

$$\min_{1\leq i\leq n} \|\mathbf{b}_i^\star\| \approx \|\mathbf{b}_n^\star\| \approx \frac{|\det(B)|^{2/n}}{\|\mathbf{b}_1^\star\|} = \frac{N^{\frac{(n-1)(m+1)}{n}} X^{n-1}}{\|\mathbf{b}_1\|}\,, \tag{4.10}$$

Thus, we need an estimation on $\|\mathbf{b}_1\|$. Since in practice, the matrix $B = B_{i+1} = \tilde{U}_i \cdot B_i \cdot P$ is already nearly size-reduced, one can skip Step 8 of Algorithm 7. Therefore, vector $\mathbf{b}_1$ is the first vector of matrix $\tilde{U}_i \cdot B_i \cdot P$. Using Corollary 39, one deduces that the first vector of matrix $\tilde{U}_i \cdot B_i$ is roughly as short as the first vector of an LLL-reduced matrix. From the well-known experimental behavior of LLL [NS06], we can model the first vector of the LLL-reduced basis as a "random" vector of norm $\approx 1.02^n |\det(B)|^{1/n}$ (where 1.02 has to be replaced by a smaller constant for dimension $n \leq 60$) . Since the Pascal matrix P has a norm smaller than 2^{n-1} (see proof of Corollary 36), one gets the bound $\|\mathbf{b}_1\| \leqslant \sqrt{n} 2^{n-1} 1.02^n |\det(B)|^{1/n}$. Therefore, we deduce that: $\min_{1\leq i\leq n} \|\mathbf{b}_i^\star\| \approx |\det(B)|^{1/n}/(\sqrt{n}2^{n-1}1.02^n)$. In practice, we conjecture (see Figure 4.3 in Section 4.3) that

$$\min_{1\leq i\leq n} \|\mathbf{b}_i^\star\| > \frac{|\det(B)|^{1/n}}{\beta^n} \qquad \text{where} \quad \beta < 2 \quad .$$

This discussion leads to the following heuristic approach regarding the method: firstly, one should rather use the estimation (4.10) in Step 9 of Algorithm 7, instead of explicitly computing the Gram-Schmidt matrix; secondly, one can skip Step 8 of Algorithm 7. This heuristic version of Algorithm 7 is the one we used during our experiments, all these assumptions were always verified.

To conclude our analysis, since as specified before, our experiments gave

$$\max_{1\leq i\leq n} \|\mathbf{b}_i^\star\| \approx \|\mathbf{b}_1^\star\| \approx \frac{|\det(B)|^{2/n}}{\|\mathbf{b}_n^\star\|} \approx \frac{|\det(B)|^{2/n}}{\min_{1\leq i\leq n} \|\mathbf{b}_i^\star\|} \quad ,$$

one gets

$$\max_{1\leq i\leq n} \|\mathbf{b}_i^\star\| \approx \frac{|\det(B)|^{2/n}}{|\det(B)|^{1/n}} \beta^n \approx |\det(B)|^{1/n} \beta^n \quad .$$

Therefore, instead of reducing a matrix such that $\max_{1\leq i\leq n} \|\mathbf{b}_i^\star\| \leqslant |\det(B)|^{1/n}\beta^n$, it suffices to reduce a rounded matrix such that

$$\max_{1\leq i\leq n} \|\tilde{\mathbf{b}}_i^\star\| \leqslant c \frac{\max_{1\leq i\leq n} \|\mathbf{b}_i^\star\|}{\min_{1\leq i\leq n} \|\mathbf{b}_i^\star\|} \leqslant c\beta^{2n} \quad ,$$

This means that we are trading entries of size $\mathcal{O}(n)$ instead of $\mathcal{O}(m \log N)$. Therefore, by considering $n = \mathcal{O}(\log N)$, we obtain the same complexity as in Theorem 29 but in a heuristic way. However, even if both asymptotic complexities are identical, in practice for reasonable dimensions the speed-up brought by using Algorithm 7 rather than Algorithm 6 is considerable (see Section 4.3). Indeed, the LLL-reduction of matrix $\tilde{U}_i \cdot B_i \cdot P$ (Step 10 of Algorithm 7) performs surprisingly faster than expected. This comes from the fact that for reasonable dimensions, the Gram-Schmidt curve of this matrix remains quite close to the one of matrix $\tilde{U}_i \cdot B_i$, where $\tilde{U}_i \cdot B_i$ turns out to be LLL-reduced

(or nearly). Besides, the overall running-time of Algorithm 7 is approximately the time spent to perform one LLL-reduction, multiplied by the number of executed loops, *i.e.* by $N^{1/\delta}/X$.

4.3 Experiments

4.3.1 Practical Considerations

We implemented Coppersmith's algorithm (Algorithm 4) and our improvements (Algorithms 6 and 7) using Magma Software V2.19-5 for N being 1024-bit and 2048-bit moduli. Our test machine is a 3.20-GHz Intel Xeon. Running times are given in seconds.

We used polynomials of the form

$$f(x) = x^2 + ax + b \equiv 0 \mod N \quad \text{with degree } \delta = 2 \ .$$

According to Coppersmith's Theorem, one can retrieve the solution x_0 if $x_0 < N^{1/2}$. More precisely, Algorithm 4, with $n = 2m + 1$, can find all roots x_0 as long as

$$|x'_0| \leq X = \lfloor 2^{-\frac{1}{2}} N^{\frac{n-\delta+1}{\delta n}} n^{-\frac{1}{n-1}} \rfloor = \lfloor 2^{-\frac{1}{2}} N^{\frac{1}{2}-\frac{1}{2n}} n^{-\frac{1}{n-1}} \rfloor \ .$$

For a fixed n, the rounding strategy (Algorithm 6) gives a worse bound than X, but the difference can be made arbitrarily small by increasing the parameter c: in our experiments, we therefore chose the smallest value of c such that $\kappa_1^{\frac{-2}{n-1}}$ and $\kappa_2^{\frac{-2}{n-1}}$ are larger than 0.90, so that the new bound is never less than the old bound X by more than 10%, which is essentially the same. However, we note that the value c can be taken smaller in practice: indeed, our theoretical analysis was a worst-case analysis. For instance, it has been proved in [VT98] that if T is a random $n \times n$ lower-triangular matrix with unit diagonal and subdiagonal coefficients normally distributed, then $(\|\|T^{-1}\|_2)^{1/n}$ converges to $1.3057\ldots$ And experimentally, when subdiagonal coefficients are uniformly distributed over $[-1/2, +1/2]$, then we have $(\|\|T^{-1}\|_\infty)^{1/n} \leq 1.1$ with high probability. This means that the constants of Lemma 32 (and therefore the implicit 3/2 in formulas for c) are better in practice.

Furthermore, it is worth noticing that since the value α is not significant in itself, in order to increase the efficiency, one can round matrices at negligible cost by taking $\alpha := 2^{\lfloor \log_2(\alpha) \rfloor}$ and performing shifts of $\lfloor \log_2(\alpha) \rfloor$ bits.

In the same vein, one can increment t by 2 instead of 1 in Step 9 of Algorithm 4 or in Step 12 of Algorithm 6, and one can multiply the matrix $\tilde{U}_i \cdot B_i$ by P^2 instead of P in Step 7 of Algorithm 7. This comes from the fact that if $0 < x'_0 < X$ (resp. $-X < x'_0 < 0$), then $x'_0 - X$ (resp. $x'_0 + X$) is also a valid solution. This refinement allows to divide by 2 the global timing of Algorithms 4 and 6. However, it seems to be much less relevant when applied to Algorithm 7.

4.3.2 Implementation Results

We have performed several tests depending on the dimension n. Results are depicted in Table 6.1 for the case $\lceil \log_2(N) \rceil = 1024$ and in Table 6.2 for the case $\lceil \log_2(N) \rceil = 2048$. In both tables, the bit-size of Coppersmith's theoretic upper-bound $X = N^{1/2}$ is given in last column. We have noted the bit-size of the bound X' associated to a dimension n for which the solution x'_0 is found in practice. We give corresponding timings for different applications:

— Time for one LLL execution on the original Coppersmith's matrix B_0 (given for comparison only: this reduction is never performed in our method). This corresponds to Step 5 of Algorithm 4.

— Time for one LLL execution on the first truncated Coppersmith's matrix (applied to reduce the first matrix B'_0 only). This corresponds to Step 7 of Algorithm 6.

— Time for one LLL execution on the truncated matrix which is quasi LLL-reduced (applied on $\tilde{B'}_i$ for $i = 1, ..., t_0$ during the exhaustive search). This corresponds to Step 10 of Algorithm 7.

— Time for the multiplication with the unimodular matrix ($U_i \cdot \tilde{B}_i$ performed for $i = 1, ..., t_0$ during the exhaustive search after each LLL computation of matrix $\tilde{B'}_i$). This corresponds to Step 11 of Algorithm 7.

Note that the cost of solving a univariate equation over $\mathbb{Z}$ is not given since it turns out to be negligible in practice. Running times are given as averages over 5 samples.

Table 4.2: Bounds and running time (in seconds if not specified) as a function of the dimension for $\lceil \log_2(N) \rceil = 1024$.

$\log_2(\mathbf{X'})$		492	496	500	503	**504**	505	$\log_2(X) = 512$
Dimension		29	35	51	71	**77**	87	N/A
Original Method	LLL (B_0)	10.6	35.2	355	2338	**4432**	11426	N/A
	Total Timing (days)	128.6 d.	26.7 d.	16.8 d.	13.9 d.	**13.1 d.**	16.9 d.	N/A
New Method	Truncated LLL (B'_0)	1.6	3.5	18.8	94	**150**	436	N/A
	Trunc. Exhaus. LLL ($\tilde{B}'_i$)	0.04	0.12	1.4	9.9	**15.1**	46.5	N/A
	Multiplication Unimodular	0.04	0.08	0.4	1.2	**1.7**	3.6	N/A
	Total Timing (hours)	23.3 h.	3.6 h.	2.1 h.	1.6 h.	**1.2 h.**	1.9 h.	N/A

As depicted in Tables 6.1 and 6.2, by increasing the dimension, one can retrieve solutions x_0 that get ever closer to $X = N^{1/2}$. However, beyond a certain point, it is not profitable to increase the dimension since an exhaustive search would end up faster. In our case, the best dimension to use is depicted in bold on both tables. Indeed, one can see that using a larger dimension allows to find a solution which is one bit longer only, for LLL executions that take more than twice as much time.

As a consequence, for $\lceil \log_2(N) \rceil = 1024$, the best trade-off is to use lattices of dimension 77, and perform an exhaustive search on $512 - 504 = 8$ bits. As depicted in

Table 4.3: Bounds and running time (in seconds if not specified) as a function of the dimension for $\lceil \log_2(N) \rceil = 2048$.

$\log_2(\mathbf{X}')$		994	1004	1007	1011	**1012**	1013	$\log_2(X) = 1024$
Dimension		35	51	63	85	**91**	101	N/A
Original Method	LLL (B_0)	164	1617	5667	39342	**60827**	125498	N/A
	Total Timing (years)	5584 y.	53.8 y.	23.6 y.	10.2 y.	**7.9 y.**	8.2 y.	N/A
New Method	Truncated LLL (B'_0)	9	48	146	825	**1200**	2596	N/A
	Trunc. Exhaus. LLL ($\tilde{B}'_i$)	0.15	1.6	6.2	33	**48**	104	N/A
	Multiplication Unimodular	0.12	0.6	1.5	5.4	**6.5**	11.5	N/A
	Total Timing (days)	3355 d.	26.7 d.	11.7 d.	3.7 d.	**2.6 d.**	2.8 d.	N/A

Table 4.4, the exhaustive search then takes $150 + (2^8 - 1)(15.1 + 1.7) \approx 1.2$ hours, which is about 262 times faster than the original method which takes $2^8 \times 4432 \approx 13.1$ days. Somehow, this represents the global speedup obtained by using Algorithm 7 rather than Algorithm 4. More specifically, when the exhaustive search is not considered, performing a single LLL execution takes 150 seconds when truncating the matrix (Algorithm 6), compared to 4332 seconds using the original method Algorithm 4.

In the same way, for $\lceil \log_2(N) \rceil = 2048$, the best trade-off is to use lattices of dimension 91, and perform an exhaustive search on $1024 - 1012 = 12$ bits. Again, as depicted in Table 4.4, the exhaustive search then takes $1200 + (2^{12} - 1)(48 + 6.5) \approx 2.6$ days, which is about 1109 times faster than the original method which takes $2^{12} \times 60827 \approx 7.9$ years. More specifically, when the exhaustive search is not considered, performing a single LLL execution takes 1200 seconds when truncating the matrix (Algorithm 6), compared to 60827 seconds using the original method (Algorithm 6).

Table 4.4: Global exhaustive search timing using original/new methods for $\lceil \log_2(N) \rceil = 1024$ and 2048.

	$\lceil \log_2(N) \rceil = 512$	$\lceil \log_2(N) \rceil = 1024$	$\lceil \log_2(N) \rceil = 1536$	$\lceil \log_2(N) \rceil = 2048$
Original method	47 minutes	13.1 days	108.5 days	7.9 years
New method	52 seconds	1.2 hours	5.2 hours	2.6 days
Speed up	54	262	502	1109

Yet, one recovers the fact that the speed up of the rounding method (Algorithm 6) is linear in $m = (n-1)/\delta$ (same as obtained in the theoretical analysis), and we obtain even more speedups by using the rounding and chaining method (Algorithm 7). Hence, our improvement is practical and allows to get much closer to the asymptotic small-root bound. Moreover, as depicted in Table 4.4, the larger the modulus N, the more significant the speed-up of Algorithm 7.

Furthermore, we verify the assumption on the value $\min_{1\leq i\leq n}\|b_i^\star\|$ for matrix B. We write $\max_{1\leq i\leq n}\|\mathbf{b}_i^\star\| \approx \beta_1^n \mathrm{vol}(L)^{1/n}$ and $\min_{1\leq i\leq n}\|\mathbf{b}_i^\star\| \approx \beta_2^n \mathrm{vol}(L)^{1/n}$. In this paper, we have assumed that $\beta_1 = 1/\beta_2$. We summarize the results of our experiments for $\lceil \log N \rceil = 512$ with dimensions $30, 60, 90, 120, 150$ in Table 4.3. We can see that $\beta_1 \times \beta_2 \approx 1$ and that $\beta_1 \leq 2$. This means our assumptions are reasonable.

Figure 4.3: Beta values for $\lceil \log N \rceil = 512$

Data	Parameter m				
type	10	20	30	40	50
β_1	1.7582	1.8751	1.9093	1.9218	1.9435
β_2	0.5460	0.5271	0.5155	0.5091	0.5077
product	0.9600	0.9883	0.9842	0.9785	0.9867

4.4 Other Small-Root Algorithms

We now discuss whether our rounding method can similarly speed up other small-root algorithms (see the surveys [May10, Ngu09]), which are based on the same main ideas where *LLL*-reduction plays a crucial role. In theory, the rounding method provides a speedup for any triangular matrix whose diagonal coefficients are all large. However, in order to have a large speedup, we need the minimal diagonal coefficient to be much larger than the ratio between the maximal diagonal coefficient and the minimal diagonal coefficient. In Coppersmith's algorithm, the smallest diagonal coefficient was $N^{m-O(1)}$, while the gap was $N^{O(1)}$, which translated into a polynomial speedup. As we see in the sequel, it turns out that other small-root algorithms do not share the same features: we only get a (small) constant speedup.

4.4.1 Gcd Generalization

Coppersmith's algorithm (Algorithm 4) has been essentially generalized by Howgrave-Graham [HG01] and Boneh *et al.* [BDHG99] (see the surveys [May10, Ngu09]) as follows:

Theorem 41. *There is an algorithm which, given as input an integer N of unknown factorization, a rational α s.t. $0 < \alpha \leq 1$ and a monic polynomial $f(x) \in \mathbb{Z}[x]$ of degree δ and coefficients in $\{0, \ldots, N-1\}$, outputs all integers $x_0 \in \mathbb{Z}$ such that $\gcd(f(x_0), N) \geq N^\alpha$ and $|x_0| \leq N^{\alpha^2/\delta}$ in time polynomial in $\log N$, δ and the bit-size of α.*

Theorem 22 is then the special case $\alpha = 1$ of Theorem 41. The algorithm underlying Theorem 41 is in fact very similar to Algorithm 4: instead of applying Lemma 23 with

N^m, one uses p^m where $p \geq N^\alpha$ is some unknown divisor of N. And one considers the same family of polynomials

$$g_{i,j}(x) = x^j N^{m-i} f^i(x) \quad ,$$

but over slightly different indices. Algorithm 4 used $0 \leq i < m$ and $0 \leq j < \delta$, and $j = 0$ for $i = m$. This time, we use the two following sets of indices:

$$1) \quad 0 \leq i < m \ \text{ and } \ 0 \leq j < \delta, \qquad 2) \quad i = m \ \text{ and } \ 0 \leqslant j < \gamma \quad ,$$

where γ is chosen asymptotically to be such that:

$$\gamma = \lfloor \delta m/(\alpha - 1) \rfloor \quad .$$

Then the dimension is $n = \delta m + \gamma$. The maximal diagonal coefficient is still $N^m X^{\delta-1}$, and the minimal diagonal coefficient is still $X^{\delta m}$, like in Lemma 30. However, the balance between these two coefficients has changed, because the bound X is much smaller than in Coppersmith's algorithm. Before, X was essentially $N^{\frac{n-\delta+1}{\delta n}}$ whose order of magnitude is the same as $N^{1/\delta}$, but now, it is close to $N^{\alpha^2/\delta}$, so that

$$X^{\delta m} \simeq N^{\delta m \alpha^2/\delta} = N^{m\alpha^2} \quad .$$

In other words, the ratio between the maximal and minimal diagonal coefficient is about

$$\frac{N^m X^{\delta-1}}{N^{m\alpha^2}} \simeq N^{(1-\alpha^2)m + (\alpha^2(\delta-1)/\delta)} \simeq N^{(1-\alpha^2)m} \quad .$$

Therefore, by performing the rounding improvement as before, *i.e.* by dividing all coefficients of B by $X^{\delta m}/c$ where $c > 1$ is a small parameter, the bit-size of the truncated elements will be bounded by a value close to $N^{(1-\alpha^2)m}$, in comparison to the univariate modular case, where the truncated elements were bounded by $N^{\mathcal{O}(1)}$.

We are thus trading an LLL-reduction of a matrix with bit-size $\approx m \log N$, with one with bit-size $\approx (1-\alpha^2) m \log N$, which can only provide a small constant speedup at best, namely $1/(1-\alpha^2)^2$ for L^2 or close to $1/(1-\alpha^2)$ for $\tilde{L}^1$. Thus, in the gcd generalization, the input basis is much less reduced than in Coppersmith's algorithm.

4.4.2 Multivariate Equations

As depicted in Chapter 3, Coppersmith [Cop96b, Cop97] showed that his algorithm for finding small roots of univariate polynomial congruences can heuristically be extended to multivariate polynomial congruences: the most famous example is the Boneh-Durfee attack [BD00] on RSA with short secret exponent.

Not all these multivariate variants use triangular matrices, though they sometimes can be tweaked: some rely on lattices which are not full-rank, including the Boneh-Durfee attack [BD00]. However, when the matrix is triangular, there is a similar problem than for the gcd generalization: the diagonal coefficients are much more unbalanced than in

the univariate congruence case, which means that the speedup of the rounding method is at most a small constant. And in the Boneh-Durfee attack, the coefficients which play the role of the diagonal coefficients are also unbalanced.

For instance, assume that one would like to find all small roots of $f(x,y) \equiv 0 \pmod N$ with $|x| \leq X$ and $|y| \leq Y$, where $f(x,y)$ has total degree δ and has at least one monic monomial $x^\alpha y^{\delta-\alpha}$ of maximal total degree. Then, for a given parameter m, the lower-triangular matrix has dimension $n = (\delta m + 1)(\delta m + 2)/2$ and diagonal coefficients

$$N^{m-v} X^{u_1+v\delta} Y^{u_2+v(\delta-\alpha)} \qquad \text{where} \quad u_1 + u_2 + \delta v \leq m\delta \quad \text{and} \quad u_1, u_2, v \geq 0 \ ,$$

with $u_1 < \alpha$ or $u_2 < \delta - \alpha$. For typical choices of X and Y such that $XY < N^{1/\delta-\varepsilon}$, the ratio between the largest and smallest diagonal coefficient is no longer $N^{O(1)}$.

Yet, we deduce that we only get a small constant speed-up for other small-root algorithms. We leave it as an open problem to obtain polynomial (non-constant) speedups for these other small-root algorithms: this might be useful to make practical attacks on certain fully-homomorphic encryption schemes (see [CH11a]).

Chapter 5

Factoring $N = p^r q^s$ for Large r

Contents

The results presented in this chapter are from a joint work with Jean-Sébastien Coron, Jean-Charles Faugère and Guénaël Renault.

Boneh, Durfee and Howgrave-Graham (BDH) showed at Crypto 99 [BDHG99] that moduli of the form $N = p^r q$ can be factored in polynomial time for large r, when $r \simeq \log p$. As recalled in Chapter 3.3, their algorithm is based on Coppersmith's technique

for finding small roots of polynomial equations [Cop97], which uses lattice reduction. In the BDH paper the generalization to moduli of the form $N = p^r q^s$ where r and s are approximately the same size, is explicitly left as an open problem. In this chapter we solve this open problem and we identify a new class of integers that can be efficiently factored. Namely, we describe a new algorithm to factor $N = p^r q^s$ in deterministic polynomial time when at least one of both exponents r or s is greater than $(\log p)^3$.

Our technique consists in decomposing the exponents r and s so as to write $N = P^u Q$ for some large enough u, where $P = p^\alpha q^\beta$ and $Q = p^a q^b$. This decomposition is obtained by LLL-reducing a well-designed matrix which only depends on r and s. Depending on the considered decomposition one subsequently applies BDH's method with $N = P^u Q$, or Coppersmith's technique for univariate congruencies. This allows to recover P and Q, and eventually the prime factors p and q.

As a next step, we generalize our technique for moduli of the form $N = \prod_{i=1}^k p_i^{r_i}$ with more than two prime factors. Namely, we show that a sufficient condition to extract a non-trivial factor of such moduli in polynomial time in $\log N$ is that the largest of the k exponents r_i is in $\Omega(\log^{\ell_k}(\max p_i))$, where $\ell_2 = 3$ and $\ell_k = 4(k-1)(1+\sum_{i=1}^{k-3}\prod_{j=i}^{k-3} j)+1$ for $k \geqslant 3$. For example, we have $\ell_3 = 9$, $\ell_4 = 25$, and more generally $\ell_k = \mathcal{O}(k!)$.

Roadmap. In Section 5.1, we give a slightly simpler condition than the one given in BDH's Theorem (Theorem 27) which will rather be used in the current chapter. In Section 5.2, we start by showing why natural approaches fail to factorize $N = p^r q^s$ in polynomial time for large r and s. Then we derive a condition on r and describe our method for a polynomial time factorization of $N = p^r q^s$. We also show that this condition can be improved in the case where s is small, or when s is close to r. In Section 5.3, we generalize our method for modulus of the form $N = \prod_{i=1}^k p_i^{r_i}$. Finally, in Section 5.5, we present our experimental results.

Tools. In this Chapter, we make use of Coppersmith's small-root method for the univariate modular case, whose reminder is provided in Chapter 3.1. We also make use of the Boneh-Durfee-Howgrave-Graham technique described in Chapter 3.3.

5.1 BDH's Theorem Slightly Revisited

In Chapter 3.3 we recalled in Theorem 27 the original BDH's Theorem from [BDHG99]. Roughly, it states that given a modulus of the form $N = p^r q$ where $q < p^c$ for some rational $c > 0$, one can factorize N in time $\exp\left(\frac{c+1}{r+c} \cdot \log p\right) \cdot \mathcal{O}(\gamma)$, where γ is the complexity of the LLL-reduction. A direct consequence is that if we have $(r+c)/(c+1) = \Omega(\log p)$, then the running time becomes $\exp(\mathcal{O}(1)) \cdot \mathcal{O}(\gamma)$, which is polynomial in $\log N$.

Actually by simple inspection of the proof of Theorem 27 in [BDHG99] one can obtain the slightly simpler condition $r = \Omega(\log q)$ instead of the previous $(r + c)/(c + 1) = \mathcal{O}(\log p)$ for a polynomial time factorization. Namely, this is highlighted in the following theorem that will rather be used in the current chapter.

Theorem 42 (BDH)**.** *Let p and q be two integers with $p \geq 2$ and $q \geq 2$, and let $N = p^r q$. The factors p and q can be recovered in polynomial time in $\log N$ if*

$$r = \Omega(\log q) \quad .$$

Proof. We start from Lemma 28 which is taken from [BDHG99] whose proof is recalled in Chapter 3.3. We emphasize that in Lemma 28 the integers p and q can be any integers greater than 2, and not necessarily prime numbers. Indeed, the proof of Lemma 28 does not depend on p and q being primes.

Then, instead of taking a dimension $n = 2r(r + c)$ as in Chapter 3.3 where c is such that $q < p^c$, we now take $n = 2\lceil r \cdot \log p \rceil$, which gives the sufficient condition:

$$|P - p| < p^{1 - \frac{c}{r+c} - \frac{1}{\log p}}$$

and therefore, in order to factor $N = p^r q$ it suffices to perform exhaustive search on a fraction $c/(r + c) < c/r$ of the bits of p, which gives a running time:

$$\exp\left(\frac{c}{r} \cdot \log p\right) \cdot \mathsf{poly}(\log N)$$

Moreover we can take c such that $(p^c)/2 < q < p^c$, which gives $p^c < 2q$. Thus one gets $c \log p < \log q + \log 2$. Therefore the running time is:

$$\exp\left(\frac{\log q}{r}\right) \cdot \mathsf{poly}(\log N)$$

and therefore a sufficient condition for polynomial-time factorization of $N = p^r q$ is

$$r = \Omega(\log q) \quad .$$

This concludes the proof of Theorem 42.

□

Note that if p and q are integers of the same bit-size, *i.e.* $c = 1$, then one retrieves the condition $r = \Omega(\log p)$ given in [BDHG99]. Furthermore, we emphasize again that in Theorem 42 (as well as in Theorem 27), the values p and q can be any integers, and not necessarily prime numbers. Thus, one will need this latter property in the sequel.

5.2 Factoring $N = p^r q^s$ for Large r

5.2.1 Two Natural Approaches that Fail

In BDH's paper, the generalization to moduli of the form $N = p^r q^s$ where r and s are approximately the same size, is explicitly left as an open problem. In the following, we start by providing two natural approaches that fail to factorize moduli $N = p^r q^s$. The first one is a straightforward application of BDH's method, and the second one is an application of Coppersmith's method for finding small roots of bivariate polynomials over $\mathbb{Z}$. We indeed show that in fact, both approaches do not lead to a polynomial-time factorization of $N = p^r q^s$.

A First Approach: Straightforward Use of BDH's method

In order to factorize moduli of the form $N = p^r q^s$, one could try to straightforwardly use BDH's technique on $N = p^r Q$ where $Q = q^s$. However, according to Theorem 42 the condition for a polynomial-time factorization becomes $r = \Omega(\log Q) = \Omega(\log q^s) = \Omega(s \log q)$, which is

$$\frac{r}{s} = \Omega(\log q) \quad .$$

Therefore, there must be a sufficient gap between the exponents r and s. Namely, the ratio r/s should be larger than $\log q$, and thus, r should be much larger than s. But in the case where r and s have approximately the same size, this approach does not allow a polynomial-time factorization of N.

A Second Approach: Use of Coppersmith's Second Theorem

Alternatively, a natural approach to factor $N = p^r q^s$ would be to write $p = P + x_0$ and $q = Q + y_0$ where $|x_0| \leqslant X$ and $|y_0| \leqslant Y$ for some y, and in a first step we assume that P and Q are given. Therefore (x_0, y_0) is a small root over $\mathbb{Z}$ of the bivariate polynomial:

$$f(x, y) = (P + x)^r (Q + y)^s \quad .$$

Without loss of generality, one can assume that $r > s$. Thus, the degree of $f(x, y)$ is at most r separately in x and y. Therefore, according to Theorem 26, one can retrieve the root (x_0, y_0) if the following condition is satisfied:

$$XY < W^{2/(3r)} \quad ,$$

where $W = P^r Q^s \simeq N$. Hence, one has the condition

$$XY < W^{2/(3r)} \simeq N^{2/(3r)} = p^{2/3} q^{2s/(3r)} \quad .$$

If r is close to s, the condition can be approximated by $XY < (pq)^{2/3}$. Therefore one should take the bounds $X \simeq p^{2/3}$ and $Y \simeq q^{2/3}$. This implies that to recover p and q in polynomial time we must know at least $1/3$ of the high-order bits of p and $1/3$ of the high-order bits of q. If r is much larger than s, the condition is close to $XY < p^{2/3}$, and one should take the bounds $X \simeq p^{1/3}$ and $Y \simeq q^{1/3}$, which means that to recover p and q in polynomial time we must know at least $2/3$ of the high-order bits of p and $2/3$ of the high-order bits of q. Since in both cases (r close or far to s) this is a constant fraction of the bits of p and q (that cannot be lowered by making r or s increasing), Coppersmith's method for the bivariate integer case does not enable to factor $N = p^r q^s$ in polynomial-time for any r and s.

5.2.2 The Main Theorem

In the following, we describe our method to factorize moduli of the form $N = p^r q^s$. As in BDH's method, we consider primes p and q which can have different sizes. Without loss of generality, we assume for the remaining of the chapter that $r > s$, as we can

swap p and q if $r < s$. We also assume that $\gcd(r, s) = 1$, otherwise we should consider $N' = N^{1/\delta}$ where $\delta = gcd(r, s)$. Finally, the exponents r and s can be supposed to be known, otherwise they can be recovered by exhaustive search in time $\mathcal{O}(\log^2 N)$.

The condition on r to factorize moduli of the form $N = p^r q^s$ in polynomial time in $\log N$ is highlighted in our main theorem:

Theorem 43. *Let $N = p^r q^s$ be an integer of unknown factorization with $r > s$ and $\gcd(r, s) = 1$. One can recover the prime factors p and q in polynomial time in $\log N$ under the condition*

$$r = \Omega(\log^3 \max(p, q)) \quad .$$

5.2.3 An Outline of the Method

In the following algorithm, we give an outline of our method that enables to show Theorem 43.

Algorithm 8 Method for factoring $N = p^r q^s$ in polynomial time in $\log N$

Input: Two positive integers r and s such that $r > s$ and $r = \Omega(\log^3 \max(p, q))$. A modulus $N = p^r q^s$.

Output: Primes p and q.

1: Find integers $\alpha > 0$ and $\beta \geqslant 0$ such that $r \cdot \beta - s \cdot \alpha = \gamma$ with $|\gamma| < 2 \cdot r^{2/3}$ and $\{\alpha, \beta\} < 2 \cdot r^{1/3}$ using LLL-reduction.
2: **if** $\lfloor r/\alpha \rfloor \leqslant s/\beta$ **then**
3: Compute $u = \lfloor r/\alpha \rfloor$.
4: Compute positive integer values a and b such that $r = \alpha u + a$ and $s = \beta u + b$.
5: Apply BDH's factorization method on $N = P^u Q$ where $P = p^\alpha q^\beta$ and $Q = p^a q^b$.
6: **else**
7: Compute $u = \lceil r/\alpha \rceil$.
8: Compute negative integer values a and b such that $r = \alpha u + a$ and $s = \beta u + b$.
9: Apply Coppersmith's method for finding small roots of univariate modular polynomials with $P^u = Q \cdot N$ where $P = p^\alpha q^\beta$ and $Q = p^{-a} q^{-b}$.
10: **end if**
11: From (P, Q), recover p and q.

5.2.4 A Useful Lemma: Decomposition of r and s

The proof is based on the following lemma.

Lemma 44. *Let r and s be two integers such that $r > s > 0$. One can compute in polynomial time integers u, α, β, a, b such that*

$$\begin{cases} r = u \cdot \alpha + a \\ s = u \cdot \beta + b \end{cases} \tag{5.1}$$

with

$$\begin{cases} 0 < \alpha \leqslant 2r^{1/3} & \text{and} \quad 0 \leqslant \beta \leqslant \alpha \\ |a| < \alpha & \text{and} \quad |b| \leqslant 6r^{2/3}/\alpha \\ u > r/\alpha - 1 \end{cases} , \tag{5.2}$$

where the integers a and b are either both $\geqslant 0$ (Case 1), or both $\leqslant 0$ (Case 2).

Proof. We first generate two small integers $\alpha > 0$ and β such that:

$$r \cdot \beta - s \cdot \alpha = \gamma \ , \tag{5.3}$$

for some small integer γ. For this we apply LLL on the following matrix M of row vectors:

$$M = \begin{pmatrix} \lfloor r^{1/3} \rfloor & -s \\ 0 & r \end{pmatrix} .$$

We obtain a short non-zero vector $\mathbf{v} = (\lfloor r^{1/3} \rfloor \cdot \alpha, \gamma)$, where $\gamma = -s \cdot \alpha + r \cdot \beta$ for some $\beta \in \mathbb{Z}$; hence we obtain integers α, β and γ satisfying equation (5.3).

According to Theorem 13, the first vector $\mathbf{v}$ of the LLL-reduced matrix M^R satisfies $\|\mathbf{v}\| < 2^{(n-1)/4}(\det M)^{1/n}$ where $n = 2$ is the dimension of the lattice. Therefore, we must have

$$\|\mathbf{v}\| \leqslant 2^{1/4} \cdot (\det M)^{1/2} \leqslant 2^{1/4} \cdot (\lfloor r^{1/3} \rfloor \cdot r)^{1/2} \leqslant 2^{1/4} \cdot r^{2/3} \ .$$

This gives the following two bounds :

$$|\alpha| \leqslant 2r^{1/3} \quad \text{and} \quad |\gamma| \leqslant 2r^{2/3} \ .$$

Note that by applying the Gauss-Lagrange algorithm instead of LLL one can obtain a slightly better bound for $\|\mathbf{v}\|$, corresponding to Minkowski bound (see Theorem 7).

Furthermore, one can assume that $\alpha \geqslant 0$ since if the obtained α is negative, then one can take vector $-\mathbf{v}$ instead of $\mathbf{v}$. Moreover we must have $\alpha \neq 0$ since otherwise we would have $\mathbf{v} = (0, \beta r)$ for some integer $\beta \neq 0$, which would give $\|\mathbf{v}\| \geqslant r$, which would contradict the previous bound. Therefore we must have $0 < \alpha \leqslant 2r^{1/3}$.

From equation (5.3) we have $\beta = (\gamma + \alpha \cdot s)/r$ and moreover using $-1 < \gamma/r < 1$ and $0 < s < r$ we obtain:

$$-1 < \frac{\gamma}{r} < \frac{\gamma + \alpha \cdot s}{r} < \frac{\gamma}{r} + \alpha < 1 + \alpha$$

Since α and β are integers this implies $0 \leq \beta \leq \alpha$.

We now show how to generate the integers u, a and b. In fact, we must ensure that a and b are either both $\geqslant 0$ (Case 1) or both $\leqslant 0$ (Case 2). This is why we distinguish two cases: a first one when $\lfloor r/\alpha \rfloor \leqslant s/\beta$ or $\beta = 0$, and a second one when $\lfloor r/\alpha \rfloor > s/\beta$.

Case 1: $\beta = 0$ or ($\beta \neq 0$ and $\lfloor r/\alpha \rfloor \leq s/\beta$). In that case we let:

$$u := \left\lfloor \frac{r}{\alpha} \right\rfloor$$

and we let

$$a := r - u \cdot \alpha \quad \text{and} \quad b := s - u \cdot \beta \ .$$

This gives (5.1) as required. Since a is the remainder of the division of r by α we must have $0 \leq a < \alpha$. If $\beta = 0$ we then have $b = s > 0$. If $\beta \neq 0$ we have using $\lfloor r/\alpha \rfloor \leq s/\beta$:

$$b = s - u \cdot \beta = s - \left\lfloor \frac{r}{\alpha} \right\rfloor \cdot \beta \geq s - \frac{s}{\beta} \cdot \beta = 0 \ ,$$

so in both cases $b \geq 0$. Therefore in Case 1 we have that the integers a and b are both ≥ 0. Moreover the relation $a \cdot \beta - b \cdot \alpha = \gamma$ is acknowledged. Indeed, by combining (5.1) and (5.3) we obtain

$$a \cdot \beta - b \cdot \alpha = a \cdot \beta - (s - u \cdot \beta)\, \alpha = (a - u \cdot \alpha)\, \beta - s \cdot \alpha = r \cdot \beta - s \cdot \alpha = \gamma \ .$$

This gives using $0 \leqslant \beta \leqslant \alpha$ and $0 \leqslant a < \alpha$:

$$0 \leq b = \frac{a \cdot \beta - \gamma}{\alpha} \leqslant \frac{a \cdot \beta + |\gamma|}{\alpha} < \frac{\alpha^2 + |\gamma|}{\alpha} = \alpha + \frac{|\gamma|}{\alpha} \leqslant \alpha + \frac{2r^{2/3}}{\alpha} \ .$$

Since $0 < \alpha \leq 2r^{1/3}$ we have $4r^{2/3}/\alpha \geq 2r^{1/3} \geq \alpha$, therefore we obtain as required:

$$0 \leq b < \frac{6r^{2/3}}{\alpha} \ .$$

Case 2: $\beta \neq 0$ and $\lfloor r/\alpha \rfloor > s/\beta$. In that case we let:

$$u := \left\lceil \frac{r}{\alpha} \right\rceil \ .$$

As previously we let $a := r - u \cdot \alpha$ and $b := s - u \cdot \beta$, which gives again (5.1). Moreover we have $-\alpha < a \leq 0$. As previously using $\lceil r/\alpha \rceil \geq \lfloor r/\alpha \rfloor > s/\beta$ we obtain:

$$b = s - u \cdot \beta = s - \left\lceil \frac{r}{\alpha} \right\rceil \cdot \beta < s - \frac{s}{\beta} \cdot \beta = 0 \ .$$

Therefore in Case 2 we have that the integers a and b are both ≤ 0. As previously using $0 \leq \beta \leq \alpha$, $-\alpha < a \leq 0$ and $\alpha \leq 4r^{2/3}/\alpha$ we obtain as required:

$$|b| \leq \left| \frac{a \cdot \beta - \gamma}{\alpha} \right| < \alpha + \frac{2r^{2/3}}{\alpha} \leq \frac{6r^{2/3}}{\alpha} \ .$$

This terminates the proof of Lemma 44.

□

5.2.5 Proof of the Main Theorem

We now proceed with the proof of Theorem 43. We are given as input $N = p^r q^s$. As said before, we can assume that the exponents r and s are known, otherwise they can be recovered by exhaustive search in time $\mathcal{O}(\log^2 N)$. We apply Lemma 44 with r, s and obtain u, α, β, a and b such that:

$$\begin{cases} r = u \cdot \alpha + a \\ s = u \cdot \beta + b \end{cases}$$

The rest of the proof differs according to whether Case 1 or Case 2 is considered.

Case 1 when $\lfloor r/\alpha \rfloor \leqslant s/\beta$ or $\beta = 0$: An application of BDH's method

We first consider Case 1 of Lemma 44 with $a \geq 0$ and $b \geq 0$. In that case the modulus $N = p^r q^s$ can be rewritten as follows:

$$N = p^r q^s = p^{u\cdot\alpha+a} q^{u\cdot\beta+b} = (p^\alpha q^\beta)^u p^a q^b = P^u Q \quad ,$$

where $P := p^\alpha q^\beta$ and $Q := p^a q^b$. One can then apply Theorem 42 on $N = P^u Q$ to recover P and Q in polynomial time in $\log N$ under the condition

$$u = \Omega(\log Q) \quad .$$

Since $u > r/\alpha - 1$, we get the sufficient condition

$$r = \Omega(\alpha \cdot \log Q) \quad .$$

We have from the bounds of Lemma 44:

$$\begin{aligned} \alpha \cdot \log Q &= \alpha \cdot (a \log p + b \log q) \leq \alpha \cdot \left(\alpha \cdot \log p + \frac{6r^{2/3}}{\alpha} \cdot \log q \right) \\ &\leq \alpha^2 \cdot \log p + 6r^{2/3} \cdot \log q \leq 10 \cdot r^{2/3} \cdot \log \max(p, q) \end{aligned}$$

which gives the sufficient condition

$$r = \Omega(r^{2/3} \cdot \log \max(p, q)) \quad .$$

Therefore one can recover P and Q in polynomial time under the condition:

$$r = \Omega(\log^3 \max(p, q)) \quad .$$

Last Step. Eventually the prime factors p and q can easily be recovered from $P = p^\alpha q^\beta$ and $Q = p^a q^b = N/P^u$. Indeed, the matrix $\begin{pmatrix} a & b \\ \alpha & \beta \end{pmatrix}$ whose determinant is $a \cdot \beta - b \cdot \alpha = \gamma$,

is invertible with inverse

$$\begin{pmatrix} \beta/\gamma & -b/\gamma \\ -\alpha/\gamma & a/\gamma \end{pmatrix} .$$

Namely we must have $\gamma \neq 0$, since otherwise we would have $\beta \cdot r = \alpha \cdot s$. But since we have $\gcd(r, s) = 1$, the integer α would be non-zero multiple of r, which would contradict the bound from Lemma 44. Therefore one can retrieve p and q by computing:

$$\begin{cases} Q^{\frac{\beta}{\gamma}} \cdot P^{\frac{-b}{\gamma}} = (p^a q^b)^{\frac{\beta}{\gamma}} \cdot (p^\alpha q^\beta)^{\frac{-b}{\gamma}} = p^{\frac{a\beta - b\alpha}{\gamma}} \cdot q^{\frac{b\beta - b\beta}{\gamma}} = p^1 \cdot q^0 = p \\ Q^{\frac{-\alpha}{\gamma}} \cdot P^{\frac{a}{\gamma}} = (p^a q^b)^{\frac{-\alpha}{\gamma}} \cdot (p^\alpha q^\beta)^{\frac{a}{\gamma}} = p^{\frac{a\alpha - a\alpha}{\gamma}} \cdot q^{\frac{a\beta - b\alpha}{\gamma}} = p^0 \cdot q^1 = q \end{cases} .$$

Complexity Analysis. The complexity of the L^2 algorithm is $\mathcal{O}(n^{5+\varepsilon} \log b + n^{4+\varepsilon} \log^2 b)$ when the entries are bounded by b and n is the dimension of the lattice used. Here the entries are bounded by $N^m < p^{(r+c)m}$, where $m > 0$ is an integer parameter (see Chapter 3.3). Since $n = \mathcal{O}((r + c)m)$, we have $B = \mathcal{O}(p^n)$. Therefore $\log b = \mathcal{O}(n \log p)$ and the time complexity is $\mathcal{O}(n^{6+\varepsilon} \log^2 p)$. Note that according to BDH's method, the asymptotical dimension n of the lattice is $n = \mathcal{O}(u \log P) = \mathcal{O}((r/\alpha)(\alpha \log p + \beta \log q)) = \mathcal{O}(r(\log p + \log q)) = \mathcal{O}(r \cdot \log \max(p, q))$. Therefore, for $r \simeq \log^3 \max(p, q)$, the dimension is $n = \mathcal{O}(\log^4 \max(p, q))$ and we have $\log N \simeq \log^4 \max(p, q)$. As a consequence, the complexity is $\mathcal{O}(\log^{26+\epsilon} \max(p, q))$, which finally gives the complexity $\mathcal{O}(\log^{6.5+\varepsilon} N)$.

Case 2 when $\lfloor r/\alpha \rfloor > s/\beta$: An application of Coppersmith's method

We now consider Case 2 from Lemma 44, that is $a \leqslant 0$ and $b \leqslant 0$. In that case we can write:

$$N = p^r q^s = p^{u \cdot \alpha + a} q^{u \cdot \beta + b} = (p^\alpha q^\beta)^u p^a q^b = P^u / Q$$

for $P := p^\alpha q^\beta$ and $Q := p^{-a} q^{-b}$. Note that Q is an integer because $a \leq 0$ and $b \leq 0$. We obtain $P^u = Q \cdot N$ which implies:

$$P^u \equiv 0 \pmod{N} .$$

Therefore P is a small root of a univariate polynomial equation of degree u modulo N. Hence we can apply Coppersmith's Theorem for the univariate modular case. The condition from Theorem 22 is

$$P \leq N^{1/u} = P/Q^{1/u} .$$

Although the condition is not directly satisfied, it can be met by doing exhaustive search on the high-order $(\log Q)/u$ bits of P, which is still polynomial time under the condition

$$u = \Omega(\log Q) .$$

This is the same condition as in Case 1 for BDH.

More precisely, we write $P = X \cdot t + x_0$ where $X = \lfloor N^{1/u} \rfloor$ and $|x_0| \leq X$. We obtain the polynomial equation:

$$(X \cdot t + x_0)^u \equiv 0 \mod N \quad .$$

For a fixed t this is a univariate modular polynomial equation of degree u and small unknown x_0. We have $X < N^{1/u}$. Therefore we can apply Theorem 22 and recover x_0 in polynomial time in $\log N$. Since the integer t is unknown, we do exhaustive search on t, where:

$$0 \leq t \leq P/X \leq 2P/N^{1/u} = 2Q^{1/u} \quad .$$

Therefore the algorithm is still polynomial time under the same condition as in Case 1, namely $u = \Omega(\log Q)$. Since in Lemma 44 the bounds on u, a and b are the same in both Case 1 and Case 2, we obtain that in Case 2 recovering P and Q is polynomial-time under the same condition

$$r = \Omega(\log^3 \max(p, q)) \quad .$$

Last Step. Similarly as before, one can recover p and q from P and $Q = P^u/N$. The same reasoning as before holds. Thus, one has $p = Q^{\frac{\beta}{\gamma}} \cdot P^{\frac{-b}{\gamma}}$ and $q = Q^{\frac{-\alpha}{\gamma}} \cdot P^{\frac{a}{\gamma}}$.

Complexity Analysis. The complexity of the L^2 algorithm is $\mathcal{O}(n^{5+\varepsilon} \log b + n^{4+\varepsilon} \log^2 b)$ when the entries are bounded by b and n is the dimension of the lattice used. Here the entries are bounded by $N^{m+1} = (p^r q^s)^{m+1}$ (see Chapter 3.1). Since $n = rm+1$, we have $b = \mathcal{O}((\max(p, q))^n)$. Therefore $\log b = \mathcal{O}(n \log \max(p, q))$ and the time complexity is $\mathcal{O}(n^{6+\varepsilon} \log^2 \max(p, q))$. Note that similarly as before, in Coppersmith's method the asymptotical dimension n of the lattice is $n = \mathcal{O}(\log N) = \mathcal{O}(r \log p + s \log q) = \mathcal{O}(r \cdot \log \max(p, q))$. Therefore, for $r \simeq \log^3 \max(p, q)$, the dimension is again $n = \mathcal{O}(\log^4 \max(p, q))$ and we have $\log N \simeq \log^4 \max(p, q)$. As a consequence, the complexity is $\mathcal{O}(\log^{26+\epsilon} \max(p, q))$, which finally gives the complexity $\mathcal{O}(\log^{6.5+\varepsilon} N)$.

As a conclusion, we have shown that in both cases, $\lfloor r/\alpha \rfloor \leqslant s/\beta$ and $\lfloor r/\alpha \rfloor > s/\beta$, the associated approach (BDH or Coppersmith) can factorize N in polynomial time in $\log N$ if $r = \Omega(\log^3 \max(p, q))$. This terminates the proof of Theorem 43.

5.2.6 Refinement of the Condition on r for Small s or for s Close to r.

In the following, we show that the condition derived in Theorem 43 can be improved for the two specific cases where s is small and when s is close to r.

Case where s is small

In the case where s is small, namely if $s < \log^2 \max(p, q)$, then a more refined bound can be derived:

Lemma 45. *Let $N = p^r q^s$ be an integer of unknown factorization with $r > s$ and $\gcd(r, s) = 1$. Let s be such that $s < \log^2 \max(p, q)$. Then one can factorize N in*

polynomial time in $\log N$ *if:*

$$\frac{r}{s} \;=\; \Omega(\log q) \;.$$

Proof. By using BDH's method on $N = (p)^r(q^s) = P^uQ$ with $u = r$, $P = p$ and $Q = q^s$, the condition $u = \Omega(\log Q)$ becomes $r = \Omega(\log q^s) = \Omega(s \cdot \log q)$. Therefore, if $s < \log^2 \max(p, q)$, then we have $s \cdot \log q < \log^2 \max(p, q) \cdot \log q < \log^3 \max(p, q)$. We deduce that a more refined condition than the one given in Theorem 43 would be $r/s = \Omega(\log q)$.

□

Case where s is close to r

Another interesting case appears when r and s are close, namely if $r-s < \log^2 \max(p, q)$, then the following bound can be derived:

Lemma 46. *Let $N = p^r q^s$ be an integer of unknown factorization with $r > s$ and $\gcd(r, s) = 1$. Let s be such that $r - s < \log^2 \max(p, q)$. Then one can factorize N in polynomial time in $\log N$ if:*

$$\frac{r}{r-s} \;=\; \Omega(\log q) \;.$$

Proof. By using Coppersmith's method on $N = (pq)^r(q^{s-r}) = P^u/Q$ with $u = r$, $P = pq$ and $Q = q^{r-s}$, the condition $u = \Omega(\log Q)$ becomes $r = \Omega(\log q^{r-s}) = \Omega((r - s) \cdot \log q)$. Therefore, if $r - s < \log^2 \max(p, q)$, then we have $(r - s) \cdot \log q < \log^2 \max(p, q) \cdot \log q < \log^3 \max(p, q)$. We deduce that a more refined condition than the one given in Theorem 43 would be $r/(r - s) = \Omega(\log q)$.

□

A summary of the conditions in Figure 5.1:

We have summarized in Figure 5.1 results from Theorem 43, Lemma 45 and Lemma 46. Namely, if s is small (i.e. $s < \log^2 \max(p, q)$) or close to r (i.e. $r - s < \log^2 \max(p, q)$) then the condition ranges from $r = \Omega(\log q)$ to $r = \Omega(\log^3 \max(p, q))$ depending on s. In all other cases, the condition is $r = \Omega(\log^3 \max(p, q))$.

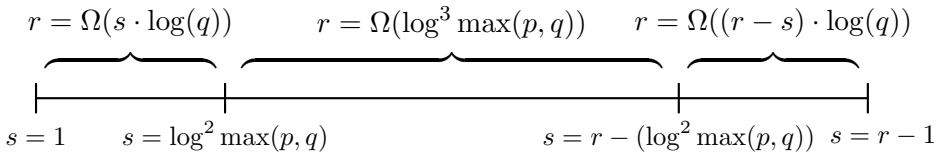
Therefore, we enlighten that for the case where $s = 1$, one retrieves the bound $r = \Omega(\log q)$ given in [BDHG99]. Besides, in the case where $s = r - 1$, one also gets the condition $r = \Omega(\log q)$.

5.3 Generalization for $N = \prod p_i^{r_i}$ for Large r_i's

We generalize the previous techniques to moduli of the form

$$N = \prod_{i=1}^{k} p_i^{r_i} \quad ,$$

Figure 5.1: Conditions on r for a polynomial-time factorization of $N = p^r q^s$ as a function of s, when p and q are two primes of the same bit-size.



with more than two prime factors. As before, we consider primes p_i for $1 \leqslant i \leqslant k$, which can have different sizes. In order to ease the notations, in this section we let $p := \max\{p_i, 1 \leqslant i \leqslant k\}$. Note that with 3 prime factors or more we cannot hope to obtain a complete factorization of N. Namely starting from an RSA modulus $N_1 = pq$ one could artificially embed N_1 into a larger modulus $N = (pq)^r q'$ for some known prime q', and recover the factorization of N_1 by factoring N, but of course, this cannot be. For the same reason we cannot hope to extract even a single prime factor of N; namely given two RSA moduli $N_1 = p_1 q_1$ and $N_2 = p_2 q_2$ and using $N = (N_1)^r N_2$, extracting a prime factor of N would factor either N_1 or N_2. Instead we show that we can always extract a non-trivial factor of N, if one of the exponents r_i is large enough. Namely, we show that we can always recover a non-trivial factor of N in polynomial time if the largest r_i is at least $\Omega(\log^{\theta_k} \max p_i)$, for some sequence θ_k with $\theta_3 = 9$, $\theta_4 = 25$, $\theta_5 = 81$ and $\theta_k = \mathcal{O}(k!)$ for large k. More precisely, we prove the following Theorem, which is a generalization of Theorem 43.

Theorem 47. *Let $k \geq 2$ be fixed and let $N = \prod_{i=1}^{k} p_i^{r_i}$ where $r_1 = \max(r_i)$. Let $p := \max\{p_i, 1 \leqslant i \leqslant k\}$. One can recover a non-trivial factor of N in time polynomial in $\log N$ if*

$$r_1 = \Omega(\log^{\theta_k} p) \quad ,$$

where $\theta_2 = 3$ and:

$$\theta_k = 4(k-1)\left(1 + \sum_{i=1}^{k-3} \prod_{j=i}^{k-3} j\right) + 1, \quad \text{for} \quad k \geqslant 3,$$

with $\theta_k = \mathcal{O}(k!)$ for large values of k.

The proof of Theorem 47 is based on three lemmas which are highlighted in the sequel.

5.3.1 A Condition on r_1 Depending on the Ratio r_1/r_{k-1}

Without loss of generality, we assume in the proof that $r_1 \geqslant r_2 \geqslant \cdots \geqslant r_k$, otherwise, one should as before interchange the primes p_i so that this condition is satisfied.

In the first Lemma 48, we derive a first condition on r_1 for a polynomial-time factorization of N, which depends on the ratio r_1/r_{k-1}. We will give afterwards an upper-bound on this ratio in order to derive a general condition which does not depend anymore on this ratio.

Lemma 48. *Let $N = \prod_{i=1}^{k} p_i^{r_i}$ for a fixed $k \geqslant 2$ and $r_1 \geqslant r_2 \geqslant \cdots \geqslant r_k$. Let $p = \max\{p_i, 1 \leqslant i \leqslant k\}$ and ρ be such that $r_1/r_{k-1} \leqslant \log^{\rho} p$. Then, one can recover a non-trivial factor of N in polynomial time in $\log N$ if*

$$r_1 = \Omega\left(\log^{2(k-1)(\rho+1)+1} p\right) \quad .$$

Proof. Our technique is as follows. We start by finding k small integers $\alpha_1, \ldots, \alpha_k$ and $k-1$ small integers $\gamma_2, \ldots, \gamma_k$ such that:

$$r_1 \cdot \alpha_i - r_i \cdot \alpha_1 = \gamma_i \tag{5.4}$$

for all $2 \leq i \leq k$. In order to get such equations, we apply Theorem 20 with $n = k-1$, $\varepsilon = \log^{-(\rho+1)} p$ and $e_{i-1} = r_i/r_1$ for $2 \leqslant i \leqslant k$. This allows us to obtain integers $\alpha_1, \alpha_2, \ldots, \alpha_k$ satisfying:

$$|\alpha_i - \alpha_1 \cdot \frac{r_i}{r_1}| \leqslant \varepsilon \quad .$$

Therefore we obtain the sought relations (5.4) with:

$$|\gamma_i| \leqslant r_1 \cdot \varepsilon \qquad \text{and} \qquad 1 \leqslant \alpha_1 \leqslant 2^{\frac{k(k-1)}{4}} \cdot \varepsilon^{-(k-1)} \quad .$$

Note that for all $2 \leqslant i \leqslant k$ we have $0 \leqslant \alpha_i \leqslant \alpha_1$. Indeed, from (5.4), we have $\alpha_i = (\gamma_i + r_i \cdot \alpha_1)/r_1$. Therefore, by using $|\gamma_i| \leq r_1 \cdot \varepsilon$ with $0 < \varepsilon < 1$, one deduces that:

$$-1 < -\varepsilon < -\varepsilon + \frac{r_i \cdot \alpha_1}{r_1} \leqslant \alpha_i = \frac{\gamma_i + r_i \cdot \alpha_1}{r_1} \leqslant \varepsilon + \frac{r_i \cdot \alpha_1}{r_1} < 1 + \alpha_1 \quad ,$$

and since α_1 and α_i are integers, one deduces that $0 \leqslant \alpha_i \leqslant \alpha_1$.

Then as previously, we compute the integer value $u > 0$ as follows:

$$u = \min\left\{ \left\lfloor \frac{r_i}{\alpha_i} \right\rfloor \text{ with } \alpha_i \neq 0, \text{ for } 1 \leqslant i \leqslant k \right\} \quad .$$

We know that such u exists because $\alpha_1 \neq 0$. We take the smallest index j such that $u = \left\lfloor \frac{r_j}{\alpha_j} \right\rfloor$. We consider two approaches according to whether $j < k$ or $j = k$. If at least one of the α_i's for $2 \leqslant i \leqslant k$ is null, then Case 1 is applied.

Case 1 when $j < k$ or when at least one α_i is null: An application of BDH's method.

The first approach consists in computing the integers a_i for all $1 \leqslant i \leqslant k$ such that:

$$r_i = u \cdot \alpha_i + a_i \quad . \tag{5.5}$$

By definition of u, we have $a_i \geqslant 0$ for $1 \leqslant i \leqslant k$. Therefore, we can write

$$N = \left(\prod p_i^{\alpha_i}\right)^u \left(\prod p_i^{a_i}\right) = P^u Q \quad ,$$

where $P = \prod p_i^{\alpha_i}$ and $Q = \prod p_i^{a_i}$. According to Theorem 42, one can therefore apply the BDH's factorization method on $N = P^u Q$ to recover P and Q in polynomial time in $\log N$ if

$$u = \Omega(\log Q) \ . \tag{5.6}$$

Since $\log Q = \sum_{i=1}^{k} (a_i \log p_i) < \sum_{i=1}^{k} (a_i) \log p$, condition (5.6) amounts to:

$$\frac{r_j}{\alpha_j} = \Omega\left((\max_{1 \leqslant i \leqslant k} a_i) \log p \right) \quad . \tag{5.7}$$

Let's give an upper-bound on the integers a_i. By using (5.4) and (5.5) we obtain that

$$\gamma_i = r_1 \cdot \alpha_i - r_i \cdot \alpha_1 = (u \cdot \alpha_1 + a_1) \cdot \alpha_i - (u \cdot \alpha_i + a_i) \cdot \alpha_1 = a_1 \cdot \alpha_i - a_i \cdot \alpha_1 \ . \tag{5.8}$$

Therefore, since we have $0 \leqslant \alpha_i \leqslant \alpha_1$ for $2 \leqslant i \leqslant k$, we deduce that a_i for $2 \leqslant i \leqslant k$ is bounded as follows:

$$a_i = \frac{a_1 \cdot \alpha_i - \gamma_i}{\alpha_1} \leqslant \frac{a_1 \cdot \alpha_i + |\gamma_i|}{\alpha_1} \leqslant a_1 + \frac{|\gamma_i|}{\alpha_1} \quad . \tag{5.9}$$

Furthermore, by using the inequality $a_j < \alpha_j$ and relation (5.8) with $i = j$ which gives $a_1 = (a_j \cdot \alpha_1 + \gamma_j)/\alpha_j$, one deduces that:

$$a_i \leqslant a_1 + \frac{|\gamma_i|}{\alpha_1} \leqslant \frac{a_j \cdot \alpha_1 + \gamma_j}{\alpha_j} + \frac{|\gamma_i|}{\alpha_1} < \alpha_1 + \frac{|\gamma_j|}{\alpha_j} + \frac{|\gamma_i|}{\alpha_1} \qquad \text{for} \quad 1 \leqslant i \leqslant k \ .$$

As a consequence, by using this upper bound on a_i and the following relation derived from (5.4):

$$\frac{r_j}{\alpha_j} = \frac{r_1}{\alpha_1} - \frac{\gamma_j}{\alpha_1 \alpha_j} \ , \tag{5.10}$$

condition (5.7) becomes:

$$\frac{r_1}{\alpha_1} = \Omega\left(\frac{|\gamma_j|}{\alpha_1 \alpha_j} + (\alpha_1 + \frac{|\gamma_j|}{\alpha_j} + \max_i(\frac{|\gamma_i|}{\alpha_1})) \log p \right) \quad .$$

The term $\frac{|\gamma_j|}{\alpha_1\alpha_j}$ is absorbed by the larger term $\frac{|\gamma_j|}{\alpha_j}\log p$, and is therefore ignored in the sequel. Furthermore, since all $|\gamma_i|$ are bounded by the same value and since $\alpha_j \leqslant \alpha_1$, the term $\max_i(\frac{|\gamma_i|}{\alpha_1})$ is also absorbed by $\frac{|\gamma_j|}{\alpha_j}$ and is neglected in the following.

Therefore, by multiplying the previous condition by α_1 in both sides, we get:

$$r_1 = \Omega\left((\alpha_1^2 + \frac{|\gamma_j|\alpha_1}{\alpha_j})\log p\right) \quad . \tag{5.11}$$

Relation (5.11) involves the ratio α_1/α_j. Yet, one shows that $\alpha_1/\alpha_j = \mathcal{O}(r_1/r_j)$. Indeed, by using (5.4) and the bound $|\gamma_j| < r_1 \cdot \varepsilon$, one deduces that

$$\frac{\alpha_1}{\alpha_j} = \frac{r_1}{r_j} - \frac{\gamma_j}{r_j\alpha_j} \leqslant \frac{r_1}{r_j} + \frac{|\gamma_j|}{r_j\alpha_j} < \frac{r_1}{r_j}\left(1 + \frac{\varepsilon}{\alpha_j}\right) < 2 \cdot \frac{r_1}{r_j} \ .$$

Since $j < k$, we have $r_j \geqslant r_{k-1}$, and therefore, we have $r_1/r_j \leqslant r_1/r_{k-1}$. Furthermore, by definition, we have $r_1/r_{k-1} \leqslant \log^\rho p$. As a consequence, in condition (5.11), one can replace the ratio α_1/α_j by $\log^\rho p$, which gives:

$$r_1 = \Omega\left((\alpha_1^2 + |\gamma_j|\log^\rho p)\log p\right) \quad . \tag{5.12}$$

By applying Theorem 20 with $\varepsilon = \log^{-(\rho+1)} p$, we get the following bounds:

$$\begin{cases} \alpha_1 \leqslant 2^{\frac{k(k-1)}{4}} \log^{(k-1)(\rho+1)} p \quad , \\ |\gamma_j| \leqslant r_1 \cdot \log^{-(\rho+1)} p \quad . \end{cases}$$

Therefore the term $|\gamma_j|\log^\rho p \cdot \log p$ in condition (5.12) can be bounded by r_1. Indeed, we have

$$|\gamma_j|\log^\rho p \cdot \log p \leqslant r_1 \cdot \log^{-(\rho+1)} p \cdot \log^\rho p \cdot \log p \leqslant r_1 \quad .$$

As a consequence, condition (5.12) becomes $r_1 = \Omega(\alpha_1^2 \cdot \log p)$. By using the upper-bound on α_1, one gets:

$$r_1 = \Omega\left(2^{\frac{k(k-1)}{2}} \cdot \log^{2(k-1)(\rho+1)} p \cdot \log p\right) \quad ,$$

which allows to retrieve the final condition given in this Lemma by neglecting the term $2^{k(k-1)/2}$.

Case 2 when $\lfloor r_k/\alpha_k \rfloor < \lfloor r_i/\alpha_i \rfloor$ for all $1 \leqslant i \leqslant k-1$: An application of Coppersmith's method.

In this case, as in Section 5.2, one cannot apply previous method since at least one of the integers a_i would be negative. The alternative approach consists in computing the integer value $u' > 0$ as follows:

$$u' = \max\left\{\left\lceil \frac{r_i}{\alpha_i} \right\rceil, \text{ for } 1 \leqslant i \leqslant k\right\} \quad .$$

As before, we take the smallest index j such that $u' = \lceil r_j/\alpha_j \rceil$. Inevitably, one has $j < k$. Indeed, since we are in Case 2, we have $\lfloor r_k/\alpha_k \rfloor < \lfloor r_i/\alpha_i \rfloor$ for all $1 \leqslant i \leqslant k-1$. This means that there exists an index j with $j < k$ such that $\lceil r_j/\alpha_j \rceil \geqslant \lceil r_k/\alpha_k \rceil$.

One computes the integers a'_i for all $1 \leqslant i \leqslant k$ such that:

$$r_i = u' \cdot \alpha_i - a'_i \quad . \tag{5.13}$$

By definition of u', we have $a'_i \geqslant 0$ for $1 \leqslant i \leqslant k-1$. Furthermore, since $j < k$, we have $u' \geqslant \lceil r_k/\alpha_k \rceil$, and therefore $a'_k \geqslant 0$. Thusly, we can write

$$N = \left(\prod p_i^{\alpha_i}\right)^{u'} \left(\prod p_i^{-a'_i}\right) = P^{u'} Q'^{-1} \quad ,$$

where $P = \prod p_i^{\alpha_i}$ and $Q' = \prod p_i^{a'_i}$. Similarly to Case 2 in Section 5.2, according to Theorem 22, one can apply Coppersmith's factorization method on $N = P^{u'} Q'$ to recover P and Q' in polynomial time in $\log N$ if

$$u' = \Omega(\log Q') \ . \tag{5.14}$$

As before, one can easily show that both conditions (5.6) and (5.14) are equivalent. For this, one should first emphasize that in both cases, we have $u = \Omega(r_1/\alpha_1)$ and $u' = \Omega(r_1/\alpha_1)$, which ensures that the bounds on u and u' are similar. It remains to show that $\log Q' = \mathcal{O}(\log Q)$. To this aim, we first show that the upper-bound on a'_i and a_i for $1 \leqslant i \leqslant k$ is similar. Indeed, by using (5.4) and (5.13) we obtain that

$$\gamma_i = r_1 \cdot \alpha_i - r_i \cdot \alpha_1 = (u' \cdot \alpha_1 - a'_1) \cdot \alpha_i - (u' \cdot \alpha_i - a'_i) \cdot \alpha_1 = -a'_1 \cdot \alpha_i + a'_i \cdot \alpha_1 \ . \tag{5.15}$$

Therefore, since we have $0 \leqslant \alpha_i \leqslant \alpha_1$ for $2 \leqslant i \leqslant k$, we deduce that a'_i for $2 \leqslant i \leqslant k$ is bounded as follows:

$$a'_i = \frac{a'_1 \cdot \alpha_i - \gamma_i}{\alpha_1} \leqslant \frac{a'_1 \cdot \alpha_i + |\gamma_i|}{\alpha_1} \leqslant a'_1 + \frac{|\gamma_i|}{\alpha_1} \quad . \tag{5.16}$$

Furthermore, by using the inequality $a'_j < \alpha_j$ and relation (5.15) with $i = j$ which gives $a'_1 = (a'_j \cdot \alpha_1 + \gamma_j)/\alpha_j$, one deduces that:

$$a'_i \leqslant a'_1 + \frac{|\gamma_i|}{\alpha_1} \leqslant \frac{a'_j \cdot \alpha_1 + \gamma_j}{\alpha_j} + \frac{|\gamma_i|}{\alpha_1} < \alpha_1 + \frac{|\gamma_j|}{\alpha_j} + \frac{|\gamma_i|}{\alpha_1} \qquad \text{for} \quad 1 \leqslant i \leqslant k \ .$$

Since the upper-bounds on $\alpha_1, \alpha_i, |\gamma_i|$ are identical in Case 1 and Case 2 and since we have $j < k$ in either cases, one gets the same upper-bound on a'_i as for a_i and we deduce that $\log Q' = \mathcal{O}(\log Q)$.

Note that the knowledge of P and Q' allows always to recover a non-trivial factor of N. Indeed, since we have $a'_j < \alpha_j \leqslant r_j$, we deduce that Q' cannot be a multiple of N

and it is therefore ensured that $\gcd(N, Q')$ is a non-trivial factor of N.

As a conclusion, we have shown that in Case 1 and Case 2, the associated approach (BDH or Coppersmith) can recover a non-trivial factor of N in polynomial time in $\log N$ under the condition on r_1 given in the present Lemma, which depends on the ratio r_1/r_{k-1}.

□

5.3.2 Factoring with Gaps

Our next lemma shows that if there is a sufficient gap between two consecutive exponents r_t and r_{t+1}, then the condition on r_1 to factorize N in polynomial time will only depend on the t first exponents, and not on $r_{t+1}, \ldots, r_k$. We also give a recursive bound on this gap.

Lemma 49. *Let $N = \prod_{i=1}^{k} p_i^{r_i}$ for a fixed $k \geqslant 2$ with $r_1 \geqslant r_2 \geqslant \cdots \geqslant r_k$ and $p = \max\{p_i, 1 \leqslant i \leqslant k\}$. Let t be an integer such that $1 \leqslant t \leqslant k-1$, and ρ_t be such that*

$$r_1/r_t \leqslant \log^{\rho_t} p$$

and

$$\frac{r_1}{r_{t+1}} > \log^{\rho_{t+1}} p = \log^{(t-1)(\rho_t+1)+1} p \quad .$$

Then one can recover a non-trivial factor of N in polynomial time in $\log N$ if

$$r_1 = \Omega\left(\log^{2(t-1)(\rho_t+1)+1} p\right) \quad .$$

Proof. As previously we can assume that the exponents r_i's are known, otherwise we can recover them by exhaustive search in time $\mathcal{O}(\log^k N)$: for a fixed k this is still polynomial in $\log N$.

In the first instance, we only consider the first t exponents $r_1, r_2, \ldots, r_t$. By applying Theorem 20 with $r_1, r_2, \ldots, r_t$ and

$$\varepsilon = \log^{-(\rho_t+1)} p \quad , \tag{5.17}$$

one can recover a non-trivial factor of N in polynomial time if $r_1 = \Omega(\log^{2(t-1)(\rho_t+1)+1} p)$. Indeed, the proof follows the one of Lemma 48 with $k = t$. The difference is that we always perform Case 1 even if $j = t$. Thus, the fact $r_j \geqslant r_{t-1}$ is not true anymore, since one can have $r_j = r_t$. As a consequence, one has an upper-bound on r_1/r_t rather than r_1/r_{t-1} and this is why we take $\rho = \rho_t$.

In order for the condition on r_1 not to be modified by also considering the integers $r_{t+1}, \ldots, r_k$, one needs those integers to be small enough so that one can simply take

$\alpha_i = 0$ and one would get $a_i = r_i$ for $t+1 \leqslant i \leqslant k$. As a consequence, because $r_{t+1} \geqslant \cdots \geqslant r_k$, we would have:

$$\max_{t+1 \leqslant i \leqslant k} a_i = \max_{t+1 \leqslant i \leqslant k} r_i = r_{t+1} \ .$$

Therefore, one would have:

$$(\max_{1 \leqslant i \leqslant k} a_i) \cdot \log p \ \leqslant \ (\max_{1 \leqslant i \leqslant t} a_i + \max_{t+1 \leqslant i \leqslant k} a_i) \cdot \log p \ \leqslant \ (\max_{1 \leqslant i \leqslant t} a_i + r_{t+1}) \cdot \log p \ .$$

Accordingly, condition (5.6) would amount to:

$$\frac{r_1}{\alpha_1} = \Omega \left((\max_{1 \leqslant i \leqslant t} a_i + r_{t+1}) \cdot \log p \right) \ .$$

This condition, without the term r_{t+1}, leads to the condition on r_1 given in the present Lemma. Therefore, in order for the term r_{t+1} not to change the condition on r_1, one needs that $r_1/\alpha_1 > r_{t+1} \log p$, or equivalently that

$$\frac{r_1}{r_{t+1}} > \alpha_1 \log p \ . \tag{5.18}$$

Yet, according to Theorem 20 with ε given in (5.17), one has the following upper bound on α_1:

$$\alpha_1 < \varepsilon^{-(t-1)} = \log^{(t-1)(\rho_t+1)} p \ .$$

Therefore, by using this upper-bound on α_1 in condition (5.18), one deduces that if

$$\frac{r_1}{r_{t+1}} > \log^{(t-1)(\rho_t+1)+1} p \ ,$$

and $r_1/r_t \leqslant \log^{\rho_t} p$, then one can recover a non-trivial factor of N in polynomial time in $\log N$ under the condition

$$r_1 = \Omega \left(\log^{2(t-1)(\rho_t+1)+1} p \right) \ .$$

□

Thusly, Lemma 49 highlights that if there is a sufficient gap between two consecutive exponents r_t and r_{t+1}, then the condition on r_1 to factorize N in polynomial time is $r_1 = \Omega \left(\log^{2(t-1)(\rho_t+1)+1} p \right)$, which only depends on the t first exponents.

5.3.3 An Iterative Definition of Function ρ_t

In order to prove our main Theorem 47, one needs to highlight a last Lemma which allows to give an iterative definition of the recursive function $\rho_{t+1} = (t-1)(\rho_t + 1) + 1$.

Lemma 50. *Let ρ be the function which is recursively defined as follows:*

$$\begin{cases} \rho_1 = 0 \quad , \\ \rho_{t+1} = (t-1)(\rho_t + 1) + 1 \quad , \quad \textit{for} \quad t \geqslant 1 \quad . \end{cases}$$

Then, the function ρ can be iteratively defined as follows:

$$\begin{cases} \rho_1 = 0 \quad , \\ \rho_t = 1 + 2 \sum_{i=1}^{t-2} \prod_{j=i}^{t-2} j \quad , \quad \textit{for} \quad t \geqslant 2 \quad . \end{cases}$$

Proof. We start by giving an intuition of the iterative formula given in Lemma 50. Indeed, since $\rho_{t+1} = (t-1)(\rho_t + 1) + 1$, we have:

$$\begin{aligned} \rho_t &= (t-2)(\rho_{t-1}+1)+1 = (t-2)\left((t-3)(\rho_{t-2}+1)+2\right)+1 \\ &= (t-2)(t-3)(\rho_{t-2}+1) + 2(t-2) + 1 \\ &= (t-2)(t-3)((t-4)(\rho_{t-3}+1)+2) + 2(t-2) + 1 \\ &= (t-2)(t-3)(t-4)(\rho_{t-3}+1) + 2(t-2)(t-3) + 2(t-2) + 1 \\ &= \quad \dots \\ &= 2(t-2)(t-3)\dots(1) + 2(t-2)(t-3)\dots(2) + \dots + 2(t-2)(t-3) + 2(t-2) + 1 \\ &= 1 + 2\sum_{i=1}^{t-2}\prod_{j=i}^{t-2} j \end{aligned}$$

We now rigorously prove Lemma 50 by recurrence on t.

Base Case: The case $t = 1$ is trivial. For $t = 2$, we have $\rho_2 = 1 + 2 \sum_{i=1}^{0} \prod_{j=i}^{0} j = 1$. And by using the recursive definition, we obtain $\rho_2 = (1-1)(\rho_1 + 1) + 1 = 1$.

Inductive Step: We assume that for an arbitrary t, we have $\rho_t = 1 + 2 \sum_{i=1}^{t-2} \prod_{j=i}^{t-2} j$. We show that the relation is still verified for $t+1$, *i.e.* that we have: $\rho_{t+1} = 1 + 2 \sum_{i=1}^{t-1} \prod_{j=i}^{t-1} j$.

Since we have the relation $\rho_{t+1} = (t-1)(\rho_t + 1) + 1$, by using the recurrence assumption for ρ_t, we deduce that

$$\begin{aligned} \rho_{t+1} &= (t-1)([1 + 2\sum_{i=1}^{t-2}\prod_{j=i}^{t-2} j] + 1) + 1 \;=\; 2(t-1)(1 + \sum_{i=1}^{t-2}\prod_{j=i}^{t-2} j) + 1 \\ &= 2((t-1) + \sum_{i=1}^{t-2}\prod_{j=i}^{t-1} j) + 1 \;=\; 1 + 2\sum_{i=1}^{t-1}\prod_{j=i}^{t-1} j \,, \end{aligned}$$

which concludes the proof.

□

5.3.4 Proof of the Generalization Theorem

One can now prove Theorem 47. Namely, we show that one can factorize $N = \prod_{i=1}^{k} p_i^{r_i}$ with $r_1 \geq r_2 \geq \cdots \geq r_k$ if r_1 is large enough. More precisely, we show that either all the r_i's are large enough, or there must be a gap between r_t and r_{t+1} for some $t < k$. As previously we can assume that the exponents r_i's are known. Otherwise we can recover them by exhaustive search in time $\mathcal{O}(\log^k N)$: for a fixed k this is still polynomial in $\log N$.

An Illustration:

We first illustrate our technique iteratively. Let's consider the ratios r_1/r_t for $1 \leqslant t \leqslant k-1$. Obviously, one has $r_1/r_1 = 1 \leqslant \log^{\rho_1} p$, which leads to $\rho_1 = 0$. Thusly, in the case where $r_1/r_2 > \log^{\rho_2} p = \log^{(1-1)(\rho_1+1)+1} p = \log p$, by applying Lemma 49, one can recover a non-trivial factor of N if $r_1 = \Omega(\log^{2(1-1)(\rho_1+1)+1} p) = \Omega(\log p)$. On the other side, if $r_1/r_2 \leqslant \log^{\rho_2} p$, then one has to check r_3. Namely, if $r_1/r_3 > \log^{\rho_3} p = \log^{(2-1)(\rho_2+1)+1} p = \log^3 p$, then by applying Lemma 49, one can recover a non-trivial factor of N if $r_1 = \Omega(\log^{2(2-1)(\rho_2+1)+1} p) = \Omega(\log^5 p)$. On the other side, if $r_1/r_3 \leqslant \log^{\rho_3} p$, then one has to check r_4, and so forth, and so on until $t = k-2$ with $r_1/r_{k-2} \leqslant \log^{\rho_{k-2}} p$. For this last case, if $r_1/r_{k-1} > \log^{\rho_{k-1}} p$, then one can apply Lemma 49 as before with the condition $r_1 = \Omega(\log^{2(k-3)(\rho_{k-2}+1)+1} p)$. But in the case where $r_1/r_{k-1} \leqslant \log^{\rho_{k-1}} p$, then one applies Lemma 48 with $\rho = \rho_{k-1}$, which allows to recover a non-trivial factor of N if $r_1 = \Omega(\log^{2(k-1)(\rho_{k-1}+1)+1} p)$. According to Lemma 50, one has $\rho_{k-1} = 1 + 2\sum_{i=1}^{k-3} \prod_{j=i}^{k-3} j$. Eventually, plugging this value for ρ_{k-1} in previous relation allows to retrieve the condition $r_1 = \Omega(\log^{\theta_k} p)$ with θ_k given in Theorem 47 to recover a non-trivial factor of N in polynomial time in $\log N$.

A constructive approach:

More rigorously, we define $\rho_1 = 0$ and for all $1 \leq t \leq k-1$:

$$\rho_{t+1} = (t-1)(\rho_t + 1) + 1 \quad . \tag{5.19}$$

We consider the following possible cases on the exponents r_i:

$$1 \leq t \leq k-2, \ \mathsf{Case}\ t: \quad \begin{cases} r_1/r_t & \leq \log^{\rho_t} p \\ r_1/r_{t+1} & > \log^{\rho_{t+1}} p \end{cases}$$

$$\mathsf{Case}\ k-1: \quad r_1/r_{k-1} \leq \log^{\rho_{k-1}} p$$

It is easy to check that Case 1 to Case $k-1$ cover all possible cases. Namely if the second inequality in Case t is not satisfied, we obtain:

$$r_1/r_{t+1} \leq \log^{(t-1)(\rho_t+1)+1} p \quad ,$$

which implies using (5.19) that the first inequality $r_1/r_{t+1} \leq \log^{\rho_{t+1}} p$ in Case $t+1$ must be satisfied. Since the first inequality in Case 1 is automatically satisfied, this implies that one of Case t must apply, for some $1 \leq t \leq k-2$.
Eventually if the second inequality in Case $t = k-2$ is not satisfied then the single inequality in Case $k-1$ must be satisfied.

Thusly, if there exists an integer t with $1 \leqslant t \leqslant k-2$ such that $r_1/r_t \leqslant \log^{\rho_t} p$ and $r_1/r_{t+1} > \log^{\rho_{t+1}} p$, we are in Case t, and then one can apply Lemma 49. This allows to recover a non-trivial factor of N in polynomial time if

$$r_1 = \Omega(\log^{2(t-1)(\rho_t+1)+1} p) \quad ,$$

where according to Lemma 50, we have

$$\rho_t = 1 + 2\sum_{i=1}^{t-2}\prod_{j=i}^{t-2} j \quad .$$

Since ρ is an increasing function, one has $\rho_{k-2} \geqslant \rho_t$ for $1 \leqslant t \leqslant k-2$. Therefore, among the "Case t", the largest condition on r_1 occurs for Case $t = k-2$, which gives the condition

$$r_1 = \Omega(\log^{2(k-3)(\rho_{k-2}+1)+1} p) = \Omega(\log^{4(k-3)\left(1+\sum_{i=1}^{k-4}\prod_{j=i}^{k-4} j\right)+1}) \quad .$$

However, if such a value t does not exist, then it means that for all t such that $1 \leqslant t \leqslant k-1$, one has $r_1/r_t \leqslant \log^{\rho_t} p$, and in particular, one has $r_1/r_{k-1} \leqslant \log^{\rho_{k-1}} p$. Therefore, we are in Case $k-1$, and one can apply Lemma 48 with $\rho = \rho_{k-1}$, which allows to recover a non-trivial factor of N if

$$r_1 = \Omega(\log^{2(k-1)(\rho_{k-1}+1)+1} p) \quad , \tag{5.20}$$

where according to Lemma 50, one has $\rho_{k-1} = 1 + 2\sum_{i=1}^{k-3}\prod_{j=i}^{k-3} j$ for $k \geqslant 3$. Therefore, one gets the stronger condition

$$r_1 = \Omega(\log^{2(k-1)(\rho_{k-1}+1)+1} p) = \Omega(\log^{4(k-1)\left(1+\sum_{i=1}^{k-3}\prod_{j=i}^{k-3} j\right)+1}) = \Omega(\log^{\theta_k} p) \quad .$$

This allows us to retrieve the condition on r_1 for $k \geqslant 3$ given in Theorem 47 to recover a non-trivial factor of N in polynomial time in $\log N$.

Furthermore, we have $\rho_1 = 0$ according to Lemma 50. Therefore, for the case $k = 2$, Condition (5.20) becomes

$$r_1 = \Omega(\log^{2(k-1)(\rho_{k-1}+1)+1} p) = \Omega(\log^{2(1)(0+1)+1} p) = \Omega(\log^3 p) \quad .$$

Obviously, this is the same condition as the one obtained in Section 5.2.

First values θ_k in condition $r_1 = \Omega(\log^{\theta_k} p)$:

Thus, in Table 5.1 we provide the first values of ρ_{k-1} and θ_k for a modulus $N = \prod_{i=1}^{k} p_i^{r_i}$ with k prime factors. The condition on the largest exponent r_1 is $r_1 = \Omega(\log^{\theta_k} p)$. Namely, we obtain a polynomial time factorization if $r_1 = \Omega(\log^9 p)$ for $k = 3$, if $r_1 = \Omega(\log^{25} p)$ for $k = 4$, etc.

Table 5.1: Values of ρ_{k-1} and θ_k for a modulus $N = \prod_{i=1}^{k} p_i^{r_i}$ with k prime factors. The condition on the largest exponent r_1 is $r_1 = \Omega(\log^{\theta_k} p)$.

k	2	3	4	5	6
ρ_{k-1}	0	1	3	9	31
$\theta_k = 2(k-1)(\rho_{k-1}+1)+1$	3	9	25	81	321

A more explicit condition for large k:

Eventually, for $k \geqslant 3$ one has the relation:

$$1 + \sum_{i=1}^{k-3} \prod_{j=i}^{k-3} j \leqslant 1 + (k-3)(k-3)! \leqslant (k-2)! \quad ,$$

which gives for large k,

$$\theta_k \leqslant 4(k-1)! + 1 = \mathcal{O}(k!) \quad .$$

Therefore, for large values of k, we obtain the more explicit bound $r_1 = \Omega(\log^{k!} p)$ and this terminates the proof of Theorem 47.

Thus, the bound on r_1 grows exponentially in the number of prime factors k, but for a fixed k, the condition is polynomial in $\log p$. We emphasize that this condition on r_1 is a sufficient but not necessary condition. Indeed, it deals with the worst possible series $r_1, r_2, \ldots, r_k$ which is the one satisfying $r_1/r_i \leqslant \log^{\rho_i} p$ for all $1 \leqslant i \leqslant k$ where ρ_i follows the definition of Lemma 50. But it can be improved following Lemma 49 if there is a sufficient gap between two consecutive r_i. In particular, for the case where $k = 2$, Lemma 49 applied with $t = 1$ allows to recover the improvement proposed in Section 5.2.6 when s is small compared to r, *i.e.* that the condition is $r = \Omega(\log \max(p, q))$ when $r/s > \log \max(p, q)$. This terminates the proof of Theorem 47.

5.4 Speeding-up by Rounding and Chaining

In Chapter 4, we described a method called Rounding and Chaining, which allows to speed-up the *LLL*-reduction performed within Coppersmith's method to find small roots of univariate modular equations. In this section we apply it to our technique for factorising moduli of the form $N = \prod_{i=1}^{k} p_i^{r_i}$.

5.4.1 Rounding

The Rounding method, uses the fact that the diagonal elements in the matrix to be LLL-reduced are all balanced. This property allows to speed up the LLL-reduction by rounding, that is by keeping only the most significant bits of all coefficients in the matrix before LLL-reducing it. That way, the matrix contains much smaller elements than originally.

In our context of factorizing moduli of the form $N = \prod_{i=1}^{k} p_i^{r_i}$, one rewrites N, according to the method used, namely, one has $N = P^u Q$ for BDH's application, and one has $N = P^u/Q$ for Coppersmith's application. Thus one considers both applications separately.

Coppersmith's method

If Coppersmith's method is used, the Rounding technique described in Chapter 4 can straightforwardly be used. Indeed, Coppersmith's method applied to our case amounts to solving the univariate modular equation

$$f(x) = (X \cdot t + x)^u \mod N \quad ,$$

where $X \cdot t$ is the known upper part of P and X is an upper-bound on the searched solution x_0. Therefore, one uses the matrix B drawn in Section 3.1, where $\delta = u$. Thus, as shown in Lemma 30 from Chapter 4 applied for $\delta = u$, the minimal diagonal coefficient of B is X^{um} and the maximal diagonal coefficient is $X^{u-1}N^m$, where m is the positive integer parameter linked to the dimension of the matrix (see Chapter 4 for more details). Therefore, as in Chapter 4, we first size-reduce B (see Chapter 2, Definition 10) to make sure that in each column, all subdiagonal coefficients are smaller than the diagonal coefficient.

Then, one can round the entries of B so that the smallest diagonal coefficient becomes $\lfloor c \rfloor$ where $c > 1$ is a small parameter (the analysis provided in Chapter 4 gave $c = (3/2)^n$ where n is the dimension of matrix B). More precisely, we create a new $n \times n$ triangular matrix $\tilde{B} = (\tilde{b}_{i,j})$ defined by:

$$\tilde{B} = \lfloor cB/X^{um} \rfloor \quad . \tag{5.21}$$

This means that the new matrix $\tilde{B}$ is made of matrix B where all of its coefficients are divided by X^{um}/c. Since the diagonal coefficients $b_{i,i}$ of matrices B are such that $b_{i,i} \geqslant X^{um}$, the diagonal coefficients $\tilde{b}_{i,i}$ of matrix $\tilde{B}$ are such that: $\tilde{b}_{i,i} \geqslant \lfloor cX^{um}/X^{um} \rfloor = \lfloor c \rfloor$. Hence, we LLL-reduce the rounded matrix $\tilde{B}$ instead of B.

Complexity: The original matrix B has entries whose bit-size was $\mathcal{O}(m \log N)$. In the rounded matrix $\tilde{B}$, the elements have bit-size $\mathcal{O}(\log c + \log N)$. Indeed, according to Lemma 30, the ratio between the maximal and the minimal diagonal coefficients of B satisfies:

$$\frac{N^m X^{u-1}}{X^{um}} \geqslant N^{1-\frac{1}{u}} \quad .$$

Therefore, the entries of the new matrix $\tilde{B} = \lfloor cB/X^{um} \rfloor$ are upper bounded by $cN^{1-\frac{1}{u}}$. By taking $um = \mathcal{O}(\log N)$, with the bound X given in Chapter 4, we obtain that the elements have bit-size $\mathcal{O}(\log c + \log N)$.

Since $\log c = \mathcal{O}(\log N)$ when $c = (3/2)^n$, we get $b \leqslant \mathcal{O}(\log N)$. Furthermore, the dimension n of $\tilde{B}$ is the same as the one of B. It follows that the running time of the LLL-reduction is $\mathcal{O}(u^6 m^6 (\log N) + u^5 m^5 (\log N)^2)$ using L^2 which is

$$L^2 : \quad \mathcal{O}((um)^5 (\log N)^2) = \mathcal{O}(\log^7 N)$$

because $u < (\log N)/2 - 1$ and $um = \mathcal{O}(\log N)$; instead of the previous $\mathcal{O}((\log^9 N)/u^2)$. The running time using $\tilde{L}^1$ is $\mathcal{O}((um)^{5+\varepsilon}(\log N) + (um)^{\omega+1+\varepsilon}(\log N)^{1+\varepsilon})$ for any $\varepsilon > 0$ using fast integer arithmetic, where $\omega \leq 2.376$ is the matrix multiplication complexity constant, which gives the complexity:

$$\tilde{L}^1 : \quad \mathcal{O}((um)^{5+\varepsilon}(\log N) + (um)^{\omega+1+\varepsilon}(\log N)^{1+\varepsilon}) = \mathcal{O}(\log^{6+\varepsilon} N) \quad ,$$

instead of the previous $\mathcal{O}((\log^{7+\varepsilon} N)/u)$.

BDH's method

For BDH's method, one uses the matrix B depicted in Section 3.3, where $r = u$. Again, a simple analysis of the diagonal coefficients in matrix B, enlightens that the minimal diagonal coefficient of B is X^{um} and the maximal diagonal coefficient is $X^{u-1}N^m$. These limit coefficients are exactly the same as for Coppersmith's method. Thusly, one can rigorously follow the description of the Rounding method performed for Coppersmith's method.

Complexity: Thusly, the complexity is the same as the one obtained for Coppersmith's method. Namely, the complexity is $\mathcal{O}(\log^7 N)$ when L^2 is used, and $\mathcal{O}(\log^{6+\varepsilon} N)$ when $\tilde{L}^1$ is used.

As a conclusion, since the speed-up depends on the degree u, namely it is $\Theta((\log^2 N)/u^2)$ when using L^2, and $\Theta((\log N)/u)$ when $\tilde{L}^1$ is employed; we emphasize that the higher the degree u, the less significant will be the speed-up. Therefore, since the degree u increases with the number k of primes in order to have a polynomial-time factorization of $N = \prod_{i=1}^{k} p_i^{r_i}$, we deduce that the speed-up becomes less significant when the number k of primes increases.

5.4.2 Chaining

As explained in Section 5.2, in order to reach Coppersmith and BDH's bounds, one should perform some exhaustive search in order to recover the $N^{1/u}$ high order bits of P which are necessary to retrieve the whole value P.

Namely, in Coppersmith's method, we write $N = P^u/Q$ and $P = X \cdot t + x_0$ where $X = \lfloor N^{1/u} \rfloor$ and $|x_0| \leq X$. We obtain the polynomial equation:

$$f_t(x_0) = (X \cdot t + x_0)^u \equiv 0 \mod N \ .$$

Since the integer t is unknown, we do exhaustive search on t for $0 \leq t \leq P/X$, and the solution x_0 is then found for the right value t_0, *i.e.* for the right polynomial f_{t_0}.

In BDH's method, the same applies, namely we write $N = P^u Q$ and $P = X \cdot t + x_0$ where $X = \lfloor N^{1/u} \rfloor$ and $|x_0| \leq X$, and we have

$$f_t(x_0) = (X \cdot t + x_0)^u \equiv 0 \mod P^u \ .$$

Again, we have $0 \leq t \leq P/X$, and we do exhaustive search on t to recover P.

Thus, in both methods, the polynomials constructed from $f_t(x)$ and written in the basis

$$\mathcal{B} = (1, xX^{-1}, (xX^{-1})^2, \ldots, (xX^{-1})^{n-1}) \ ,$$

where n is the dimension of the lattice. For the case $t+1$, one tries to solve the polynomial

$$f_{t+1}(x) = f(X \cdot (t+1) + x) = f(X \cdot t + x + X) = f_t(x + X) \, .$$

Therefore, the shifted polynomials constructed from f_{t+1} are the same as for the case t, but written in the different basis

$$\mathcal{B}' = (1, xX^{-1} + 1, (xX^{-1} + 1)^2, \ldots, (xX^{-1} + 1)^{n-1}) \ .$$

Therefore, if B is the matrix used for the polynomial $f_t(x)$, then according to Lemma 35 from Chapter 4, since one has the property $\mathcal{B}'^T = P \cdot \mathcal{B}^T$ where P is the Pascal matrix, the matrix $B \cdot P$ is a basis of the "next" lattice used for the polynomial $f_{t+1}(x)$ and can be used instead for the LLL-reduction. Therefore, one can chain all LLL-reductions during the exhaustive search, as performed in Chapter 4.2. In particular, we can use a matrix of the form $B^R \cdot P$ where B^R is an LLL-reduced basis of the previous lattice used to solve $f_t(x)$. As highlighted in Chapter 4.2, even if the speed-up is heuristic, it is in practice faster to LLL-reduce such matrices than the original Coppersmith's matrices independently.

Note that as it is done in Chapter 4.2.2, both speed-ups Rounding and Chaining can be combined. Namely, during the chaining loop, the matrices can also be rounded before the LLL-reduction.

5.5 Experiments

5.5.1 Practical Considerations

We have implemented our algorithm using Magma Software V2.19-5. We considered the case of moduli $N = p^r q^s$ with two primes ($k = 2$). As highlighted, in Section 5.2, the sufficient condition for a polynomial time factorization of N is that $r = \Omega(\log^3 max(p, q))$.

Since this condition is somehow difficult to be put into practice with a reasonably large size of p and q, we rather used relatively small r and s in comparison to the theoretical condition. Namely, we considered four moduli $N = p^r q^s$ with $r = 8$, and $s = 1, 3, 5, 7$, with 128-bit primes p and q, which means that moduli N are of bit-length 1152, 1408, 1664 and 1920. As a consequence, in our experiments, one cannot hope to obtain a polynomial-time factorization of N. Nevertheless, it seems to us interesting to run the algorithm with such parameters in order to study its behaviour as a function of the parameter s, for a steady r, and to get some practical timings. Since in Section 5.2 a fraction $1/u$ of the bits of Q is guessed by exhaustive search, for each modulus N we have determined the values of α, β, a and b that minimize the quantity $\log(Q)/u$. Such minimum is reached either by the BDH method (Case 1), or by the Coppersmith method (Case 2), and we have indicated which of both behaves best according to N.

5.5.2 Speed-up by Rounding and Chaining

To speed up the LLL-computation we have implemented the Rounding and Chaining methods recalled in previous section and thoroughly described in Chapter 4. We gave timings corresponding to the best possible dimension (the one which yielded the timeless exhaustive search). Note that the first LLL-reduction is costlier than the subsequent ones, therefore we considered the running time LLL_f of the first LLL reduction, and running time LLL_c of subsequent LLL reductions.

5.5.3 Implementation Results

In Table 5.2 we give the optimal decomposition of N, using either the BDH method (Case 1) or the Coppersmith method (Case 2), the best decomposition between both is highlighted in bold. For each case, we provide the number of bits given, the lattice lattice dimension and the running times LLL_f (first LLL-reduction) and LLL_c (next LLL-reductions). Finally we also estimate the total running time of the factorization, by multiplying LLL_c by 2^t where t is the number of bits given (or alternatively recovered by exhaustive search).

Thusly, for the case where $N = p^8 q$, the decomposition which yields the best condition, *i.e.* which yields the less bits that are needed to be given, is the trivial decomposition $N = (p^1 q^0)^8 (p^0 q^1) = (p)^8 (q)$ where $\alpha = 1, \beta = 0, a = 0$ and $b = 1$, with the application of BDH's method. This amounts to straightforwardly apply [BDHG99] on $N = p^8 q$ (together with the Rounding and Chaining improvement). By applying Algorithm 8, we obtain that the number of bits that should be given is 29. Note that in theory, the number of bits given is $(\log Q)/u = (\log q)/u = 128/8 = 16$, which is smaller than 29. As explained in Section 5.2, what makes the difference between the practice and the theory is that one uses relatively small dimensions for practical timing reasons. Hence, for a dimension 68, we obtain that the first LLL-reduction takes 142 seconds, and all subsequent reductions take 8.6 seconds (note that without the Rounding and Chaining improvement, each reduction would have taken approximately 1000 seconds). Thusly,

the estimated running time of the exhaustive search is $2^{29} \times 8.6$ seconds ≈ 146 years. By testing all possible decompositions with $\alpha < r$ and $\beta \leqslant \alpha$, one can show that this is the decomposition which yields the smallest number of bits given. By way of comparison, the use of Coppersmith's method on $N = (p^4q)^2q^{-1}$ (which is the best decomposition associated to this method), would have yielded a theoretical number of bits given which is $(\log Q)/u = (\log q)/2 = 128/2 = 64$, which is far larger than 16. Note that in practice, we obtain 77 as the number of bits given for this decomposition.

In the case where $N = p^8q^3$, then the best decomposition is $N = (p^2q)^4q^{-1}$, which means that $\alpha = 2, \beta = 1, a = 0$ and $b = 1$, with the application of Coppersmith's method. The theoretical number of bits that should be given is $(\log Q)/u = (\log q)/u = 128/4 = 32$ (in practice it is again higher than that: our experiments with a dimension 61 give 51 bits). Again, by testing all possible decompositions, one can show that this is the decomposition which yields the smallest number of bits given. By way of comparison, a straightforward use of BDH's method on $N = (p)^8(q^3)$, would have yielded a number of bits given which is $(\log Q)/u = (3\log q)/8 = 3 \times 128/8 = 48 > 32$.

The same analysis can be done for the other moduli $N = p^8q^5$ and $N = p^8q^7$ for which we obtain that the best decompositions are respectively $(p^2q)^4q$ with the application of BDH's method and $(pq)^8q^{-1}$ with Coppersmith's method. The number of bits that should be given and the corresponding timings for the *LLL*-reduction are depicted in Table 5.2.

Furthermore, as mentioned in Section 5.2, we emphasize that the number of bits given is smaller for the case where s is small (29 bits for $s = 1$) or when r and s are close (38 bits for $s = 7$) in comparison to more average cases (51 bits for $s = 3$ and 55 bits for $s = 5$).

Table 5.2: Number of bits given, lattice dimension, running time LLL_f of the first LLL, running time LLL_c of subsequent LLLs, and estimated total running time.

	Method	$\mathbf{(p^\alpha q^\beta)^u p^a q^b}$	**bits given**	**dim.**	$\mathrm{LLL_f}$	$\mathrm{LLL_c}$	**Est. time**
$\mathbf{N = p^8q}$	**BDH**	$\mathbf{p^8q}$	**29**	**68**	**142 s**	**8.6 s**	**146 years**
	Copp.	$(p^4q)^2q^{-1}$	77	69	96 s	8.6 s	$4 \cdot 10^{16}$ years
$\mathbf{N = p^8q^3}$	BDH	p^8q^3	51	61	160 s	5.7 s	$4 \cdot 10^8$ years
	Copp.	$\mathbf{(p^2q)^4q^{-1}}$	**51**	**61**	**86 s**	**4.2 s**	$\mathbf{3 \cdot 10^8}$ **years**
$\mathbf{N = p^8q^5}$	**BDH**	$\mathbf{(p^2q)^4q}$	**55**	**105**	**115 s**	**1.3 s**	$\mathbf{2 \cdot 10^9}$ **years**
	Copp.	$(pq)^8q^{-3}$	70	65	141 s	5.8 s	$2 \cdot 10^{14}$ years
$\mathbf{N = p^8q^7}$	BDH	$(pq)^7p$	40	81	319 s	12.2 s	$4 \cdot 10^5$ years
	Copp.	$\mathbf{(pq)^8q^{-1}}$	**38**	**81**	**676 s**	**26 s**	$\mathbf{2 \cdot 10^5}$ **years**

5.5.4 Comparison with ECM

It is well known that the BDH algorithm for factoring $N = p^r q$ is unpractical. Namely the experiments from [BDHG99] show that the BDH algorithm is practical only for relatively small primes p and q, but for such small prime factors the ECM method [Len87] performs much better. Namely for 128-bit primes p and q and $N = p^{10}q$ the predicted runtime of ECM from [BDHG99] is only 7000 hours [BDHG99], instead of 146 years for BDH for $N = p^8 q$. Needless to say, our algorithm for factoring $N = p^r q^s$ is even less practical, as illustrated in Table 5.2, since for $N = p^r q^s$ we need much larger exponents r or s than in BDH for $N = p^r q$.

However the complexity of the ECM factorization algorithm for extracting a prime factor p is $\exp\left((\sqrt{2} + \circ(1))\sqrt{\log p \log\log p}\right)$. Therefore, the ECM scales exponentially, whereas our algorithm scales polynomially. Hence for large enough primes p and q our algorithm (like BDH) must outpace ECM.

Chapter 6

Combined Attack on RSA-CRT: why public verification must not be public?

Contents

The results presented in this chapter are from a joint work with Guillaume Barbu, Alberto Battistello, Guillaume Dabosville, Christophe Giraud, Guénaël Renault and Soline Renner. It was published in the proceedings of the PKC' 13 conference [BBD+13].

Over the last few years, the cryptographic community has investigated the possibility of combining the two main kinds of physical attacks applied to embedded systems, which involve Side-Channel Analysis (SCA) and Fault Injection (FI). This has resulted in a new class of attacks called Combined Attacks (CA) that can defeat implementations which are meant to resist both SCA and FI. However, as far as we know very few CA have

been published since their introduction in 2007, proving the difficulty to conceive such attacks.

In this chapter, we describe a new Combined Attack on a CRT-RSA implementation which is independently resistant against Side-Channel Analysis (assuming that blinding countermeasures are used) and Fault Injection attacks (assuming that a public verification is performed, *i.e.* the signature is verified using the public exponent, before outputting it). Such an implementation is known to resist each and every kind of attack published so far. In particular, it prevents an attacker from obtaining the signature when a fault has been induced during the computation since such a value would allow the attacker to recover the RSA private key by computing the *gcd* of the public modulus and the faulty signature.

However, we demonstrate that when injecting a fault during the signature computation, a value depending on the message and on a multiple of a secret prime is manipulated in plain during the public verification. Therefore, we notice that one can perform a Side-Channel Analysis to gain some information on such a sensitive value. The resulting information is then used to factorize the public modulus, leading to the disclosure of the whole RSA private key. Besides, we exploit lattice-based techniques, and in particular Coppersmith's methods for finding small solutions to polynomial equations [Cop96b, Cop97], to significantly reduce the complexity of our Combined Attack. We also provide simulations that confirm the efficiency of our attack as well as two different countermeasures having a very small impact on the performance of the algorithm.

As it performs a Side-Channel Analysis during a Fault Injection countermeasure to retrieve the secret value, this chapter recalls the need for Fault Injection and Side-Channel Analysis countermeasures as monolithic implementations. That is, both types of countermeasures should be dealt unitedly and not independently.

Roadmap. In Section 6.1 we present the context of application of our attack. In Section 6.2 we describe our new CA on a CRT-RSA implementation that is known to resist both SCA and FI attacks. In Section 6.3 we present the results of our simulations which prove the efficiency of our new attack. We then improve its complexity by using lattice reduction techniques in Section 6.4. Finally, we suggest in Section 6.5 possible countermeasures having a negligible penalty on the performance of the algorithm.

State of the Art. The reminder on CRT-RSA signature together with the corresponding attacks and countermeasures provided in Chapter 1 is a preliminary of the present chapter. Coppersmith's lattice-based method, which is recalled in Chapter 3, will also be used in this chapter.

6.1 Context and Principle

6.1.1 RSA Signature Using the CRT Mode

In embedded systems like smart cards, most RSA implementations use the CRT mode (also called CRT-RSA) which yields an expected speed-up factor of about four [CQ82] in comparison to the use of the Standard mode. We refer the reader to Chapter 1 for a reminder on such modes. Thus, in this chapter, we consider a CRT-RSA Signature implementation. Here, we only recall some notations on such an implementation: the computation of the Signature $S = m^d \bmod N$ is carried out in two sequences which yields $S_p = m^{d_p} \bmod p$ and $S_q = m^{d_q} \bmod q$, and the signature S is the CRT recombination usually done by using Garner's formula [Gar59]: $S = S_q + q(i_q(S_p - S_q) \bmod p)$ where $i_q = q^{-1} \bmod p$.

6.1.2 Countermeasures Against SCA and FI

As stated in Chapter 1, several countermeasures have been developed to protect CRT-RSA embedded implementations against both SCA and FI. In the framework of this chapter, we consider an algorithm protected:

— against SCA by using message and exponent blinding as suggested in [WvWM11], a regular exponentiation algorithm such as the Square Always [CFG+11] and a mask refreshing method along the exponentiation such as the one presented in [DV11]. Moreover, the blinding is kept all along the CRT-recombination.

— against FI by verifying the signature using the public exponent e [BDL97]. In addition, we also use the approach presented in [DGRS09] which mainly consists in checking the result of the verification twice to counteract double FI attacks.

Figure 6.1 depicts the main steps of such an implementation where the k_i's are random values (typically of 64 bits) generated at each execution of the algorithm and S'_p, S'_q and S' represent the blinded version of S_p, S_q and S respectively.

In the following, we assume that the fault injected by the attacker follows either the *bit-fault*, the *stuck-at* or the *unknown constant* fault models (cf. Chapter 1.2). Moreover, we assume the attacker is able to choose which byte of the message is affected by the fault.

As mentioned in Chapter 1.2, injecting a fault during the signature computation leads to a faulty signature that allows the attacker to recover the private key. However in the implementation considered in this chapter, the verification with the public exponent detects such a disturbance and the faulty signature is never revealed to the attacker. The main contribution of this chapter is to show that in this case, an SCA can still allow the attacker to gain enough information on the faulty signature to recover the private key.

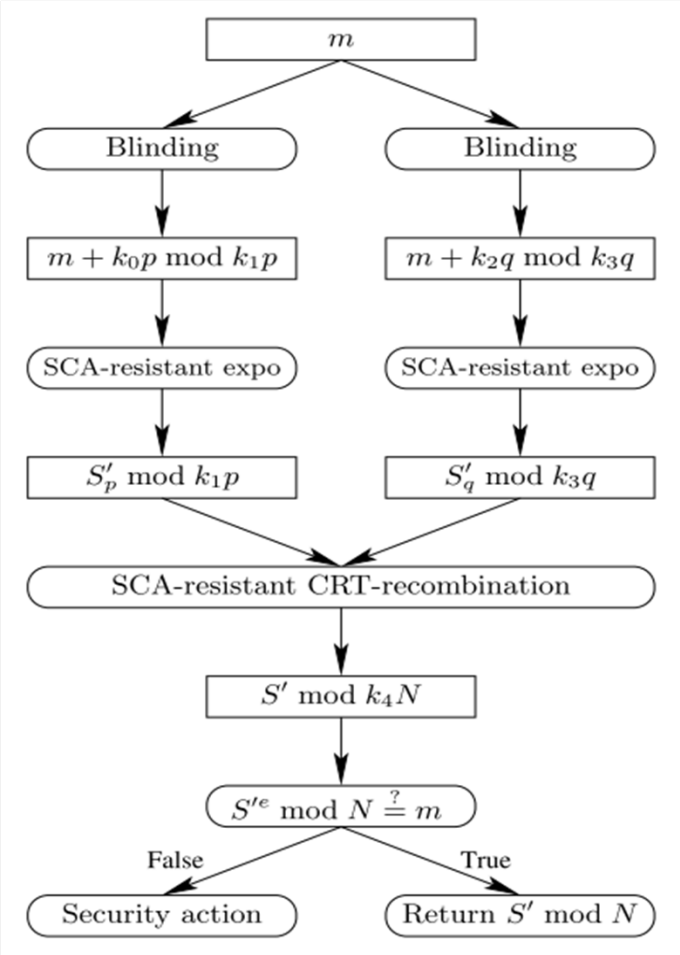


Figure 6.1: Main steps of a CRT-RSA implementation secure against SCA and FI.

6.2 A New Combined Attack on CRT-RSA

At first glance, it seems impossible to perform such an attack during the signature process due to the blinding countermeasure. However by observing Figure 6.1, one may note that the faulty signature $\widetilde{S}$ remains blinded until the end of exponentiation with e modulo N. Therefore if we can express $\widetilde{S}^e \bmod N$ in terms of the message m and of the private key then we can perform an SCA on this value. In the following, we exhibit such a relation allowing us to mount a CA on an SCA-FI-resistant CRT-RSA implementation.

6.2.1 A Useful Relation

As stated in [BDL97], a fault injected in the message before the first operation of S_p (or S_q) modular exponentiation, leads to a faulty signature $\widetilde{S}$ which corresponds to an

erroneous message $\widetilde{S}^e \bmod N = \widetilde{m}$. In order to exploit the fault induced on the message, we make use of the following proposition on which our attack is based.

Proposition 51. *If a fault ε is induced in m such that the faulty message $\widetilde{m}$ is equal to $m + \varepsilon$ at the very beginning of the computation of S_p then*

$$\widetilde{S}^e \equiv m + \varepsilon q i_q \bmod N \ , \tag{6.1}$$

where $\widetilde{S}$ corresponds to the faulty signature.

Proof. By definition of the CRT-RSA signature, we have:

$$\begin{cases} \widetilde{S} \equiv (m+\varepsilon)^d \bmod p \\ \widetilde{S} \equiv m^d \bmod q \end{cases} \tag{6.2}$$

because the computation of S_p is perturbed with a fault ε induced in m such that the faulty message $\widetilde{m}$ is equal to $m + \varepsilon$ and the computation of S_q remains unchanged. It comes then straightforwardly that:

$$\begin{cases} \widetilde{S}^e \equiv m + \varepsilon \bmod p \\ \widetilde{S}^e \equiv m \bmod q \end{cases} \tag{6.3}$$

Finally, applying Gauss recombination to (6.3) in order to get $\widetilde{S}^e \bmod N$, leads to (6.1) since:

$$\begin{aligned} \widetilde{S}^e &\equiv p\, i_p\, m + q\, i_q\, (m+\varepsilon) \bmod N && (6.4)\\ &\equiv (p\, i_p\, m \ + \ q\, i_q\, m) + \varepsilon\, q\, i_q \bmod N && (6.5)\\ &\equiv m \ + \ \varepsilon\, q\, i_q \bmod N \ , && (6.6) \end{aligned}$$

where $i_p = p^{-1} \bmod q$ and $i_q = q^{-1} \bmod p$.

□

One may note that a similar relation holds if m is disturbed at the very beginning of S_q computation due to the symmetrical roles of p and q in both branches of the CRT-RSA. For the sake of simplicity, we will use the case where S_p computation is disturbed in the rest of this chapter.

6.2.2 Recovering the Private Key

Following the attack's principle depicted in Section 6.1 and using Proposition 51, we will now present in detail the main steps of our attack.

Firstly, the attacker asks the embedded device to sign several messages m_i through a CRT-RSA implemented as described in Section 6.1. For each signature, the computation of S_q is performed correctly and a constant additive error ε is injected on the message m_i at the beginning of each S_p computation. Then during each signature verification,

the attacker monitors the corresponding side-channel leakage $\mathcal{L}_i$ which represents the manipulation of $\widetilde{S}_i^e \bmod N$.

From Proposition 51, we know that there exists a sensitive value k satisfying the relation $\widetilde{S}_i^e \bmod N = m_i + k$. Therefore, the attacker will perform a CPA to recover this sensitive value by computing the Pearson correlation coefficient $\rho_k(m_i + k, \mathcal{L}_i)$ for all the possible values of k (cf. Chapter 1.2).

Depending on the set $\{(m_i, \widetilde{S}_i^e \bmod N)\}_i$, it follows from Relation (6.1) that k will be equal either to $\varepsilon q i_q \bmod N$ or to $\varepsilon q i_q \bmod N - N$. Therefore, the value $\widehat{k}$ producing the strongest correlation at the end of the CPA will be one of these two values. Once $\widehat{k}$ recovered, the attacker must then compute the *gcd* between $\widehat{k}$ and N, which leads to the disclosure of q. From this value, the private key is straightforwardly computed.

Regarding the practicality of our fault model (i.e. a constant additive fault), one may note that by fixing a small part of the message (e.g. a byte), the disturbance of such a part in either the stuck-at, the bit-flip or the unknown constant fault model results in a constant additive error during the different signature computations. Therefore our fault model is definitely valid if the attacker can choose the messages to sign, or even if she can only have the knowledge of the messages and attack only those with a given common part.

Finally, one may note that it is not possible to perform a statistical attack targeting the full value of k at once due to its large size (i.e. $\lceil \log_2(N) \rceil$ bits). However, one can attack each subpart of this value, for instance by attacking byte per byte starting with the least significant one in order to be able to propagate easily the carry. It is worth noticing that CPA only applies when the corresponding part of the message varies. Therefore, if the attacker fixes the MSB of the message, then the corresponding set of measurements can be used to recover the whole but last byte of $\widehat{k}$. In such a case, a brute force search can be used to recover the missing byte.

In the next section, we present simulations of our attack which prove the efficiency of our method and which are based on the attacker's capability to inject the same fault and on the noise of the side-channel measurements.

6.3 Experiments

The success of the attack presented in Section 6.2 relies on the ability of the attacker to both measure the side-channel leakage of the system during the signature verification and induce the same fault ε on the different manipulated messages.

In order to evaluate the effectiveness of this attack, we have experimented it on simulated curves of the side-channel leakage $\mathcal{L}$, according to the following leakage model:

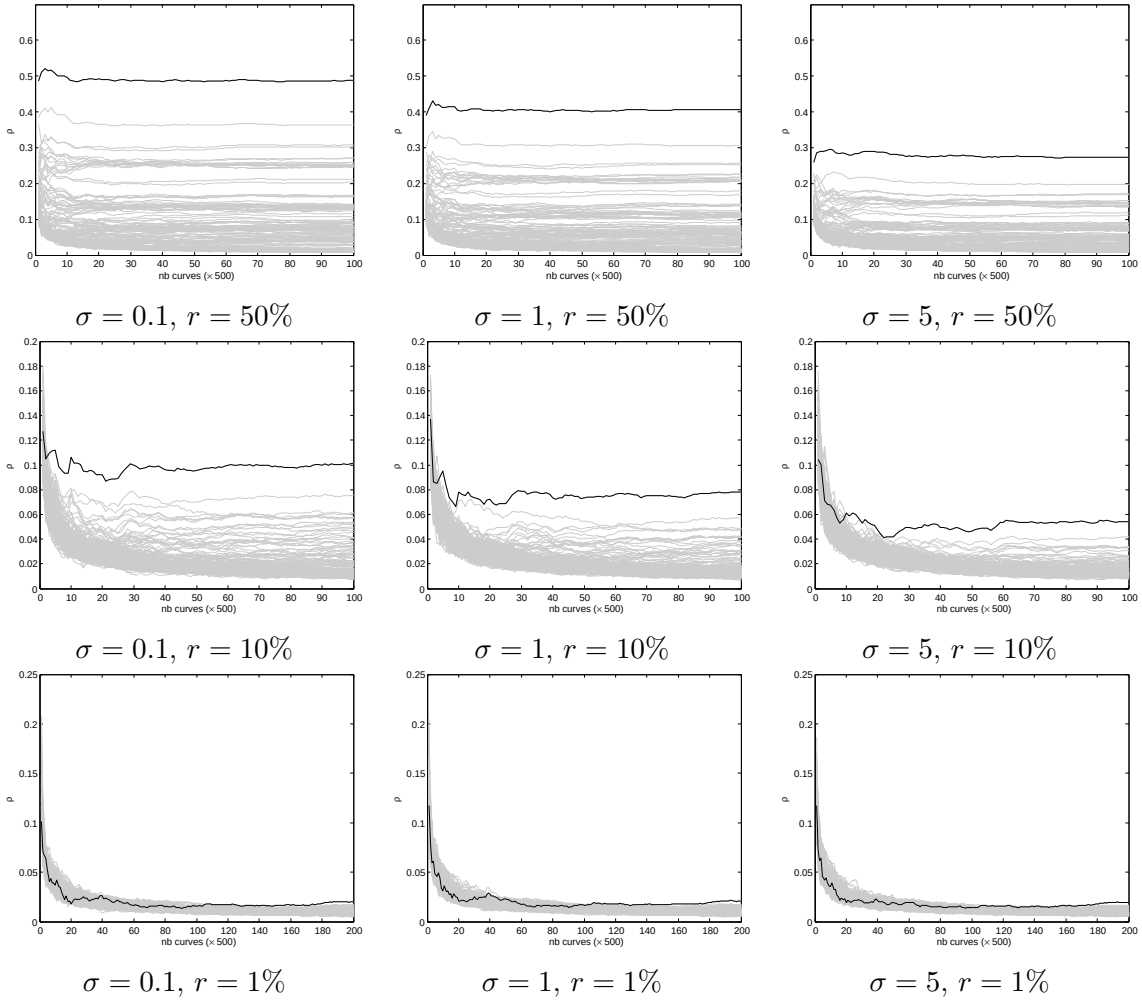
$$\mathcal{L}(d) = HW(d) + \mathcal{N}(\mu, \sigma) \tag{6.7}$$

with $\mathcal{N}(\mu, \sigma)$ a Gaussian noise of mean μ and standard deviation σ, and $HW(d)$ the Hamming weight function evaluated for the manipulated data d. In the framework of our experiments, we consider that the processor manipulates 8-bit words and we use three different levels of noise, namely $\sigma = 0.1$, 1 and 5.

As well as the side-channel leakage, the faults were also simulated by setting the most significant word of the message m to all-0 at the very beginning of the S_p computation. These faults were induced with a given success rate r, varying in our different experiment campaigns (namely 50%, 10% and 1%).

Depending on the experimental settings, all the different words of the secret value will be equivalently correlated with the simulated curves. The graphs presented in Figure 6.2 present the convergence of the correlation for each possible value k of one particular byte (the 5^{th} least-significant byte) of the secret depending on the number of side-channel measurements with different simulation settings σ and r.

Figure 6.2: Convergence of the correlation for the 256 possible values k_i for the secret (the correct one being depicted in black) depending on the number of side-channel measurements ($\times$500) for different levels of noise σ and fault injection success rates r.



$\sigma = 0.1$, $r = 50\%$ $\sigma = 1$, $r = 50\%$ $\sigma = 5$, $r = 50\%$

$\sigma = 0.1$, $r = 10\%$ $\sigma = 1$, $r = 10\%$ $\sigma = 5$, $r = 10\%$

$\sigma = 0.1$, $r = 1\%$ $\sigma = 1$, $r = 1\%$ $\sigma = 5$, $r = 1\%$

As exposed in Figure 6.2, the number of traces required to recover the secret value

depends essentially on the fault injection success rate. This comes from the fact that every wrongly-faulted computation can be considered as noise in the scope of our statistical analysis. The number of curves required to retrieve the secret word grows as the fault injection success rate decreases and to a fewer extent as the noise of the side-channel leakage increases.

With regard to the results obtained when $\sigma = 5$ and $r = 10\%$, which appear to be plausible values in practice, it took us 3.35 seconds to retrieve one byte of the secret value by performing the CPA on $15,000$ curves of 128 points each [1]. Assuming a genuine curve should be made of at least $5,000$ points, we can estimate the time required to practically perform the attack to about 1 minute 5 seconds per byte. That is to say, it takes about 2 hours 20 minutes to recover the complete secret value if we consider a 1024-bit RSA module.

For the sake of clarity, we restrained the experiments presented here to the case where the processor manipulates 8-bit words, and thus ε is an 8-bit error. The same experiments have been run for processor word-size up to 32 bits with success. Besides, about the same number of curves were necessary for the CPA to highlight the correct secret byte.

Section 6.4 shows how it is possible to considerably reduce the complexity of our attack thanks to the use of lattice techniques.

6.4 Reducing the Attack Complexity Using Coppersmith's Methods

This section aims at improving the attack complexity using Coppersmith's methods. It is in line with the problem of factorizing N knowing half part of prime p (or q), that was solved in [Cop97]. With respect to our case, we highlight that if the CA presented in Section 6.2 provides about half of the secret $\varepsilon q i_q \bmod N$, then the other half part can be straightforwardly computed by solving a well-designed modular polynomial equation that we elaborate in the sequel. Besides, we deal with two cases (ε known and unknown), depending on the fault model that is considered.

6.4.1 Bringing Up the Original Problem to Solving a Modular Equation

Suppose we are given the t least significant bits (LSB) of the secret $\varepsilon q i_q \bmod N$, which is denoted by k. The latter value can be rewritten as follows:

$$\varepsilon q i_q \equiv 2^t x_0 + k \bmod N \ , \tag{6.8}$$

where t and k are known values, and x_0 is the $\lceil \log_2(N) - t \rceil$-bit unknown integer that is to be recovered. In the following Lemma, we provide a modular polynomial equation $P_\varepsilon(x)$ which admits x_0 as an integer root, *i.e.* such that $P_\varepsilon(x_0) \equiv 0 \bmod N$.

1. The execution time given here and in Section 6.4.2 have been obtained on a 32-bit CPU @3.2GHz

Lemma 52. *The unknown secret part x_0 is solution of the polynomial $P_\varepsilon(x)$:*

$$P_\varepsilon(x) = x^2 + c\ (2^{t+1}k - 2^t\varepsilon)\ x + c\ (k^2 - k\varepsilon) \equiv 0 \bmod N\ , \tag{6.9}$$

where $c = (2^{2t})^{-1} \bmod N$, k, t, N are known, and ε is the induced fault.

Proof. The Bézout identity applied to our context yields that primes p and q interrelate with integers $i_p = p^{-1} \bmod q$ and $i_q = q^{-1} \bmod p$ by the following relation:

$$pi_p + qi_q \equiv 1 \bmod N\ . \tag{6.10}$$

Multiplying (6.10) by ε leads to the relation $\varepsilon pi_p + \varepsilon qi_q \equiv \varepsilon \bmod N$, or equivalently to $\varepsilon pi_p \equiv \varepsilon - \varepsilon qi_q \bmod N$. Therefore, replacing εqi_q using (6.8) allows us to deduce an equivalence for εpi_p:

$$\varepsilon pi_p \equiv \varepsilon - 2^t x_0 - k \bmod N\ . \tag{6.11}$$

We then multiply (6.8) by (6.11), to get the relation:

$$\varepsilon qi_q \cdot \varepsilon pi_p \equiv (2^t x_0 + k) \cdot (\varepsilon - 2^t x_0 - k) \bmod N\ .$$

Since $N = pq$, we deduce that $\varepsilon qi_q \cdot \varepsilon pi_p \equiv 0 \bmod N$, which gives the equation:

$$(2^t x_0 + k) \cdot (\varepsilon - 2^t x_0 - k) \bmod N\ . \tag{6.12}$$

Eventually, developing the right-hand side of (6.12), and multiplying it by $c = (2^{2t})^{-1} \bmod N$ leads to the obtention of the monic polynomial $P_\varepsilon(x)$.

□

The initial problem of retrieving the unknown part of $\varepsilon qi_q \bmod N$ is thereby altered in solving the modular polynomial equation (6.9). In the sequel, we deal with two possible cases regarding ε, whether it is known to the adversary or not.

Case 1: The fault ε is known to the adversary

This case corresponds to the *bit-flip* and *stuck-at* fault models (Section 1.2) since the message is known to the attacker and the fault location can be chosen. In both cases, since the fault ε is known, the problem is reduced to solving a univariate modular polynomial equation, *cf.* Relation (6.9). This problem is known to be hard. However, when the integer solution x is small, Coppersmith showed [Cop96a] that it can be retrieved using the well-known LLL algorithm. Accordingly, we induce the following proposition:

Proposition 53. *Given $N = pq$ and the low order $1/2\ \log_2(N)$ bits of $\varepsilon qi_q \bmod N$ and assuming ε is known, one can recover in time polynomial in $(\log_2(N), d)$ the factorization of N.*

Proof. From Coppersmith's Theorem [Cop97], we know that, given a monic polynomial $P(x)$ of degree d, modulo an integer N of unknown factorization, and an upper bound X on the desired solution x_0, one can find in polynomial time all integers x_0 such that

$$P(x_0) \equiv 0 \bmod N \qquad \text{and} \qquad |X| < N^{1/d} \ . \tag{6.13}$$

In our case we have $d = 2$, and since x_0 is a $\lceil \log_2(N) - t \rceil$-bit integer, we know that $|x_0| < X = 2^{\lceil \log_2(N) - t \rceil}$. Thus, the condition in (6.13) becomes $2^{\lceil \log_2(N) - t \rceil} < N^{1/2}$, *i.e.*

$$t > \frac{1}{2} \log_2(N) \ . \tag{6.14}$$

Therefore, knowing at least half part of the secret $\varepsilon q i_q \bmod N$ allows to recover the whole secret. As previously done, computing $gcd(\varepsilon q i_q \bmod N, N)$ provides the factorization of N.

□

Note that the method is deterministic, and as will be seen further (Table 6.1), it is reasonably fast.

Case 2: The fault ε is unknown to the adversary

This case is met in the *unknown constant* fault model (see Chapter 1.2). In such a case, one can consider the polynomial $P_\varepsilon(x)$ as a bivariate modular polynomial equation with unknown values x and ε. This specific scheme has also been studied by Coppersmith and includes an additional difficulty of algebraic dependency of vectors which induces the heuristic characteristic of the method [Cop96b]. As depicted in Section 6.4.2, in our experiments nearly 100% of the tests verified the favorable property of independency. Accordingly, in this vast majority of cases, the following proposition holds:

Proposition 54. *Under an hypothesis of independency (see discussion above), given $N = pq$ and the low order $1/2 \ \log_2(N) + s$ bits of $\varepsilon q i_q \bmod N$, where s denotes the bitsize of ε, and assuming ε is unknown, one can recover in time polynomial in $(\log_2(N), d)$ the factorization of N.*

Proof. Coppersmith's Theorem for the bivariate modular case [Cop96b] notifies that given a polynomial $P(x, \varepsilon)$ of total degree d, modulo an integer N of unknown factorization, and upper bounds X and E on the desired solutions x_0, ε_0, it may be possible (heuristic) to find in polynomial time all integer couples (x_0, ε_0) such that

$$P(x_0, \varepsilon_0) \equiv 0 \bmod N \qquad \text{and} \qquad |X \cdot E| < N^{1/d} \ . \tag{6.15}$$

In our case, we have $d = 2$ and $E = 2^s$. The integer x_0 is $\lceil \log_2(N) - t \rceil$-bit long, therefore we have $X = 2^{\lceil \log_2(N) - t \rceil}$. Thus, the condition in (6.15) becomes $2^{\lceil \log_2(N) - t \rceil} \cdot 2^s < N^{1/2}$, *i.e.*

$$t > \frac{1}{2} \log_2(N) + s \ . \tag{6.16}$$

This means that knowing s more bits of the secret $\varepsilon qi_q \bmod N$ than before, would allow the recovering of the whole secret.

□

Remark 55. *The bound of success in Proposition 54 can actually be slightly improved using results of [BM05a]. Indeed, Coppersmith's bound applies to polynomials whose monomials shape is rectangular, while in our case the monomial ε^2 does not appear in $P(x, \varepsilon)$ which corresponds to what they called an extended rectangle in [BM05a]. For the sake of simplicity, we only mentioned Coppersmith's bound since practical results are similar.*

6.4.2 Results From Our Implementation

We have implemented this lattice-based improvement using Magma Software [BCP97], with N a 1024-bit integer *i.e.* 128 bytes long, in the cases where ε is an 8-bit known value (for Case 1) and a 32-bit unknown value (for Case 2). We chose Howgrave-Graham's method [HG97] for the univariate case, and its generalization by Jochemsz *et al.* [JM06] for the bivariate case since both have the same bound of success as Coppersmith's method (sometimes even better for [JM06]) and they are easier to implement. As we know, the theoretical bound given in Coppersmith's method is only asymptotic [Cop97]. Thus, we report in Table 6.1 (for Case 1) and in Table 6.2 (for Case 2) the size t (in bytes) of the secret $\varepsilon qi_q \bmod N$ that is known to the attacker before applying Coppersmith's method, the lattice dimension used to solve Relation (6.9) and finally the timings of our attack. We emphasize that the timings are taken from the original publication of our work at PKC'13 [BBD+13]. But for Case 1, the Rounding-LLL improvement proposed in [BCF+14] and described in Chapter 4 has been implemented (the Rounding method is less relevant when applied to the polynomial equation used in Case 2). Thus, an additional row is provided in Table 6.1 and represents the new timing using the Rounding method [BCF+14]. Naturally in the sequel, for Case 1 we only consider the timings provided by the use of the Rounding method (the speed up roughly ranges from 5 to 15, depending on the dimension of the lattice: the larger the dimension, the higher the speed-up).

Table 6.1: Size t required (in bytes) for the method to work and running time of the LLL-reduction (Magma V2.19-5), as a function of the lattice dimension in Case 1 (ε known, being an 8-bit integer).

Size t required (bytes)		69	68	67	**66**	65	64
Dimension		15	17	23	**37**	73	N/A
Timing (seconds)	**Original Method [PKC13]**	0.29	0.52	2.63	**34.25**	2588	N/A
	Rounding Method	0.05	0.1	0.4	**3.5**	170	N/A

Table 6.2: Size t required (in bytes) for the method to work and running time of the LLL-reduction (Magma V2.19-5), as a function of the lattice dimension in Case 2 (ε unknown, being a 32-bit integer).

Size t required (bytes)	74	73	**72**	71	70	69
Dimension	35	51	**70**	117	201	N/A
Timing (seconds) [PKC13]	1.17	5.88	**30.22**	606	12071	N/A

As depicted in Table 6.1, and combining these results with the experiments of Section 6.3, the best trade-off is to perform a CPA on the 66 first bytes, taking $66 \times 1m05s = 1h11m30s$, and to retrieve the 62 remaining bytes using lattices in $3.5s$, bringing the total time up to 1 hour 12 minutes, instead of the previous 2 hours 20 minutes.
In order to illustrate Case 2, we have chosen to rather show our results for ε being a 32-bit value, since when ε is 8-bit long, we obtained slightly better results by considering the 255 possible values of the variable ε together with their corresponding polynomials $P_\varepsilon(x)$, and by running the method on each of the polynomials until finding the solution x_0 that allows to factorize N. This indeed leads to a best trade-off of 68 bytes required from the CPA and the 60 remaining bytes computed with lattices by performing 255 times the LLL algorithm in the worst case, for a total of $68 \times 1m05s + 255 \times 0.1s$, *i.e.* 1 hour 14 minutes instead of 2 hours 20 minutes. Besides, this exhaustive search can be performed in parallel and it also has the advantage to be deterministic.
However, when ε is 32-bit long, an exhaustive search becomes impractical and, as depicted in Table 6.2, the best trade-off would be to perform a CPA on 72 bytes and to compute the 56 remaining bytes with lattices (even if heuristic, it worked in nearly 100% of the tests in practice), resulting in a total of $72 \times 1m05s + 30.22s$, *i.e.* 1 hour 19 minutes instead of the previous 2 hours 20 minutes.

6.5 Countermeasures

In this section, we describe different candidate countermeasures to protect an implementation against the CA presented in Section 6.2.

6.5.1 Blind Before Splitting

Our first proposition consists in avoiding the possibility to inject the same fault during several signature computations. To do so, we deport the blinding of the input message m before executing the two exponentiations modulo p and q:

$$m' = m + k_0 N \bmod k_1 N \ , \tag{6.17}$$

with k_0 and k_1 two n-bit random values generated at each algorithm execution (n being typically 64). Hence $S'_p = m'^{d_p} \bmod k_2 p$ and $S'_q = m'^{d_q} \bmod k_3 q$.

This countermeasure prevents an attacker from injecting always the same error during the signature computation. Indeed if the fault is injected on m at the very beginning of one exponentiation, then the corresponding error cannot be fixed due to the blinding injected by Rel. (6.17).

Moreover, if the fault is injected when the message m is manipulated during (6.17), then the error ε impacts the computation of both S'_p and S'_q, leading to seemingly unworkable faulty outputs.

Such a countermeasure induces a small overhead in terms of memory space since m' must be kept in memory during the first exponentiation but the execution time remains the same.

6.5.2 Verification Blinding

Our second countermeasure aims at annihilating the second hypothesis of our attack: a predictive variable is manipulated in plain during the verification. To do so, we inject a $\lceil \log_2(N) \rceil$-bit random r before performing the final reduction with N, *cf.* Rel. (6.18). Therefore, each and every variable manipulated during the verification is blinded.

$$((\widetilde{S}^e + r - m) \bmod k_1 N) \bmod N \stackrel{?}{=} r \ . \tag{6.18}$$

One may note that the final comparison should be performed securely with regards to the attack described in [LRT12] since information on $\varepsilon q i_q$ could leak if such a comparison was performed through a substraction.

The cost of such a countermeasure is negligible since it mainly consists in generating a $\lceil \log_2(N) \rceil$-bit random variable.

6.6 Conclusion

This chapter introduces a new Combined Attack on CRT-RSA. Even if a secure implementation does not return the faulty signature when the computation is disturbed, we show how to combine FI with SCA during the verification process to obtain information on the faulty signature. Such information allows us to factorize the public modulus and thus to recover the whole private key. Therefore, the main consequence of this result is that fault injection countermeasures must also be designed to resist SCA and vice versa. Indeed, stacking several countermeasures does not provide global security despite addressing each and every attack separately.

We also show that Coppersmith's methods for finding small solutions to univariate and bivariate modular polynomial equations can be used to significantly reduce the complexity of the attack. In particular, it highlights that lattice-based techniques can be very useful and complementary to physical attacks.

Bibliography

[AFMV07] Frédéric Amiel, Benoît Feix, Louis Marcel, and Karine Villegas. Passive and Active Combined Attacks – Combining Fault Attacks and Side Channel Analysis –. In L. Breveglieri, S. Gueron, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography – FDTC 2007*, pages 92–99. IEEE Computer Society, 2007.

[AFV07] Frédéric Amiel, Benoît Feix, and Karine Villegas. Power Analysis for Secret Recovering and Reverse Engineering of Public Key Algorithms. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *Selected Areas in Cryptography – SAC 2007*, LNCS, pages 110–125. Springer, 2007.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems. *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, STOC '96*, pages 99–108, 1996.

[ANS98] ANSI X9.31. *Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (rDSA)*. American National Standards Institute, September 1998.

[Bau08] Aurélie Bauer. *Vers une généralisation rigoureuse des méthodes de Coppersmith pour la recherche de petites racines de polynômes*. PhD thesis, Université de Versailles Saint-Quentin-en-Yvelines, 2008.

[BBD+13] Guillaume Barbu, Alberto Battistello, Guillaume Dabosville, Christophe Giraud, Guénaël Renault, Soline Renner, and Rina Zeitoun. Combined attack on RSA-CRT: why public verification must not be public? In *Public Key Cryptography – Proc. PKC '13*, volume 7778 of *Lecture Notes in Computer Science*, pages 198–215. Springer, 2013.

[BCF+14] Jingguo Bi, Jean-Sébastien Coron, Jean-Charles Faugère, Phong Q. Nguyen, Guénaël Renault, and Rina Zeitoun. Rounding and chaining LLL: Finding faster small roots of univariate polynomial congruences. In *Public Key Cryptography – Proc. PKC '14*, volume 8383 of *Lecture Notes in Computer Science*, pages 185–202. Springer, 2014.

[BCP97] Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997. Computational algebra and number theory (London, 1993).

[BD99] Dan Boneh and Glenn Durfee. Cryptanalysis of RSA with private key d less than $N^0.292$. In J. Stern, editor, *Advances in Cryptology – EUROCRYPT '99*, volume 1592 of *LNCS*. Springer, 1999.

[BD00] Dan Boneh and Glenn Durfee. Cryptanalysis of RSA with private key d less than $N^{0.292}$. *IEEE Transactions on Information Theory*, 46(4):1339, 2000.

[BDF98] Dan Boneh, Glenn Durfee, and Yair Frankel. An attack on RSA given a small fraction of the private key bits. In K. Ohta and P. Dinggyi, editors, *Advances in Cryptology – ASIACRYPT '98*, volume 1514 of *LNCS*, pages 25–34. Springer, 1998.

[BDHG99] Dan Boneh, Glenn Durfee, and Nick Howgrave-Graham. Factoring $n = p^r q$ for large r. In *Advances in Cryptology - Proc. CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 326–337. Springer, 1999.

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In W. Fumy, editor, *Advances in Cryptology – EUROCRYPT '97*, volume 1233 of *LNCS*, pages 37–51. Springer, 1997.

[Bel96] Bellcore. New Threat Model Breaks Crypto Codes. Press Release, September 1996.

[BM05a] Johannes Blömer and Alexander May. A Tool Kit for Finding Small Roots of Bivariate Polynomials over the Integers. In R. Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 251–267. Springer, 2005.

[BM05b] Johannes Blömer and Alexander May. A tool kit for finding small roots of bivariate polynomials over the integers. In *Advances in Cryptology - Proc. EUROCRYPT '05*, volume 3494 of *Lecture Notes in Computer Science*, pages 251–267. Springer, 2005.

[BM06] Daniel Bleichenbacher and Alexander May. New attacks on RSA with Small Secret CRT-Exponents. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography – PKC 2006*, volume 3958 of *LNCS*. Springer, 2006.

[BOS03] Johannes Blömer, Martin Otto, and Jean-Pierre Seifert. A New RSA-CRT Algorithm Secure against Bellcore Attacks. In S. Jajodia, V. Atluri, and T. Jaeger, editors, *ACM Conference on Computer and Communications Security – CCS'03*, pages 311–320. ACM Press, 2003.

[Buc94] Johannes Buchmann. Reducing lattice bases by means of approximations. In *Algorithmic Number Theory – Proc. ANTS-I*, volume 877 of *Lecture Notes in Computer Science*, pages 160–168. Springer, 1994.

[Cas] John W.S. Cassels. *An introduction to the geometry of numbers*, volume 2. Kluwer Academic Publishers.

[CFG+11] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. Square Always Exponentiation. In Daniel J. Bernstein

and Sanjit Chatterjee, editors, *INDOCRYPT*, volume 7107 of *LNCS*, pages 40–57. Springer, 2011.

[CH11a] Henry Cohn and Nadia Heninger. Approximate common divisors via lattices. *IACR Cryptology ePrint Archive*, 2011:437, 2011.

[CH11b] Henry Cohn and Nadia Heninger. Ideal forms of Coppersmith's theorem and Guruswami-Sudan list decoding. *ICS*, pages 298–308, 2011.

[CJK+09] Jean-Sébastien Coron, Antoine Joux, Ilya Kizhvatov, David Naccache, and Pascal Paillier. Fault Attacks on RSA Signatures with Partially Unknown Messages. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems – CHES 2009*, volume 5747 of *LNCS*, pages 444–456. Springer, 2009.

[CLO07] David Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms: an Introduction to Computational Algebraic Geometry and Commutative Algebra*, volume 10. Springer, 2007.

[CNS99] Christophe Coupé, Phong Q. Nguyen, and Jacques Stern. The effectiveness of lattice attacks against low-exponent RSA. In *Public Key Cryptography – Proc. PKC '99*, volume 1560 of *Lecture Notes in Computer Science*, pages 204–218. Springer, 1999.

[CNT10] Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. Fault Attacks against EMV Signatures. In Josef Pieprzyk, editor, *Topics in Cryptology – CT-RSA 2010*, volume 5985 of *Lecture Notes in Computer Science*, pages 208–220. Springer, 2010.

[Coh93] Henri Cohen. *A course in computational algebraic number theory*, volume 138 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin, 1993.

[Coh95] Henri Cohen. *A Course in Computational Algebraic Number Theory*. Graduate Texts in Mathematics. Springer, 2nd edition, 1995.

[Cop96a] Don Coppersmith. Finding a small root of a bivariate integer equation; factoring with high bits known. In *Advances in Cryptology - Proc. EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 178–189. Springer, 1996.

[Cop96b] Don Coppersmith. Finding a small root of a univariate modular equation. In *Advances in Cryptology - Proc. EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 155–165. Springer, 1996.

[Cop97] Don Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *J. Cryptology*, 10(4):233–260, 1997. Journal version of [Cop96b, Cop96a].

[Cor07] Jean-Sébastien Coron. Finding small roots of bivariate integer polynomial equations: A direct approach. In *Advances in Cryptology – Proc. CRYPTO '07*, volume 4622 of *Lecture Notes in Computer Science*, pages 379–394. Springer, 2007.

[CQ82] C. Couvreur and Jean-Jacques Quisquater. Fast Decipherment Algorithm for RSA Public-Key Cryptosystem. *Electronics Letters*, 18(21):905–907, 1982.

[DGRS09] Emmanuelle Dottax, Christophe Giraud, Matthieu Rivain, and Yannick Sierra. On Second-Order Fault Analysis Resistance for CRT-RSA Implementations. In Olivier Markowitch, Angelos Bilas, Jaap-Henk Hoepman, Chris J. Mitchell, and Jean-Jacques Quisquater, editors, *Information Security Theory and Practices – WISTP 2009*, volume 5746 of *LNCS*, pages 68–83. Springer, 2009.

[DV94] Hervé Daudé and Brigitte Vallée. An upper bound on the average number of iterations of the LLL algorithm. *Theor. Comput. Sci.*, 123(1):95–115, 1994.

[DV11] Vincent Dupaquis and Alexandre Venelli. Redundant Modular Reduction Algorithms. In Prouff [Pro11], pages 102–114.

[EBCO04] Éric Brier, Christophe Clavier, and Francis Olivier. Correlation Power Analysis with a Leakage Model. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *LNCS*, pages 16–29. Springer, 2004.

[EBNNT11] Éric Brier, David Naccache, Phong Nguyen, and Mehdi Tibouchi. Modulus Fault Attack against RSA-CRT Signatures. In Preneel and Takagi [PT11], pages 192–206.

[FGV11] Jungfeng Fang, Benedikt Gierlichs, and F. Vercauteren. To Infinity and Beyond: Combined Attack on ECC Using Points of Low Order. In Preneel and Takagi [PT11], pages 143–159.

[FIP13] FIPS PUB 186-4. *Digital Signature Standard*. National Institute of Standards and Technology, 2013.

[FOM91] Atsushi Fujioka, Tatsuaki Okamoto, and Shoji Miyaguchi. Esign: an efficient digital signature implementation for smartcards. *Eurocrypt*, pages 446–457, 1991.

[GAKK93] S. K. Godunov, A. G. Antonov, O. P. Kiriljuk, and V. I. Kostin. *Guaranteed accuracy in numerical linear algebra*, volume 252 of *Mathematics and its Applications*. Kluwer Academic Publishers Group, Dordrecht, 1993. Translated and revised from the 1988 Russian original.

[Gar59] Harvey L. Garner. The Residue Number System. *IRE Transactions on Electronic Computers*, 8(6):140–147, June 1959.

[Gir06] Christophe Giraud. An RSA Implementation Resistant to Fault Attacks and to Simple Power Analysis. *IEEE Transactions on Computers*, 55(9):1116–1120, September 2006.

[GT04] Christophe Giraud and Hugues Thiebeauld. A Survey on Fault Attacks. In J.-J. Quisquater, P. Paradinas, Y. Deswarte, and A.A. El Kalam, editors,

Smart Card Research and Advanced Applications VI – CARDIS 2004, pages 159–176. Kluwer Academic Publishers, 2004.

[GVL13] Gene H. Golub and Charles F. Van Loan. *Matrix computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD, fourth edition, 2013.

[Hås85] Johan Håstad. On using RSA with low exponent in a public key network. In H.C. Wiliams, editor, *Advances in Cryptology – CRYPTO '85*, volume 218 of *LNCS*, pages 403–408. Springer, 1985.

[HG97] Nick Howgrave-Graham. Finding small roots of univariate modular equations revisited. In *Cryptography and Coding – Proc. IMA '97*, volume 1355 of *Lecture Notes in Computer Science*, pages 131–142. Springer, 1997.

[HG01] Nick Howgrave-Graham. Approximate integer common divisors. In *Proc. CaLC '01*, volume 2146 of *Lecture Notes in Computer Science*, pages 51–66. Springer, 2001.

[HPS11] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. *Proceedings CRYPTO '11*, 6841:447–464, 2011.

[JM06] Ellen Jochemsz and Alexander May. A strategy for finding roots of multivariate polynomials with new applications in attacking rsa variants. In *ASIACRYPT'06*, pages 267–282, 2006.

[JY02] Marc Joye and Sung-Ming Yen. The Montgomery Powering Ladder. In B.S. Kaliski Jr., Ç.K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *LNCS*, pages 291–302. Springer, 2002.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In M.J. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *LNCS*, pages 388–397. Springer, 1999.

[Knu71] Donald Knuth. The analysis of algorithms. *Actes du Congrès International des Mathématiciens (Nice, 1970)*, 3:269–274, 1971.

[Koc96] Paul Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In N. Koblitz, editor, *Advances in Cryptology – CRYPTO '96*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996.

[Len87] Hendrik W. Lenstra. Factoring integers with elliptic curves. *Ann. Math.*, 126:649–673, 1987.

[Len96] Arjen K. Lenstra. Memo on RSA Signature Generation in the Presence of Faults. Manuscript, 1996.

[LKYL00] Seongan Lim, Seungjoo Kim, Ikkwon Yie, and Hongsub Lee. A generalized takagi-cryptosystem with a modulus of the form $p^r q^s$. *Indocrypt*, 1977:283–294, 2000.

[LLL82] Arjen K. Lenstra, Hendrik W. Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Ann.*, 261:513–534, 1982.

[LRT12] Victor Lomne, Thomas Roche, and Adrian Thillard. On the Need of Randomness in Fault Attack Countermeasures – Application to AES. In Guido Bertoni and Benedikt Gierlichs, editors, *Fault Diagnosis and Tolerance in Cryptography – FDTC 2012*, pages 85–94. IEEE Computer Society, 2012.

[Mat00] Yuri Matiyasevich. On Hilbert's Tenth Problem. *Pacific Institute for the Mathematical Sciences Distinguished Lecturer Series*, 2000.

[May10] Alexander May. Using LLL-reduction for solving RSA and factorization problems: A survey. 2010. In [NV10].

[Min12] Hermann Minkowski. *Geometrie der Zahlen.* Teubner Verlag, 1912.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks – Revealing the Secrets of Smartcards.* Springer, 2007.

[Ngu09] Phong Q. Nguyen. Public-key cryptanalysis. In I. Luengo, editor, *Recent Trends in Cryptography*, volume 477 of *Contemporary Mathematics.* AMS–RSME, 2009.

[NJD11] Hossein Najafi, M.E.D. Jafari, and Mohamed-Oussama Damen. On adaptive lattice reduction over correlated fading channels. *Communications, IEEE Transactions on*, 59(5):1224–1227, 2011.

[NS06] Phong Q. Nguyen and Damien Stehlé. LLL on the average. In *Algorithmic Number Theory – Proc. ANTS*, LNCS, pages 238–256. Springer, 2006.

[NS09] Phong Q. Nguyen and Damien Stehlé. An LLL algorithm with quadratic complexity. *SIAM J. of Computing*, 39(3):874–903, 2009.

[NSV11] Andrew Novocin, Damien Stehlé, and Gilles Villard. An LLL-reduction algorithm with quasi-linear time complexity: extended abstract. In *Proc. STOC '11*, pages 403–412. ACM, 2011.

[NV10] Phong Q. Nguyen and Brigitte Vallée, editors. *The LLL Algorithm: Survey and Applications.* Information Security and Cryptography. Springer, 2010.

[OU98] Tatsuaki Okamoto and Shigenori Uchiyama. A new public key cryptosystem as secure as factoring. *Eurocrypt*, pages 310–318, 1998.

[Pol74] John M. Pollard. Theorems on factorization and primality testing. *Mathematical Proceedings of the Cambridge Philosophical Society*, 76:521–528, 1974.

[Pro11] Emmanuel Prouff, editor. *Smart Card Research and Advanced Applications, 10th International Conference – CARDIS 2011*, LNCS. Springer, 2011.

[PT11] Bart Preneel and Tsuyoshi Takagi, editors. *Cryptographic Hardware and Embedded Systems – CHES 2011*, volume 6917 of *LNCS.* Springer, 2011.

[QS00] Jean-Jacques Quisquater and David Samyde. A New Tool for Non-intrusive Analysis of Smart Cards Based on Electro-magnetic Emissions, the SEMA and DEMA Methods. Presented at Eurocrypt 2000 Rump Session, 2000.

[Rit10] Maike Ritzenhofen. *On Efficiently Calculating Small Solutions of Systems of Polynomial Equations*. PhD thesis, 2010.

[Riv09] Matthieu Rivain. Securing RSA against Fault Analysis by Double Addition Chain Exponentiation. In Marc Fischlin, editor, *Topics in Cryptology – CT-RSA 2009*, volume 5473 of *LNCS*, pages 459–480. Springer, 2009.

[RLK11] Thomas Roche, Victor Lomné, and Karim Khalfallah. Combined Fault and Side-Channel Attack on Protected Implementations of AES. In Prouff [Pro11], pages 152–169.

[RM07] Bruno Robisson and Pascal Manet. Differential Behavioral Analysis. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007*, volume 4727 of *LNCS*, pages 413–426. Springer, 2007.

[RSA78] Ron Rivest, Adi Shamir, and Leonard Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[Sch82] Arnold Schönhage. *The fundamental theorem of algebra in terms of computational complexity - preliminary report*. Universität Tübingen, 1982.

[Sho02] Victor Shoup. OAEP reconsidered. *J. Cryptology*, 15(4):223–249, 2002.

[SKKO06] Bagus Santoso, Noboru Kunihiro, Naoki Kanayama, and Kazuo Ohta. Factorization of square-free integers with high bits known. *Progress in Cryptology - VIETCRYPT 2006*, 4341:115–130, 2006.

[SMSV14] Saruchi, Ivan Morel, Damien Stehlé, and Gilles Villard. LLL reducing with the most significant bits. *Proceedings ISSAC*, ACM Press, 2014.

[Tak98] Tsuyoshi Takagi. Fast rsa-type cryptosystem modulo $p^k q$. *Crypto*, pages 318–326, 1998.

[Vig08] David Vigilant. RSA with CRT: A New Cost-Effective Solution to Thwart Fault Attacks. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2008*, volume 5154 of *LNCS*, pages 130–145. Springer, 2008.

[VT98] D. Viswanath and L. N. Trefethen. Condition numbers of random triangular matrices. *SIAM J. Matrix Anal. Appl.*, 19(2):564–581 (electronic), 1998.

[Wie90] Michael J. Wiener. Cryptanalysis of short RSA secret exponents. *IEEE Transaction on Information Theory*, 36(3):553–558, May 1990.

[Wil82] Hugh C. Williams. A $p + 1$ method for factoring. *Mathematics of Computation*, 39:225–234, 1982.

[WvWM11] Marc Witteman, Jasper van Woudenberg, and Federico Menarini. Defeating RSA Multiply-Always and Message Blinding Countermeasures. In Aggelos Kiayias, editor, *Topics in Cryptology – CT-RSA 2011*, volume 6558 of *LNCS*, pages 77–88. Springer, 2011.