Algorithmic of curves in the context of bilinear and post-quantum cryptography

Simon Masson

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

THALES

Algorithmique des courbes destinées au contexte de la cryptographie bilinéaire et post-quantique

THÈSE

présentée et soutenue publiquement le 4 décembre 2020

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Simon Masson

Composition du jury

Présidente :	Monique Teillaud	Directrice de recherche, INRIA Nancy
Rapporteurs :	Andreas Enge	Directeur de recherche, INRIA Bordeaux
	Francisco Rodríguez-Henríquez	Directeur de recherche, CINVESTAV Mexico
Examinateur :	Emmanuel Fouotsa	Chargé de cours, Université de Bamenda
Invité :	Olivier Bernard	Ingénieur cryptologue, Thales
Directeurs de thèse :	Emmanuel Thomé	Directeur de recherche, INRIA Nancy
	Aurore Guillevic	Chargée de recherche, INRIA Nancy

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Remerciements

Ce document n'aurait pu voir le jour sans l'aide précieuse de mes deux directeurs de thèse. Merci Emmanuel et Aurore pour votre expertise et votre disponibilité durant ces trois années.

Je suis aussi très reconaissant envers Andreas Enge et Francisco Rodrígues–Henríquez pour leurs rapports minutieux. Merci également à Emmanuel Fouotsa et Monique Teillaud pour avoir accepté de faire partie de mon jury, et enfin à Olivier Bernard pour m'avoir fait découvrir le monde des courbes elliptiques ainsi que son "livre bleu".

Je remercie toute l'équipe du laboratoire Chiffre de Thales Gennevilliers. J'ai été accueilli dans une ambiance très chaleureuse, et j'ai vraiment apprécié mon passage dans le couloir avec vue sur la tour Eiffel. J'ai eu la chance de pouvoir me déplacer entre Gennevilliers et Nancy avec beaucoup de liberté. Merci à Anne pour l'aide lors des (rares) problèmes logistiques.

Malgré ma présence partielle au LORIA à Nancy, j'ai été très bien accueilli par toute l'équipe CARAMBA. Merci pour ces "cafés jeunes" très productifs, pour votre expertise sur des domaines très variés ($\mathbb{Q}_p$, Climb'Up, les choucas, etc.). Merci à Tom, Robin et Michel de m'avoir permis de me sentir chez moi à chaque passage rue du Portugal, et à Tarek (ceux qui le connaissent savent de quoi je parle).

Je remercie aussi tous les collègues avec qui j'ai eu l'occasion de travailler : Luca pour m'avoir fait découvrir les isogénies à Versailles et pour m'avoir fait confiance par la suite, Antonio pour nos nombreuses conversations sur Twitter, et enfin Christophe et Antonin pour m'avoir aidé à plusieurs reprises pour différents algorithmes.

Je tiens à remercier toutes les personnes que j'ai pu cotoyer durant ces dernières années. Merci à tous mes coéquipiers volleyeurs de Coutances, Boulogne-Billancourt et Sartrouville. Merci à mes amis rencontrés lors de mon passage aux universités de Nantes, Rennes et Versailles. Merci à tous les gens que j'ai eu la chance de cotoyer sur Paris. Vous êtes trop nombreux pour que je puisse tous vous citer ici.

Merci enfin à ma famille pour m'avoir toujours soutenu depuis le début. Merci de m'avoir inculqué les bases de la cryptographie à base d'isogénies dès mon plus jeune âge : preuve en est la peinture ci-dessous que j'ai faite à l'âge de 3 ans. On reconnaît bien-sûr le graphe de 2-isogénies de courbes supersingulières définies sur $\mathbb{F}_p$ (à isomorphisme sur $\mathbb{F}_p$ près) pour $p \equiv 1 \bmod 4$.

1996, un an avant [Cou06].

Contents

Résumé en français

Cette thèse étudie l'algorithmique d'objets mathématiques liés à des protocoles cryptographiques. Les courbes elliptiques sont actuellement très utilisées, autant pour des signatures électroniques (ECDSA) que pour des échanges de clés (ECDH). Nous étudions ici deux familles particulières de courbes. D'une part, nous nous intéressons à des courbes à couplages, dont l'arithmétique est étroitement liée aux calculs dans une extension d'un corps fini. Nous proposons dans le Chapitre 5 de nouvelles courbes résistantes aux dernières avancées sur les attaques de logarithme discret dans le cas des couplages. D'autre part, nous étudions les courbes supersingulières utilisées en cryptographie à base d'isogénies. Dans le chapitre 6, nous implémentons le calcul d'anneaux d'endomorphismes de ces courbes, étant donné la connaissance d'une isogénie. Enfin, dans le chapitre 7, nous présentons deux constructions de fonctions à délai vérifiables, basées sur des calculs de couplages et d'évaluations d'isogénies de grand degré friable. Ces dernières ne sont pas considérées comme résistantes aux ordinateurs quantiques, mais apportent plusieurs nouveautés par rapport aux constructions actuelles. Nous analysons leur sécurité et effectuons une comparaison entre toutes ces fonctions à un niveau de sécurité de 128 bits.

Chapitre 1. Corps finis

Nous présentons dans ce premier chapitre des exemples de groupes utilisés dans des applications à la cryptographie et qui sont centraux pour les applications des chapitres 4 et 5. Pour un corps fini $\mathbb{F}_{p^k}$ (avec p un nombre premier et k est un entier strictement positif), nous considérons un sous-groupe d'ordre premier r de $\mathbb{F}_{p^k}^*$, et nous notons g un de ses générateurs : $G = \langle g \rangle \subset \mathbb{F}_{p^k}^*$. Dans ce groupe, le calcul de logarithme discret est difficile : si $h = g^x$ avec $0 \leq x \leq r-1$, alors le meilleur algorithme pour retrouver x à partir de g et h est sous-exponentiel en la taille de r. À partir d'un groupe cyclique, Diffie et Hellman ont introduit en 1976 une solution pour partager un secret commun, qui est actuellement très utilisé en pratique: Alice (resp. Bob) choisit un entier secret a (resp. b) et calcule sa clé publique $g_A = g^a$ (resp. $g_B = g^b$). A partir de ces valeurs, Alice et Bob ont la clé $g^{ab} = g_A^b = g_B^a$ en commun.

Dans le cas des corps finis, une méthode basée sur un calcul d'indices permet d'obtenir des algorithmes avec une complexité sous-exponentielle. A partir d'une base de facteurs S composée de petits éléments de $\mathbb{F}_{p^k}^*$, l'algorithme se déroule en trois étapes : une collecte de relations, de l'algèbre linéaire grâce aux relations afin de déterminer des logarithmes discrets de certains éléments, et enfin le calcul du logarithme discret cible à partir des calculs précédents.

Les algorithmes de calcul d'indices que nous étudions ici sont basés sur des méthodes de crible dans différentes structures algébriques. Lorsque p est relativement petit, un logarithme discret peut être obtenu à l'aide d'algorithmes quasi-polynomiaux [BGJT14]. Pour des tailles de corps finis plus grandes, les meilleurs algorithmes ont une complexité sous-exponentielle $L_{p^k}(1/3, c + o(1))$, où $L_{p^k}(\alpha, c) = \exp(c(\log p^k)(\log\log p^k)^{1-\alpha})$. Dans la suite, nous considérons

des corps de moyenne et grande caractéristique, pour lesquelles l'algorithme le plus efficace est un crible algébrique, aussi appelé NFS pour Number Field Sieve). L'idée du crible algébrique est d'obtenir des relations comme dans le calcul d'indices, en passant par des morphismes dans deux corps deux nombres. Le choix des polynômes qui définissent ces corps permettent d'obtenir différentes complexités pour calculer des logarithmes discrets.

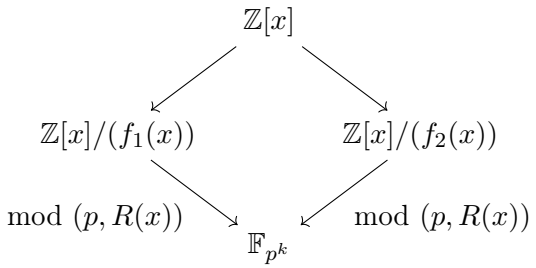


Diagramme commutatif pour NFS.

Le choix original des polynômes permettait de cribler d'une part dans un corps de nombres, et d'autre part dans $\mathbb{Z}$ (en choisissant $\deg f_2 = 1$). L'algorithme original permettait de calculer des logarithmes discrets dans un corps premier $\mathbb{F}_p$, correspondant à $\deg R = 1$, et sa complexité est $L_p(1/3, \sqrt[3]{64/9} + o(1))$. Plusieurs variantes ont ensuite permi de cribler dans des tours d'extensions de corps, et de réduire la constante de la complexité. Ainsi, cela permet d'estimer plus précisemment la complexité pratique de ces algorithmes pour certaines tailles de corps. Les complexités sont résumées dans le tableau suivant.

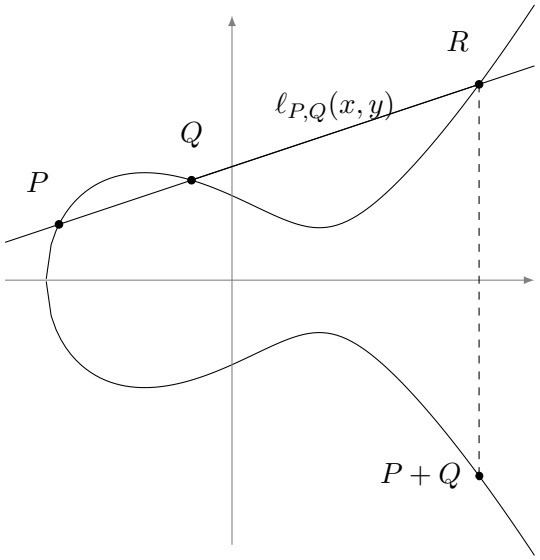
Variante de NFS	**Caracteristique**	**Complexité asymptotique** $L_{p^k}(1/3, c + o(1))$
Originale	Grande	$c = \sqrt[3]{64/9}$
(généralisée) Joux-Lercier	Grande	$c = \sqrt[3]{64/9}$
Speciale	Grande, spéciale	$c = \sqrt[3]{32/9}$
Conjuguée	Moyenne	$c = \sqrt[3]{96/9}$
(étendue) Tower	Moyenne	$c = \sqrt[3]{48/9}$
Spéciale Tower	Moyenne, spéciale	$c = \sqrt[3]{32/9}$

À partir des estimations du coût des variantes de NFS, nous obtenons des tailles de corps finis pour un niveau de sécurité donné. En particulier, nous nous focalisons sur le niveau de 128 bits : nous cherchons des tailles pour p et k tel que le meilleur algorithme de calcul de logarithme discret a un coût de $O(2^{128})$ opérations. De plus, nous nous intéressons à des tours d'extension particulières, qui seront utilisées dans les Chapitres 4, 5 et 7. Nous obtenons des estimations de temps de calcul de la multiplication dans $\mathbb{F}_{p^k}$ (notée $\mathbf{m}_k$) en calculant le nombre de multiplications d'entiers modulo p, puis en estimant le coût de l'arithmétique sur $\mathbb{F}_p$. De même, nous estimons le coût d'une élévation au carré $\mathbf{s}_k$, d'un Frobenius $\mathbf{f}_k$, et d'inversions $\mathbf{i}_k$ dans $\mathbb{F}_{p^k}$. Nous portons aussi une attention particulière aux carrés cyclotomiques $\mathbf{s}_k^{\text{cyclo}}$, calculs spécifiques à un sous-groupe d'ordre $\Phi_k(p)$ de $\mathbb{F}_{p^k}$.

k	1	2	3	5	6	7	8	12	16
$\mathbf{m}_k$	$\mathbf{m}$	$3\mathbf{m}$	$6\mathbf{m}$	$13\mathbf{m}$	$18\mathbf{m}$	$22\mathbf{m}$	$27\mathbf{m}$	$54\mathbf{m}$	$81\mathbf{m}$
$\mathbf{s}_k$	$\mathbf{m}$	$2\mathbf{m}$	$5\mathbf{m}$	$13\mathbf{m}$	$12\mathbf{m}$	$22\mathbf{m}$	$18\mathbf{m}$	$36\mathbf{m}$	$54\mathbf{m}$
$\mathbf{f}_k$	0	0	$2\mathbf{m}$	$4\mathbf{m}$	$4\mathbf{m}$	$6\mathbf{m}$	$6\mathbf{m}$	$10\mathbf{m}$	$14\mathbf{m}$
$\mathbf{s}_k^{\text{cyclo}}$					$6\mathbf{m}$		$12\mathbf{m}$	$18\mathbf{m}$	$36\mathbf{m}$
$\mathbf{i}_k - \mathbf{i}_1$	0	$4\mathbf{m}$	$12\mathbf{m}$	$48\mathbf{m}$	$34\mathbf{m}$	$104\mathbf{m}$	$44\mathbf{m}$	$94\mathbf{m}$	$134\mathbf{m}$
$\mathbf{i}_k$, with $\mathbf{i}_1 = 25\mathbf{m}$	$25\mathbf{m}$	$29\mathbf{m}$	$37\mathbf{m}$	$73\mathbf{m}$	$59\mathbf{m}$	$129\mathbf{m}$	$69\mathbf{m}$	$119\mathbf{m}$	$159\mathbf{m}$

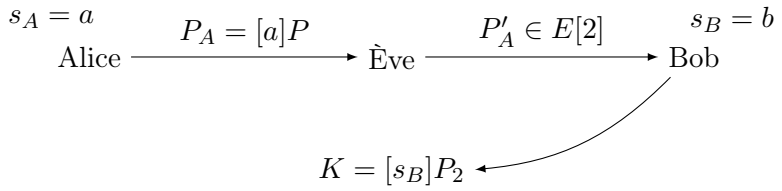
Chapitre 2. Courbes elliptiques

Les courbes elliptiques sont des courbes algébriques très répandues dans les applications cryptographiques. Elles ont une structure particulière qui permet de définir une loi de groupe. Géométriquement, cette loi peut être représentée à l'aide de lignes obliques et verticales :



Loi de groupe géométrique.

Le groupe des points d'une courbe elliptique est un groupe abélien, et les meilleurs algorithmes pour calculer des logarithmes discrets dans ce cas sont des algorithmes génériques. En particulier, les variantes de NFS ne s'appliquent pas et la complexité est exponentielle en la taille du groupe considéré. Cela permet d'obtenir des tailles de clés beaucoup plus petites que dans le cas de $\mathbb{F}_{p^k}^*$ décrit précédemment. A titre d'exemple, pour un niveau de sécurité de 128 bits, il est recommandé d'utiliser un corps premier (générique) d'au moins 3072 bits alors qu'un sous-groupe d'ordre premier de $E(\mathbb{F}_q)$ de taille $\approx 2^{256}$ est souvent suffisant. Dans certains cas, des attaques peuvent permettre de réduire la sécurité. Nous étudions ici les attaques basées sur les petits sous-groupes et utilisant des tordues de courbes elliptiques. L'attaque du milieu est un cas classique qui s'adapte bien dans le cas où $\#E(\mathbb{F}_q)$ est pair : Ève se fait passer pour Alice et modifie sa clé publique afin d'obtenir une partie du secret de Bob.



Attaque par intrusion au milieu quand $\#E(\mathbb{F}_q)$ est pair.

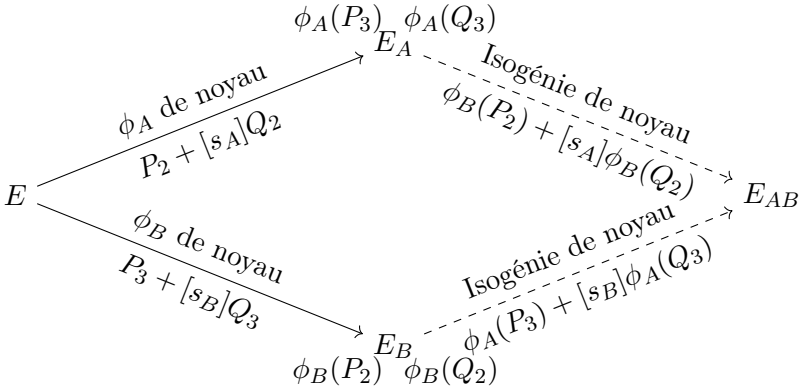
Nous présentons aussi dans ce chapitre différentes possiblités pour calculer la loi de groupe, suivant la structure de la courbe choisie. Nous apportons enfin des détails sur la structure des courbes étudiées. Il est par exemple possible d'obtenir des éléments de torsion grâce aux polynômes de division. Pour un entier ℓ premier à la caractéristique, la ℓ-torsion est isomorphe à $\mathbb{Z}/\ell\mathbb{Z} \times \mathbb{Z}/\ell\mathbb{Z}$. Il est fréquent d'appeler $\mathbb{G}_1$ et $\mathbb{G}_2$ deux sous-groupes d'ordre ℓ générant la ℓ-torsion complète. Ces deux groupes seront réutilisés pour les applications à base de couplages dans les chapitres qui suivent. Il est aussi possible de caractériser l'anneau d'endomorphismes d'une courbe elliptique. Cela permet de séparer les courbes en deux catégories : les courbes ordinaires (utilisées dans les Chapitres 4 et 5) pour lesquelles l'anneau d'endomorphismes est un ordre dans le corps quadratique imaginaire $\mathbb{Q}(\sqrt{t^2-4q})$ et les courbes super-singulières (utilisées dans les Chapitres 6 et 7) pour lesquelles l'anneau d'endomorphismes est un ordre maximal d'une algèbre de quaternions particulière.

Chapitre 3. Isogénies de courbes elliptiques

Les isogénies de courbes elliptiques apportent des applications cryptographiques qui se distinguent de la cryptographie basée sur les courbes elliptiques : ce n'est pas le problème du logarithme discret sur une courbe qui est utilisé. La cryptographie à base d'isogénies repose sur le fait qu'il est difficile de déterminer une isogénie entre deux courbes. Nous considérons ici seulement des isogénies cycliques (dont le noyau est un sous-groupe cyclique des points de la courbe).

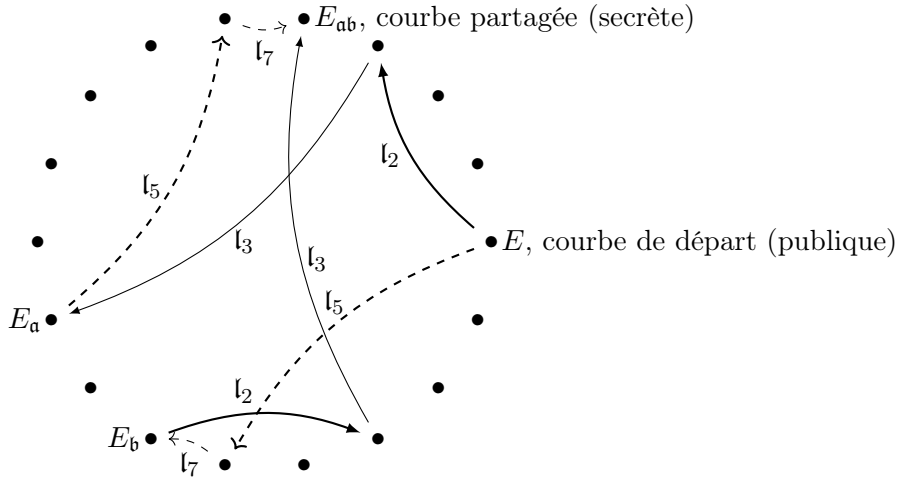
Nous rappelons d'abord la structure du graphe d'isogénies dans le cas des courbes ordinaires. Cela permet de définir une action de groupe et donne des applications intéressantes en cryptographie, avec notamment [DKS18].

Nous nous intéressons ensuite au cas des courbes supersingulières définies sur $\mathbb{F}_{p^2}$. Le graphe d'isomorphismes de courbes isogènes est beaucoup moins structuré et cela permet de définir des algorithmes résistants aux ordinateurs quantiques. L'échange de clé SIDH est décrit en Section 3.3.3. Nous notons ici $\langle P_2, Q_2\rangle$ et $\langle P_3, Q_3\rangle$ des bases de la $2-$ et $3-$torsion.



Échange de clé Diffie-Hellman en utilisant des isogénies de courbes super-singulières.

Le cas du graphe d'isogénies super-singulières définies sur $\mathbb{F}_p$ est à la frontière entre les deux précédents cas, puisqu'il utilise un graphe de classes d'isomorphismes particuliers de courbes (super-singulières) qui a une structure très proche du cas ordinaire. Une isogénie définie sur $\mathbb{F}_p$ peut être représentée par l'action d'un idéal du groupe de classes du corps quadratique correspondant à $\mathrm{End}_p(E)$. Cela permet des applications en cryptographie telles que [CLM+18], ou encore la signature [BKV19]. La figure ci-dessous donne un exemple d'échange de clé dans sa version décrite avec des idéaux du groupe de classes agissant sur l'ensemble des courbes super-singulières définies sur $\mathbb{F}_p$.

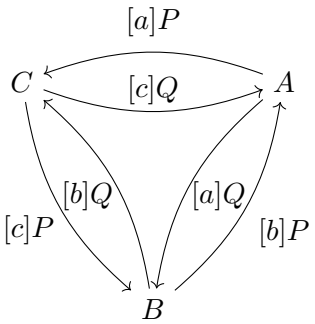


Nous nous intéressons aussi dans ce chapitre au coût de calcul d'isogénies. Les applications du Chapitre 7 utilisent le fait que les meilleurs algorithmes de calcul d'isogénies ont une complexité exponentielle en le degré de l'isogénie. Nous présentons plusieurs variantes provenant de [JD11] afin d'évaluer des isogénies.

Chapitre 4. Courbes à couplages

Un couplage est une application $e : G \times G' \to G''$ entre trois groupes G, G' et G'' d'exposant premier qui est non-dégénérée, bilinéaire et évaluable de manière efficace.

Les couplages ont été introduits avec des sous-groupes d'une courbe elliptique et d'un corps fini. Plus précisement, G et G' sont des sous-groupes de points d'une courbe elliptique et G'' est un groupe de racines r-ièmes de l'unité dans un corps fini $\mathbb{F}_{q^k}$. Avec ces notations, Alice, Bob et Charlie peuvent partager un secret commun en un tour en utilisant le protocole ci-dessous. La loi de groupe est en notation additive pour G et G', et en notation multiplicative pour G''. En utilisant la bilinéarité du couplage, Alice calcule $e([b]P, [c]Q)^a = e(P, Q)^{abc}$. De manière similaire, Bob et Charlie peuvent aussi calculer $e(P, Q)^{abc}$, le secret commun.



Échange de clé Diffie–Hellman à trois en un tour.

En pratique, les couplages manipulent des points d'ordre premier r d'une courbe elliptique E définie sur $\mathbb{F}_q$. Il est parfois nécessaire d'étendre la courbe E à $\mathbb{F}_{q^k}$, où k est le degré de plongement par rapport à r, c'est-à-dire le plus petit entier tel que toute la r-torsion est rationelle sur $\mathbb{F}_{q^k}$. A partir de cette extension $\mathbb{F}_{q^k}$ de $\mathbb{F}_q$, les constructions de couplages se basent sur une évaluation de fonction de Miller. Pour $Q \in E(\mathbb{F}_{q^k})$ et $n \in \mathbb{Z}$, la fonction de Miller $f_{n,Q}$ est la fonction rationnelle de $\mathbb{F}_{q^k}(E)$ telle que $\mathrm{div}(f_{n,Q}) = n(Q) - ([n]Q) - (n-1)(0_E)$. Des algorithmes permettent d'évaluer une fonction de Miller en un diviseur en temps linéaire en $\log_2(n)$. Dans la suite, nous notons pour D_Q (pour Q un point d'ordre r) un diviseur équivalent au diviseur $r(Q) - r(0_E)$. À partir de cette fonction, trois couplages sont fréquemment utilisés, avec des contraintes différentes. Les groupes d'exposant r utilisés sont $E[r]$, $E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k})$, ainsi que $\mathbb{F}^*_{q^k}/(\mathbb{F}^*_{q^k})^r$. En pratique, on peut souvent se passer de ces groupes abstraits et choisir des représentants de classes d'équivalence dans $\mathbb{G}_1$ et $\mathbb{G}_2$ pour les groupes de la courbe, et des racines r-ième de l'unité pour les représentants de $\mathbb{F}^*_{q^k}/(\mathbb{F}^*_{q^k})^r$. Dans la suite, nous considérons des points P et Q d'ordre r d'une courbe elliptique, et choisissons D_Q (resp. D_P) de sorte que son support soit disjoint de $\{P, 0_E\}$ (resp. $\{Q, 0_E\}$). Ainsi, cela permet de définir :

- Le couplage de Weil $e_r(P,Q) = f_{r,P}(D_Q)/f_{r,Q}(D_P)$ pour P et Q dans $E(\mathbb{F}_{q^k})[r]$.
- Le couplage de Tate réduit $t_r(P,Q) = f_{r,P}(D_Q)^{(q^k-1)/r}$ pour $P \in E(\mathbb{F}_{q^k})[r]$ et $Q \in E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k})$.
- Le couplage de ate réduit $A_r(Q,P) = f_{T,Q}(D_P)^{(q^k-1)/r}$ pour $(Q,P) \in \mathbb{G}_2 \times \mathbb{G}_1$, et T une racine k-ième de l'unité modulo r.

Les couplages manipulent des points définis sur $\mathbb{F}_{q^k}$ où k est le degré de plongement. Il faut donc choisir une courbe particulière de sorte que k ne soit pas trop grand et que les calculs soient raisonnables. Plusieurs méthodes de générations de courbes permettent d'obtenir des courbes à couplages. Nous présentons dans ce chapitre trois familles de courbes dont les paramètre (la caractéristique p, la trace t, l'ordre des sous-groupes r, etc.) sont paramétrisés par des polynômes.

- Les courbes Barreto-Naehrig (BN) sont paramétrées par les polynômes suivants :

$$\begin{aligned} r(x) &= 36x^4 + 36x^3 + 18x^2 + 6x + 1, \\ t(x) &= 6x^2 + 1, \\ p(x) &= 36x^4 + 36x^3 + 24x^2 + 6x + 1. \end{aligned}$$

  Elles ont un degré de plongement $k = 12$ et les automorphismes de la courbe permettent de manipuler exclusivement des élements de $\mathbb{F}_{p^2}$ (au lieu de $\mathbb{F}_{p^{12}}$). Ces courbes étaient

particulièrement intéressantes avant les récentes variantes de NFS puisqu'elles permettaient d'obtenir une courbe définie sur $\mathbb{F}_p$ et un corps fini $\mathbb{F}_{p^{12}}$ de taille minimale pour atteindre le niveau de sécurité de 128 bits. Depuis les variantes Spéciale, Tower et spécial Tower, il a fallu augmenter la taille de la caractéristique, et donc du corps finis $\mathbb{F}_{p^k}$.

- Les courbes Barreto-Lynn-Scott (BLS12) sont aussi définies pour un degré de plongement $k = 12$, et possèdent les mêmes propriétés que les courbes BN en terme d'automorphismes. Elles sont paramétrées par :

$$\begin{aligned} r(x) &= x^4 - x^2 + 1, \\ t(x) &= x + 1, \\ p(x) &= (x^6 - 2x^5 + 2x^3 + x + 1)/3. \end{aligned}$$

  Ces courbes étaient aussi optimales dans le même sens que les courbes BN, et nécessitent désormais une caractéristique plus grande pour obtenir des applications qui ne sont pas vulnérables aux variantes de NFS.

- Les courbes Kashisa-Schaefer-Scott (KSS16) ont un degré de plongement $k = 16$ et possèdent des automorphismes de degré 4 qui permettent de manipuler des points définis sur $\mathbb{F}_{p^4}$. Les entiers r, t et p de la courbe sont paramétrés par :

$$\begin{aligned} r(x) &= (x^8 + 48x^4 + 625)/61250, \\ t(x) &= (2x^5 + 41x + 35)/35, \\ p(x) &= (x^{10} + 2x^9 + 5x^8 + 48x^6 + 152x^5 + 240x^4 + 625x^2 + 2398x + 3125)/980. \end{aligned}$$

  Cette troisième famille est aussi vulnérable aux variantes de NFS.

Nous étudions le coût du couplage de ate réduit pour un niveau de sécurité de 128 bits. Pour les courbes BN, Barbulescu et Duquesne proposent dans [BD19] de choisir $x_0 = 2^{114} + 2^{101} - 2^{14} - 1$ et ainsi obtenir un niveau de sécurité suffisant pour le calcul de logarithmes discrets sur les deux sous-groupes $\mathbb{G}_1$ et $\mathbb{G}_2$ de la courbe, ainsi que sur le corps fini $\mathbb{F}_{p^{12}}$. De même, ils proposent de choisir $x_0 = -2^{77} + 2^{50} + 2^{33}$ pour les courbes BLS12 et $x_0 = 2^{35} - 2^{32} - 2^{18} + 2^8 + 1$ pour les courbes KSS16. Ces choix de paramètres sont en particulier liés aux estimations du coût de NFS étudié dans le chapitre 1. La définition du couplage de ate réduit est différente pour ces trois familles de courbes, mais se calcule en deux étapes : une boucle de Miller pour une exponentiation à la puissance $(p^k - 1)/r$. Nous résumons ici le coût en donnant le nombre de multiplications nécessaire pour chacune de ces deux étapes.

Curve	**Miller loop**	**Final exponentiation**	**Optimal ate**
BN	12005**m**	5742**m**	17747**m**
BLS12	7685**m**	6288**m**	13973**m**
KSS16	7573**m**	25521**m**	33094**m**

Chapitre 5. Nouvelles courbes à couplages

Dans ce chapitre, nous construisons de nouvelles courbes à couplages résistantes aux variantes de NFS. En particulier, nous générons des courbes pour lesquelles le corps fini n'est pas vulnérable aux attaques SNFS (en utilisant une caractéristique qui n'est pas vulnérable au caractère *spécial*). Notre méthode de génération est une variante de la méthode Cocks–Pinch. L'algorithme est le suivant :

Entrée. un degré de plongement k, un discriminant $-D$, une plage d'entiers $\{T_0, \ldots, T_{\max}\}$, une taille d'entier λ_r et une taille d'entier λ_p.
Sortie. Une courbe à couplage de degré de plongement k, de discriminant $-D$ avec $\lceil \log_2(p) \rceil = \lambda_p$ et $\lceil \log_2(r) \rceil = \lambda_r$.
Pour T entre T_0 et $T_{\max}$ **faire**
  Si $r = \Phi_k(T)$ n'est pas premier, **continuer**.
  Si $\lceil \log_2(r) \rceil \neq \lambda_r$ ou $-D$ n'est pas un carré modulo r, **continuer**.
  Pour i entre 1 et $k-1$ tel que $\gcd(i,k) = 1$ **faire**
    $t_0 = T^i + 1 \bmod r$; $y_0 = (t_0 - 2)/\sqrt{-D} \bmod r$.
    Soit $\pi_0 = \frac{t_0 + y_0\sqrt{-D}}{2}$.
    Choisir h_t et h_y tels que $\pi = \pi_0 + \frac{h_t + h_y\sqrt{-D}}{2} r$ est un entier algébrique, et $\lceil \log_2(\pi\bar{\pi}) \rceil = \lambda_p$.
    $t = t_0 + h_t r$; $y = y_0 + h_y r$; $p = \pi\bar{\pi} = (t^2 + Dy^2)/4$.
    Si p est un premier congru à 1 modulo 4 alors **retourner** $[p, r, T, t, y]$.

Nous utilisons cet algorithme afin de sélectionner des courbes qui possèdent de bonnes propriétés en terme de sécurité : logarithme discret difficile sur la courbe et sur le corps fini $\mathbb{F}_{p^k}$, sécurité par rapport aux attaques sur les petits sous-groupes et sur la tordue quadratique, etc. Nous obtenons des courbes de degré de plongement $k = 5$ à 8 pour un niveau de sécurité de 128 bits, et les comparons aux courbes BN, BLS12 et KSS16 avec des paramètres légèrement plus petits que ceux proposés par Barbulescu et Duquesne (présentés dans le chapitre 4). Nous comparons aussi nos courbes aux constructions de [CMR17] de degré de plongement $k = 1$. En estimant le coût d'une multiplication sur $\mathbb{F}_p$ pour différentes tailles de premiers, nous obtenons le coût d'un couplage optimal sur les différentes courbes.

Courbe	**Premier**	**Boucle de Miller** Estimation de temps	**Exponentiation** Estimation de temps	**Total**	Estimation de temps
$k = 5$	663-bit	14496**m** 2.6ms	9809**m** 1.8ms	24305**m**	4.4ms
$k = 6$	672-bit	4601**m** 0.8ms	3871**m** 0.7ms	8472**m**	1.5ms
$k = 7$	512-bit	18330**m** 1.9ms	13439**m** 1.4ms	31769**m**	3.4ms
$k = 8$	544-bit	4502**m** 0.6ms	7056**m** 0.9ms	11558**m**	1.5ms
BN	446-bit	11620**m** 1.0ms	5349**m** 0.5ms	16969**m**	1.4ms
BLS12	446-bit	7805**m** 0.7ms	7723**m** 0.7ms	15528**m**	1.3ms
BN	462-bit	12180**m** 1.3ms	5727**m** 0.6ms	17907**m**	1.9ms
BLS12	461-bit	7685**m** 0.8ms	6283**m** 0.7ms	13968**m**	1.5ms
KSS16	339-bit	7691**m** 0.5ms	18235**m** 1.2ms	25926**m**	1.7ms
$k = 1$	3072-bit	4651**m** 17.7ms	4100**m** 15.6ms	8751**m**	33.3ms

Chapitre 6. Endomorphismes de courbes super-singulières

L'anneau d'endomorphisme d'une courbe elliptique super-singulière est un ordre maximal d'une algèbre de quaternion de la forme $H_{-a,-b} := \mathbb{Q}\mathbf{1}+\mathbb{Q}\mathbf{i}+\mathbb{Q}\mathbf{j}+\mathbb{Q}\mathbf{k}$, avec $\mathbf{i}^2+a = 0$, $\mathbf{j}^2+b = 0$ et $\mathbf{k} = \mathbf{ij}$ pour des entiers a et b spécifiques. Etant donnée une courbe super-singulière aléatoire (c'est-à-dire ne provenant pas d'une courbe définie sur un corps de nombre avec un petit discriminant), le meilleur algorithme pour calculer l'anneau d'endomorphismes est exponentiel en la taille des paramètres de la courbe. Les constructions cryptographiques à base d'isogénies telles que SIDH ou CSIDH ont une sécurité qui repose sur ce problème.

Les récents travaux [KLPT14] et [GPS20] permettent de calculer l'anneau d'endomorphismes d'une courbe étant donnée une isogénie entre celle-ci et une courbe d'anneau d'endomorphismes connu. Nous proposons ici une implémentation de ces algorithmes pour une estimation pratique de leur complexité. Notre code est disponible à l'adresse

`https://gitlab.inria.fr/smasson/endomorphismsthroughisogenies`.

Chapitre 7. Fonctions à délai vérifiables

Dans ce dernier chapitre, nous proposons deux constructions de fonction à délai vérifiables (VDF). Par définition, une telle fonction se décompose en trois étapes :

1. **Init**$(\lambda, T) \to (\mathsf{ek}, \mathsf{vk})$: une procédure d'initialisation qui prend en entrée un paramètre de sécurité λ, un paramètre de délai T, et retourne des paramètres pubics (une clé d'évaluation ek et une clé de vérification vk).

2. **Éval**$(\mathsf{ek}, s) \to (a, \pi)$: une procédure pour évaluer la fonction pour une entrée s. La sortie produite est l'image a de s, ainsi qu'une preuve π (qui peut être vide) qui sera utilisée pour la vérification. Cette procédure est censée être évaluable en au moins T étapes, même avec plusieurs machines mises en parallèle.

3. **Vérif**$(\mathsf{vk}, s, a, \pi) \to \{$vrai,faux$\}$: une procédure qui permet de vérifier que a est bien la sortie de s (en utilisant la preuve π).

Des VDF ont été construites à partir d'exponentiations modulaires modulo un entier RSA, ou encore avec des multiplications d'idéaux dans le groupe de classes d'un corps quadratique imaginaire. Chacune des VDF a ses avantages et ses inconvénients, mais aucune VDF résistant aux ordinateurs quantiques n'a été construite. Nous proposons ici des constructions basées sur le calcul d'isogénies pour l'évaluation, et une vérification à l'aide de couplages. Nous proposons deux versions : une avec des isogénies définies sur $\mathbb{F}_p$, et une avec des isogénies définies sur $\mathbb{F}_{p^2}$.

Version sur $\mathbb{F}_p$	**Version sur $\mathbb{F}_{p^2}$**

Init(λ, T)

1. Choisir des premiers r, p avec des propriétés de sécurité par rapport au paramètre de sécurité λ;

2. Sélectionner une courbe super-singulière E définie sur $\mathbb{F}_p$;

3. Choisir une direction dans le graphe horizontal de ℓ-isogénies, et calculer l'isogénie cyclique $\phi : E \to E'$ de degré ℓ^T, et sa duale $\hat{\phi}$;	3. Calculer une marche aléatoire de T pas dans le $\mathbb{F}_{p^2}$-graphe de ℓ-isogénies, définissant une ℓ^T-isogénie cyclique $\phi : E \to E'$ et sa duale $\hat{\phi}$;

4. Choisr un générateur P de $\mathbb{G}_1 = \tau_2^{-1}(E^t(\mathbb{F}_p)[r])$, et calculer $\phi(P)$;

5. Retourner $(\mathsf{ek}, \mathsf{vk}) = \big(\hat{\phi}, (E, E', P, \phi(P))\big)$.

Éval$(\hat{\phi}, Q \in \mathbb{G}_2')$ calculer et retourner $\hat{\phi}(Q)$.	**Éval**$(\hat{\phi}, Q \in E'[r])$ calculer et retourner $(\mathrm{Tr} \circ \hat{\phi})(Q)$.
Vérif$(E, E', P, Q, \phi(P), \hat{\phi}(Q))$: vérifier que $\hat{\phi}(Q) \in \mathbb{G}_2 = E(\mathbb{F}_p)[r]$ et $e(P, \hat{\phi}(Q)) = e'(\phi(P), Q)$.	**Vérif**$(E, E', P, Q, \phi(P), (\mathrm{Tr} \circ \hat{\phi})(Q))$ vérifier que $(\mathrm{Tr} \circ \hat{\phi})(Q) \in E(\mathbb{F}_p)[r]$ et $e(P, (\mathrm{Tr} \circ \hat{\phi})(Q)) = e(\phi(P), Q)^2$.

Nous étudions la sécurité de ces deux constructions. Nos constructions reposent sur la difficulté de calculer des logarithmes discrets dans un corps fini $\mathbb{F}_{p^2}^*$. En ce sens, nos deux constructions ne sont pas résistantes aux ordinateurs quantiques et ce problème reste ouvert à ce jour. Nous définissons cependant une résistance partielle pour la VDF définie sur $\mathbb{F}_{p^2}$ dans le sens où un attaquant doit réutiliser son attaque quantique à chaque évaluation de la VDF. Ce n'est pas le cas des constructions dans un anneau RSA ou dans le groupe de classes d'un corps quadratique imaginaire (l'attaquant peut effectuer son attaque quantique dès lors que l'initialisation est effectuée). Notre VDF doit être initialisée avec une courbe dont l'anneau d'endomorphismes n'est pas connu. En effet, des attaques basées sur des calculs d'anneaux d'endomorphismes des courbes permettent de calculer un raccourcis dans le graphe d'isogénies, et ainsi obtenir une évaluation en temps inférieur à T. Pour remédier à cela, il faut choisir la courbe de départ grâce à un tierce de confiance, ou bien effectuer au préalable une marche aléatoire à plusieurs afin de commencer l'étape d'initialisation avec une courbe dont personne ne connaît l'anneau d'endomorphismes. Cette utilisation de tierce personne est aussi récessaire dans le cas des VDF basées sur RSA, mais pas dans le cas de celles utilisant des idéaux du groupe de classes d'un corps quadratique. Nous proposons enfin une implémentation de nos VDF, disponible à l'adresse

`https://github.com/isogenies-vdf/isogenies-vdf-sage/`.

Nous obtenons en particulier les mesures suivantes pour un niveau de sécurité de 128 bits (sur une machine Intel Core i7-8700 @ 3.20GHz, avec SageMath 8.5) :

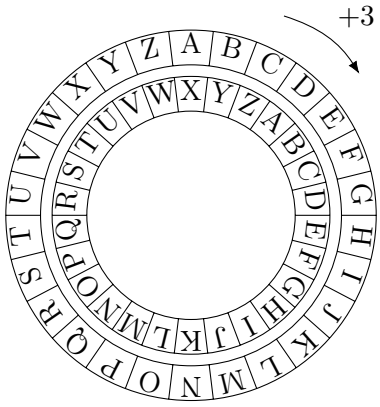
Protocole	Étape	taille de ek	Temps	Débit
$\mathbb{F}_p$ graph	Setup	238 kb	90 s	0.75isog/ms
	Evaluation	–	89 s	0.75isog/ms
	Verification	–	0.3 s	–
$\mathbb{F}_{p^2}$ graph	Setup	491 kb	193 s	0.35isog/ms
	Evaluation	–	297 s	0.23isog/ms
	Verification	–	4 s	–

List of Figures

Introduction

History of cryptography

Cryptography is the science of securing communication in the presence of third parties. The roots of cryptography are found in the Roman and Egyptian civilizations. One of the most famous examples of cryptographic encryption schemes is the shift of three letters, used by Caesar in order to communicate with his generals. It is clear that an exhaustive search on the different possible keys totally breaks this encryption scheme.



Symmetric and asymmetric cryptography

Before 1976, all the encryption schemes needed a secret key, and most of the designs themselves were kept secret. Diffie and Hellman proposed *new directions* in cryptography in [DH76]. This paper marked the begining of a new period for cryptography. They distinguish cryptography in two categories of schemes:

- The symmetric schemes are protocols where the two participants already share a secret in common. A symmetric cryptography encryption scheme can be illustrated as follows: Alice and Bob share a common key that opens a locker. In order to communicate, they simply put their message into a box and use the locker to keep the confidentiality of their message. We do not study the symmetric cryptography in this thesis. The standard for symmetric encryption is the AES scheme [AES01]. In order to use it, it is needed to share a common secret, which can be done using asymmetric cryptography.

- The asymmetric cryptography corresponds to all the communications where the participants do not have a common secret. In particular, digital signatures, key exchanges, and also zero-knowledge proof of identification fall in this category of cryptographic schemes. This category is also called public key cryptography in the sense that it uses some private and public keys. Encryption schemes based on public and private keys are also part of the asymmetric cryptography. Rivest, Shamir and Adelman introduced in 1978 a setting for public key cryptosystems [RSA78]. It is related to factorization of integers and is still widely used today.

- Bob chooses two large prime integers p and q corresponding to its secret key. He also computes its public key $N = pq$.
- From N, Alice encrypts a message using a specific algorithm we do not detail here.
- In order to decrypt, the knowledge of the factorization of N is essential. This problem of factorization of integers is known to have a complexity which is subexponential (we give a formal definition in the next section). Hence, Bob chooses large primes p and q so that nobody can get the factorization, and he is the only one able to decrypt the message.

Diffie and Hellman describe in [DH76] a way of sharing a common secret using a different mathematical object. We study in this thesis different structures related to asymmetric cryptography. Security of public key cryptography should be based on the hardness of some rare but well-known mathematical problems. The RSA encryption scheme is related to the integer factorization problem, while the Diffie-Hellman key exchange assumes that the discrete logarithm problem on the group considered is hard. We define in the next section the hardness of an algorithmic problem.

Algorithmic complexity and cryptographic security

Cryptographic schemes are said to be secure at a level λ if it takes more than 2^λ operations in order to break the scheme (get relevant information on the secret). The choice of the parameters of a public key cryptosystem is closely related to the best algorithm for solving the underlying mathematical problem. It is common to consider the asymptotic complexity of algorithms using the O notation. An algorithm has polynomial (resp. exponential) complexity if its asymptotic cost is polynomial (resp. exponential) in the size of the parameter inputs. We introduce the L notation in order to define sub-exponential complexity. Suppose that an algorithm takes q as an input. We define

$$L_q(\alpha) = \exp((\log q)^\alpha (\log\log q)^{1-\alpha}).$$

This notation allows us to represent a complexity which is between polynomial and exponential, using the parameter α. $L_q(0) = \log q$ is polynomial in the log of q, while $L_q(1) = q$ is exponential in $\log q$. For $0 < \alpha < 1$, $L_q(\alpha)$ corresponds to a complexity which is called subexponential.

The integer factorization and the discrete logarithm problem over a finite field have subexponential complexity with a parameter $\alpha = 1/3$. These complexities hold on a classical computer. A quantum computer would break these schemes in polynomial time.

The threat of quantum computers

Quantum computations have been introduced theoretically at the end of the twentieth century. Recently, investment into quantum computing research has increased in both the public and private sector. The possible existence of a quantum computer threaten the current cryptographic schemes. Both the factorization and the discrete logarithm problem would be computed in quantum polynomial time. In order to prevent this threat, the National Institute of Standards and Technology (NIST) began in 2017 a standardization campaign for post-quantum protocols. The considered candidates are based on different mathematical problems, sometimes new for the cryptography community.

In this thesis, we investigate mathematical objects related to asymmetric cryptography. More precisely, we focus on structures related to elliptic curves. We study both classical and post-quantum cryptography of several algorithms, depending on the context.

Organisation of the thesis

Chapter 1 introduces a first possible structure for designing key exchange. The group of invertible elements of a finite field is cyclic and looking at a prime order subgroup leads to efficient instantiations of key echanges. This chapter splits in two parts: first, we investigate the recent improvements on the Number Field Sieve (NFS) algorithm variants which compute the discrete logarithm over a finite field, and then we investigate the efficiency of the arithmetic in several sizes of finite fields.

Chapter 2 describes another structure used in practice for cryptography, called elliptic curves. These algebraic curves have a group structure and for a generic curve, solving the discrete logarithm problem has exponential complexity in the size of the group considered. We study the optimizations available for the group law and consider the set of endomorphisms of elliptic curves.

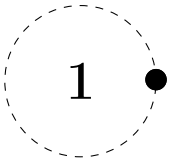
Chapter 3 considers morphisms between elliptic curves, called isogenies. Some cryptosystems based on isogenies of elliptic curves are not threatened by quantum computers so that they define post-quantum cryptography protocols. We investigate the computation of isogenies of large prime power degree in the case of elliptic curves called supersingular. We finally present several key exchanges based on different settings.

Chapter 4 is a study of particular families of curves allowing specific applications to cryptography. An efficient bilinear map relates groups of points of the elliptic curve to a multiplicative subgroup of a finite field. These curves are called pairing-friendly curves. We recall the different methods for generating such curves and study the choice of parameters in order to reach the 128-bit security level. Cryptosystems based on pairings are vulnerable to attacks on the discrete logarithm on the curve as well as in the finite field. Some of the curves are threatened by the recent variants described in Chapter 1. We finally estimate the cost of computing a pairing on these curves, for a 128-bit security level.

Chapter 5 investigates the construction of new families of curves resistant to the recent variants of NFS. We study the trade-off between security and efficiency for these new curves and compare them with the curves of Chapter 4.

Chapter 6 provides an implementation of algorithms from recent works. The security of the cryptographic schemes based on isogenies of supersingular curves relies on the fact that it is hard (i.e. the best algorithms are exponential in the parameters defining the curves) to compute endomorphism rings of the curves. We provide a code that allows us to compute these rings when an isogeny is known (i.e. the easy case).

Chapter 7 presents two constructions of specific functions called Verifiable Delay Functions (VDF). We use supersingular pairing-friendly curves so that the evaluation is based on composing isogenies of elliptic curves, and the verification evaluates pairings.



Finite fields

In this chapter, we instantiate examples of groups which bring applications to cryptography and are central in the theory of Chapters 4 and 5. In order to define these groups, we need to introduce finite fields. We denote $\mathbb{F}_q$ a field with q elements. If $q = p$ is prime, one can see $\mathbb{F}_p$ as the ring of integers modulo p which is indeed a field: every non-zero $x \in \mathbb{Z}/p\mathbb{Z}$ has an inverse which can be computed using the extended gcd algorithm). If $q = p^k$ with p a prime integer and $k \geq 1$, then one can define the field $\mathbb{F}_{p^k}$ to be the quotient of $\mathbb{F}_p[x]$ by a monic irreducible polynomial R of degree k: $\mathbb{F}_{p^k} := \mathbb{F}_p[x]/(R(x))$. Given this definition, $\mathbb{F}_{p^k}$ is a degree k extension of $\mathbb{F}_p$. Up to isomorphism, finite fields are unique in the sense that two irreducible polynomials of $\mathbb{F}_p[x]$ of degree k lead to isomorphic finite fields.

We consider the multiplicative subgroup of $\mathbb{F}_{p^k}$ denoted $\mathbb{F}_{p^k}^*$. Let g be a generator of the cyclic group $\mathbb{F}_{p^k}^* = \mathbb{F}_{p^k} - \{0\}$. The discrete logarithm problem in $\mathbb{F}_{p^k}^* = \langle g \rangle$ is hard in the sense that given $h \in \mathbb{F}_{p^k}^*$, the best algorithm to find $0 \leq x \leq p^k - 1$ such that $h = g^x$ has subexponential complexity in the input size $O(k \log p)$. We study this complexity in the case of particular finite fields in Section 1.1.

From this group, we obtain many applications to cryptography. In the seventies, Diffie and Hellman obtained a key exchange (see Figure 1.1) which is still very used in practice: Alice (resp. Bob) chooses a secret integer a (resp. b) and compute her public key $g_A = g^a$ (resp. $g_B = g^b$). From these values, they can compute the common secret $g^{ab} = g_A^b = g_B^a$. Originally, the Diffie–Hellman key exchange was designed for a multiplicative subgroup of a finite field, but it has been extended to other groups (for instance the group of points of an elliptic curve, see Chapter 2). This protocol is still deployed because the discrete logarithm problem and the related

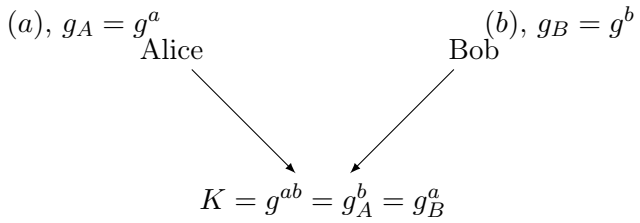


Figure 1.1: Diffie–Hellman key exchange in $\mathbb{F}_{p^k}^* = \langle g \rangle$.

Diffie–Hellman problem are still hard in various large finite fields. We present in Section 1.1

algorithms for computing discrete logarithms in different categories of finite fields. We present in particular estimations of these algorithms for finite fields that are relevant in the context of cryptographic applications of Chapters 4 and 5.

Summary

From now on, an element $x \in \mathbb{F}_{p^k}$ is represented as a polynomial, which is a representative of a coset of $\mathbb{F}_p[x]$ modulo $R(x)$, i.e. k coefficients in $\mathbb{F}_p$. In Section 1.2, we investigate the arithmetic computation in various finite fields (different size of prime p, and different extension degree k). In particular, we obtain timing estimations that will be our point of comparison for pairing efficiency in Chapters 4 and 5.

1.1 Discrete logarithm problem algorithms

We recall the discrete logarithm problem in the case of a generic cyclic group $G = \langle g \rangle$ of order ℓ: given $h \in G$, find $x \in \mathbb{Z}/\ell\mathbb{Z}$ such that $h = g^x$. We begin with a *generic* algorithm in the sense that it would work in any cyclic group, with polynomial time group operations (including comparison).

1.1.1 The baby-step–giant-step algorithm

We write the discrete logarithm x as $x = x_0 + \lceil\sqrt{\ell}\rceil x_1$, where $x_0, x_1 < \lceil\sqrt{\ell}\rceil$. In a first phase, we compute all the possible values for hg^{-x_0} and store them in a data structure that allows fast searching, together with the corresponding value of x_0. In the second phase, all values $g^{x_1\lceil\sqrt{\ell}\rceil}$ are computed, and each time we check whether the group element belongs to the list of elements computed in the first phase. When it matches, we have $h = g^{x_0+x_1\lceil\sqrt{\ell}\rceil}$ and we deduce the solution from x_0 and x_1. This algorithm computes $O(\sqrt{\ell})$ group operations while storing and sorting $O(\sqrt{\ell})$ group elements. The time and space complexity is $\tilde{O}(\sqrt{\ell})$.

In practice, other algorithms achieve the $\tilde{O}(\sqrt{\ell})$ complexity together with essentially no memory. The Pollard-Rho algorithm has the advantage to be easily parallelizable, even though it is not rigorously analyzed without assumptions on the existence of hash functions with nice properties.

We investigate in Chapter 2 groups whose order is as large as 2^{256}. For these groups, the best algorithm for solving the discrete logarithm problem has $O(\sqrt{\ell})$ complexity so that we reach the 128-bit security level ($\sqrt{\ell} \approx 2^{128}$).

1.1.2 Index calculus method

In the case of finite fields, we deal with a multiplicative subgroup G of $\mathbb{F}_{p^k}$. In this context, the index calculus method depends on a chosen factor base S composed of small elements of G. The best algorithms for computing discrete logarithms in finite fields are based on this method, originally from Adleman [Adl79]. The index calculus method splits in three steps:

1. Relations collection. We aim to get several relations $g^{a_i} = \prod_{q \in S} q^{e_{q,i}}$. To do so, we try different values for a_i until we obtain an element g^{a_i} whose prime factors are all in S. The cost of the relation collection really depends on the algorithm used to get the S factors of g^{a_i}.

2. Linear algebra. By taking logarithms of the previous relations, we obtain linear equations modulo the order ℓ of the group considered: $a_i \equiv \sum_{q \in S} e_{q,i} \log q \pmod{\ell}$, where the unknowns are the $\log q$ for $q \in S$. If enough relations are collected in Step 1, we compute the logarithms of the set S by solving a matrix kernel whose rows are the exponents in the relations collection.

3. Find a relation between h and the small elements and recover the discrete logarithm x from the logarithms obtained in Step 2.

Remark 1.1. Note that the first two steps of the index calculus method do not depend on h and can be seen as precomputations. It means that once Steps 1 and 2 are done, one only needs to execute Step 3 to get the discrete logarithm of h.

The index calculus algorithms we study here are based on sieving in different algebraic structures in order to get the relations of Step 1 above. Depending on the size of the parameters defining the finite field, the efficiency of the algorithms varies. When p is relatively small, quasi-polynomial time algorithms can be designed [BGJT14], but when p grows, the most efficient algorithms have subexponential complexity, namely $L_{p^k}(1/3, c + o(1))$ where $L_{p^k}(\alpha, c) = \exp(c(\log p^k)^\alpha (\log \log p^k)^{1-\alpha})$. Choosing the suited algorithm depends on the size of p compared to p^k.

Definition 1.2 (Small, medium, and large characteristic)**.** *Let $(p_i)_{i \in \mathbb{N}}$ be a sequence of prime integers, and $(k_i)_{i \in \mathbb{N}}$ be a sequence of integers such that $9_i^{k_i} \to +\infty$. Then, the family of finite fields $\mathbb{F}_{p_i^{k_i}}$ is said to be:*

- *of* small *characteristic if there exists $c > 0$ such that the primes p_i can be parametrized as $p_i = L_{p_i^{k_i}}(\alpha_i, c)$ for $\alpha_i < 1/3$,*
- *of* medium *characteristic if there exists $c > 0$ such that the primes p_i can be parametrized as $p_i = L_{p_i^{k_i}}(\alpha_i, c)$ for $\alpha_i \in]1/3, 2/3[$,*
- *of* large *characteristic if there exists $c > 0$ such that the primes p_i can be parametrized as $p_i = L_{p_i^{k_i}}(\alpha_i, c)$ for $\alpha_i > 2/3$.*

Quasi-polynomial time algorithms [BGJT14, GKZ14] are efficient in small characteristic, but we consider here fields of medium and large characteristic. In these cases, the best algorithms for computing discrete logarithms are related to sieving in number fields.

1.1.3 Sieving in number fields

From now on, we consider fields of medium and large characteristic: the prime fields correspond to $p = L_p(1, 1)$ and the non-prime fields are of the form $\mathbb{F}_{p^k}$ with $256 \leq \log_2(p) \leq 672$ and $5 \leq k \leq 16$. For this range of sizes, the "large" or "medium characteristic" algorithms are the ones that perform best. The computation of discrete logarithms in these types of finite fields is addressed by the Number Field Sieve algorithm (NFS). It computes the relations collection by sieving in subrings of number fields.

The Number Field Sieve has been introduced in 1990 for large integer factorization, but is also suited for discrete logarithm computations over finite fields. Today, the largest computation records [BGG+20] for integer factorization as well as for discrete logarithm over $\mathbb{F}_p$ have been done using an efficient implementation of the NFS algorithm (CADO-NFS [Tea17]). The idea is to search for relations using two number fields as follows:

- Define two subrings of number fields using particular irreducible polynomials $f_1(x)$ and $f_2(x)$. These polynomials need to have appropriate degrees and coefficient sizes, and a common factor $R(x)$ modulo p (where $\deg R = k$).
- Select a range of polynomials of the form $a - bx \in \mathbb{Z}[x]$.
- Map them into two different number fields by reducing modulo $f_1(x)$ and $f_2(x)$ and collect the relations in these two different structures. Reducing modulo f_i $(i = 1, 2)$, we obtain elements of the form $a - b\alpha_i$ where α_i is a root of the polynomial f_i. In number fields, the prime factorization is not always unique. That is why we deal with factorization in prime ideals. This way, we look for ideals $(a - b\alpha_1)$ having few (small norm) prime factors. The factorization in prime ideals is closely related to the factorization of the norm of the ideal, which is in this case $N(a - b\alpha_i) = b^d f_i(a/b)$. When this integer is B-smooth, we know that the principal ideal $(a - b\alpha_i)$ splits into a product of prime ideals of norm less than B: $(a - b\alpha_i) = \prod_{N(\mathfrak{q})<B} \mathfrak{q}^{e_\mathfrak{q}}$.
- Choosing f_1 and f_2 so that the two algebraic structures have nothing to do with each other, we obtain relations when ideals have B-smooth norm in both sides of Figure 1.2. Then, we map these relations into $\mathbb{F}_{p^k}$ using a reduction modulo $(p, R(x))$. Technical difficulties arise from the fact that only elements, and not prime ideals in the number field, can be reduced modulo the prime ideal $(p, R(x))$. These obstructions are dealt with using standard techniques which we do not detail here.
- Finally, we end the index calculus algorithm using sparse linear algebra for recovering the logarithms of the small elements, and then the targeted discrete logarithm. We warn the reader that we do not investigate here the technicalities of the NFS algorithm.

We refer to [LLJ93] for a complete presentation of the Number Field Sieve algorithm. The NFS setup is often represented with the commutative diagram of Figure 1.2.

The polynomial selection is an essential step for the NFS efficiency. The two polynomials $f_1(x)$ and $f_2(x)$ need to have small enough coefficients for the algorithm to perform well. We develop in the next paragraphs several choices that are well adapted with NFS when parameters become large.

The original NFS setup. The initial NFS algorithm was set for a prime finite field $\mathbb{F}_p$. It corresponds to $\deg R = 1$ in Figure 1.2. We choose the two polynomials $f_1(x)$ and $f_2(x)$ so that

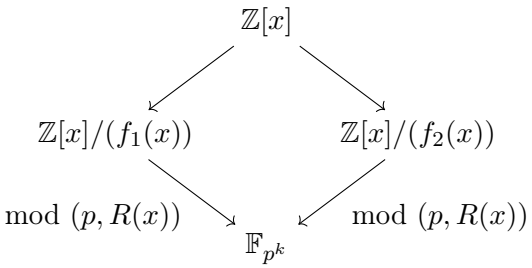


Figure 1.2: Commutative diagram for NFS.

we get relations in two well-known rings: on the one hand, we look for relations in $\mathbb{Z}$, and on the other hand, we study ideals of a number field of degree d. Note that the degree d is tuned after a complexity analysis of the whole algorithm. We do not investigate further, but the asymptotic value of d is the integer closest to $(3 \log p / \log\log p)^{1/3}$ [Gor93, Sch93]. In practice, d is often between 4 and 7. The original NFS setup selects the two polynomials $f_1(x), f_2(x) \in \mathbb{Z}[x]$ as follows:

- We first choose an irreducible polynomial $f_1(x) \in \mathbb{Z}[x]$ of degree d having a root m modulo p. To do so, we set m to be an integer close to $p^{1/d}$ and write p in base m. The coefficients of p in base m define an irreducible polynomial $f_1(x) = \sum_{i=0}^{d} f_{1,i} x^i$ so that $p = \sum_{i=0}^{d} f_{1,i} m^i = f_1(m)$ and the $f_{1,i}$ are bounded with m. This lets us define a subring $\mathbb{Z}[x]/(f_1(x))$ of the number field $\mathbb{Q}[x]/(f_1(x))$ (the left part of Figure 1.2). From now on, we write this number field $\mathbb{Q}(\alpha_1)$ (i.e. $f_1(\alpha_1) = 0$).
- From this polynomial, we simply set $f_2(x) = x - m$ so that f_1 and f_2 share a common root m modulo p. It corresponds to the right part of Figure 1.2 and as we choose a degree 1 polynomial, the order $\mathbb{Z}[x]/(f_2(x)) = \mathbb{Z}[m]$ is isomorphic to $\mathbb{Z}$.

From this setup, we get relations by trying several polynomials of the form $a - bx \in \mathbb{Z}[x]$, and map them in the two number fields. We do not investigate the detailed complexity of each step of the algorithm but simply give the final asymptotic complexity: $L_p(1/3, \sqrt[3]{64/9} + o(1))$. We refer the reader to [LLJ93, Gor93, Sch93] for details on the complexity of the algorithm.

We present few variants of the original setup, which improve in particular cases the NFS cost. The Joux-Lercier algorithm [JL03] applies for prime fields, and have the same asymptotic complexity $L_p(1/3, \sqrt[3]{64/9} + o(1))$.

The Joux-Lercier variant. Joux and Lercier propose in [JL03] another polynomial selection, which works with two number fields of defining polynomials of degree d and $d-1$:

1. Choose $f_1(x)$ of degree d with tiny coefficients, irreducible over $\mathbb{Q}$, with a root m modulo p.
2. Define the vector subspace in $\mathbb{Z}[x]$ of all the polynomials multiple of p and $(x-m)$ and of degree up to $d-1$: $\mathbb{Z}[x]\langle p, x-m, x(x-m), x^2(x-m), \ldots, x^{d-2}(x-m)\rangle$. One can represent this set with a matrix whose rows are p and the $x^i(x-m)$ in the canonical basis $(1, x, x^2, \ldots, x^{d-1})$:

$$\begin{pmatrix} p & 0 & \ldots & 0 \\ -m & 1 & 0 & \vdots \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & -m & 1 \end{pmatrix}$$

3. Using lattice based reduction on the matrix, obtain a short vector corresponding to a polynomial f_2 with small coefficients. More precisely, $f_2(x) \in \mathbb{Z}[x]$ and $\|f_2\|_\infty = O(p^{1/d})$ and $f_2(m) = 0 \bmod p$.

In the same way as in the left part of the initial setup of NFS, we need to work with principal ideals in both $\mathbb{Z}[x]/(f_1(x))$ and $\mathbb{Z}[x]/(f_2(x))$. The complexity of this variant is also $L_{p^k}(1/3, \sqrt[3]{64/9} + o(1))$, but we will see in Section 1.1.4 that sometimes it performs better than the initial setup. The algorithm is generalized to any (non-prime) finite field in [Bar13, BGGM15], with the same complexity. Other variants apply to non-prime fields. We mention only two variants that work in non-prime fields.

- The generalized Joux-Lercier [Bar13] works in large characteristic non-prime fields with complexity $L_{p^k}(1/3, \sqrt[3]{64/9} + o(1))$.
- The Joux-Lercier-Smart-Vercauteren variant [JLSV06] applies for a larger range non-prime fields (including medium characteristic) with complexity $L_{p^k}(1/3, \sqrt[3]{128/9} + o(1))$.
- In medium characteristic, the Conjugation variant [BGGM15] is preferred. Its asymptotic complexity is $L_{p^k}(1/3, \sqrt[3]{96/9} + o(1))$.

A significant improvement was obtained in 2013 by Joux and Pierrot [JP14] in some particular cases when p is *special* in the sense of a definition we give in the next paragraph.

The special variant. In [JP14], Joux and Pierrot introduce a variant of NFS when the characteristic p of the field $\mathbb{F}_{p^k}$ can be parameterized by a polynomial with several properties. This new variant improves the constant in the exponent of the NFS complexity, and applies for many applications in *pairing-based* cryptography, studied in Chapters 4, 5 and 7. The situation is quite similar to the Special Number Field Sieve (SNFS) situation of old [Gor93], which allows for particularly efficient factoring of numbers of a special form. As it turns out, extensions of this "special form" benefit apply to discrete logarithm context as well, which is why the Joux-Pierrot method is also referred to as an SNFS(-like) algorithm. For the Special Number Field Sieve to apply, the prime p needs to be parametrized by a *special* prime in the sense of the following informal definition.

Definition 1.3 (Special prime)**.** *A prime p is* special *if $p = P(u)$ where $P(x)$ has degree d at least 3, and coefficients significantly smaller than $p^{1/d}$.*

In this context, Joux and Pierrot slightly change the choice of polynomials in the NFS setting. The polynomial f_1 is chosen of the form $f_1(x) = x^k + s(x) - u$. $s(x)$ is a polynomial of small degree with coefficients 0, 1 and -1, such that $f_1(x)$ is irreducible over $\mathbb{Z}$. As $u \approx p^{1/\deg P}$, the f_1 coefficients are small enough for the NFS efficiency. Then, they fix $f_2(x) = P(x^k + s(x))$ so that $\deg f_2 = k \deg P$ and its coefficients are small enough (in $O(\log_2(k)^{\deg P})$, see [JP14]). Modulo $f_1(x)$, $x^k + s(x) \equiv u$ and so $f_2(x) \equiv P(u) = p$. Finally, $f_2(x)$ is a multiple of $f_1(x)$ modulo p, and $\gcd(f_1, f_2) \equiv f_1(x) \bmod p$ is an irreducible polynomial of degree k, which leads to an efficient NFS variant. Once again, we do not precisely detail the complexity analysis, but the SNFS setup leads to a very different asymptotic cost: $L_{p^k}(1/3, \sqrt[3]{32/9} + o(1))$. In practice, this smaller constant is significant at the 128-bit security level. This variant applies only when the prime is special in the sense of Definition 1.3. We now describe another variant which applies in towers of finite fields.

NFS variant	**Characteristic**	**Asymptotic complexity** $L_{p^k}(1/3, c + o(1))$
Initial setup	Large	$c = \sqrt[3]{64/9}$
(generalized) Joux-Lercier	Large	$c = \sqrt[3]{64/9}$
Special	Large, special	$c = \sqrt[3]{32/9}$
Conjugation	Medium	$c = \sqrt[3]{96/9}$
(extended) Tower	Medium	$c = \sqrt[3]{48/9}$
Special Tower	Medium, special	$c = \sqrt[3]{32/9}$

Table 1.1: Asymptotic complexities of the Number Fied Sieve variants

The (extented) tower variant. Barbulescu, Gaudry and Kleinjung introduced in 2015 the Tower Number Field Sieve (TNFS) in [BGK15]. The main idea of the tower variant is to work with polynomials whose coefficients are in $R = \mathbb{Z}[x]/(h(x))$ instead of $\mathbb{Z}$, where h is a monic irreducible polynomial of degree k. In order to get a diagram similar to Figure 1.2, we also require having a unique ideal $\mathfrak{p}$ above p in R, in order to map our polynomials into $\mathbb{F}_{p^k}$. This setup is improved in [KB16] when there exists subfields of $\mathbb{F}_{p^k}$, which means that k is composite. In this context, write $k = \eta\kappa$ with $\eta, \kappa \neq 1$. Then, choose $h(x)$ and $\phi(x)$ to be monic irreducible polynomials of degree η and κ. This way, the extended TNFS is summarized in Figure 1.3, where R is the order $\mathbb{Z}[t]/(h(t))$.

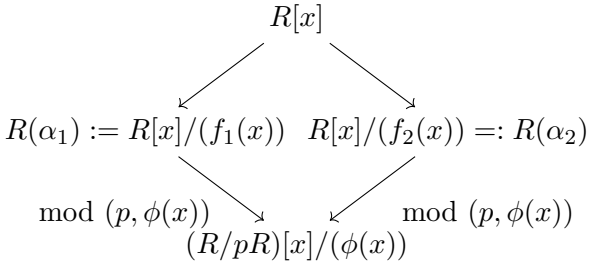


Figure 1.3: Commutative diagram for TNFS.

The complexity of this algorithm depends on the choice of the subfields of $\mathbb{F}_{p^k}$ (and hence the choices for $h(x)$ and $\phi(x)$). If k is prime, then the extended TNFS corresponds to the initial TNFS of [BGK15] and is often not more efficient than the JSLV setting. The asymptotic complexity of the TNFS variant is $L_{p^k}(1/3, \sqrt[3]{48/9} + o(1))$. This (extended) tower variant can also be adapted to a special setting.

The special tower variant. The best complexity of the NFS variants is achieved for a special prime together with proper subfields. Barbulescu, Gaudry and Kleinjung in [BGK15], and then Kim and Barbulescu in [KB16] adapt the Joux-Pierrot special variant with their tower NFS. The efficient choices of $f_1(x)$ and $f_2(x)$ from SNFS together with $h(x)$ and $\phi(x)$ from TNFS lead to the STNFS setting whose asymptotic complexity is $L_{p^k}(1/3, \sqrt[3]{32/9} + o(1))$.

1.1.4 Practical examples

We end this NFS exposition with two typical examples: first, a prime field with a non-special prime p, and then a degree $k = 12$ extension of a prime field where the special variants apply.

The Number Field Sieve is efficiently implemented in practice (see [Tea17]). Not all the variants presented in Table 1.1 are implemented, but we are able to estimate the cost of these sieving methods more precisely using the tool provided in the URL below, which is more accurate than the asymptotic complexities.

`https://gitlab.inria.fr/tnfs-alpha/alpha`.

The reader can refer to [GS19] for more information on these cost estimations.

A non-special prime field

We begin with a simple example in the sense that only few variants apply. We consider here a prime field so that the tower variants do not make sense, and we also consider p that is not special in the sense of Definition 1.3: $p = r^2A^2 + 1$ where $r = 2^{256} - 2^{194} - 1$ and A is a random integer of 1280 bits such that $Ar \equiv 2 \bmod 4$. For instance, let

$$\begin{aligned} A = \; & 11178383037109115702629987811692648017014796539562957939109071944 \\ & 75572592246254514114919885913704393021436290446967275237352961782 \\ & 70127869708198088399326830893938384641987509100633741961709235621 \\ & 70330515999228978847736605604407946688366081823166084678395047320 \\ & 55398961073319567775782234203412020187048509905393811895892963500 \\ & 93539194274574891234277822747988638997463456478667899637036 54. \end{aligned}$$

In this context, p is 3072-bit long and we do not know how to get a polynomial representation of p whose coefficients would be small enough. We present here two choices for f_1 and f_2:

- Using the initial NFS setup without any optimization on the choice of m, we can obtain the two polynomials

$$f_1(x) = x^6 + 3x^5 + y_4x^4 + y_3x^3 + y_2x^2 + y_1x + y_0 \qquad f_2(x) = x - m$$

where

$$\begin{aligned} y_4 = \; & 60138388361716476232440690049882099433074816645243 33 \\ & 01336411400727797150505544871506275979376696779854 33 \\ & 06502321617502421319604655528618875428667796163 1 \\ y_3 = \; & 11908259399253214494052404897926665287390624055527 95 \\ & 14216407141262248855494662682991742760850095240374 22 \\ & 00569606590319347074084137852070268588076335084799 7 \\ y_2 = \; & 37067653986733344938906328265369533259222322169507379 \\ & 69592424576192220737372388441729761518731957108794 78 \\ & 04485505733815758926507015594935745934944368164561 \\ y_1 = \; & 10968364509106431408969047361721145857513772420653 57 \\ & 21046191988111319512246922991737354935487258539529 66 \\ & 75998359500690828580810018355849983890811876071435 2 \\ y_0 = \; & 54695091334312776084704598546566299036731110861287 77 \\ & 33968360595089445799000609510946827655730590038152 77 \\ & 23004488946983628845237207121578253930500006345 80 \\ m = \; & 12815797734573391168852833799499604832208208834270 72 \\ & 38867727676426198660215684816416518729748895501432 68 \\ & 21257757425456409509980270276882666805517303299627 9. \end{aligned}$$

We estimate that the discrete logarithm computation would run in 2^{144} operations.

- The Joux-Lercier setting is the best choice for the polynomial selection in this case. Choosing $\deg f_1 = 7$ and $\deg f_2 = 6$, we get (for instance using [Tea17]) the two polynomials

$$f_1(x) = -3 - x - x^2 + 3x^3 - 2x^4 - 2x^5 + 2x^6 + x^7, \qquad f_2(x) = \sum_{i=0}^{6} z_i x^i$$

where

$$\begin{aligned}
z_6 = {} & -46800310601481545507866752844141956277546267486478459749781848986037592233749521206351243658858499476739812272153823059567280985030\\
z_5 = {} & 10714907134954872378031691249407325924118884936363209797120534130692160399264697678490314043012288417329545663451873823516131749980364\\
z_4 = {} & 68818721436787611162640195817935838503950421203957383302536593924926993090594418755935889816624609610414535946923352100217224630472\\
z_3 = {} & -97022949253728485938677307360245093005113238507365475837993776402871461365098080899700750867723291458086351275735285134538185097606\\
z_2 = {} & -14488940796680944552093814628361778723938463768625552418344211367843832259506430637230022956767807813439553779487172132022414203650\\
z_1 = {} & 41316577326908071052605637463856298159134533985901951929396443568306304908235832560351196931435474521986404643943830200139256909430\\
9 = {} & -28023744887988015874670809546798820390369281390006053910630023985011604394686840870426199955638183027691976910056234686436069710995 4.
\end{aligned}$$

  We obtain that the estimated cost is approximately 2^{128} operations for this range of parameter sizes.

Finally, we can consider that the field $\mathbb{F}_p$ reaches the 128-bit security level in the sense that the best algorithm would cost at least 2^{128} operations.

A field where the special tower variant applies

We consider a finite field for which all the variants apply. We do not detail how to estimate the practical cost of the NFS variants and we only give the asymptotic complexity. However, the computation of discrete logarithms is an active research subject and even if some of the variants presented above are not implemented yet, we are able to estimate the practical cost of the variants using for instance [GS19] together with their code mentioned above. We now look at $\mathbb{F}_{p^{12}}$ where $p = P(u) = 36u^4 + 36u^3 + 24u^2 + 6u + 1$. We consider $u = -2^{62} - 2^{55} - 1$; one can check that $P(u)$ is indeed a prime integer of 254 bits. We now estimate the complexity of two NFS variants.

- Using the special parameterization of p, one can choose f_1 and f_2 as in the SNFS variant:

$$f_1(x) = x^{12} + x^2 - x - u \qquad f_2(x) = P(x^{12} + x^2 - x)$$

  Using this setup, the NFS algorithm cost estimation is 2^{128} operations.

- Using the work of Guillevic and Singh, we obtain the polynomial choices for the STNFS variant:

$$\begin{aligned}
h(t) &= t^6 + t^3 + 2t^2 - 2t + 1\\
f_1(x) &= 36x^8 + 36tx^6 + 24t^2x^4 + 6t^3x^2 + t^4 \in R[x]/(h(t))\\
f_2(x) &= x^2 - 4593689212103950336t \in R[x]/(h(t)).
\end{aligned}$$

  The sieving method cost estimation using their code is 2^{103} operations.

In order to reach the 128-bit security level for the discrete logarithm problem over this field $\mathbb{F}_{p^{12}}$, one needs to take a larger seed u. Choosing $u = 2^{110} + 2^{36} + 1$ as in [PSNB11], we obtain a 5343-bit finite field and all the NFS variants would run in at least 2^{132} operations. The best parameter choice for the STNFS variant is obtained for $\deg h = 6, \deg f_1 = 8$ and $\deg f_2 = 2$ as above.

1.2 Arithmetic of finite fields

In the next chapters, we estimate the cost of several computations involving finite field operations. We investigate the cost of the arithmetic of various finite fields in this section. As we aim at the cryptographic size fields for a 128-bit security level, we study a range of fields whose size $\log_2(p^k)$ is between 3000 bits (Section 1.1.4) and 5500 bits (Section 1.1.4). In particular, we study prime fields arithmetic where $\log_2(p) = 3072$ as in Section 1.1.4, and operations in extension fields $\mathbb{F}_{p^k}$ where $\log_2(p) \geq 256$ and $5 \leq k \leq 16$, with different sizes of p^k.

The finite field operations are closely related to polynomial and integer arithmetic. We consider here the main operations in finite fields: multiplications, squarings, inversions. Note that we neglect the cost of additions and multiplications by small constants, which are not significant compared to multiplications. We also investigate exponentiations, building blocks of the Diffie–Hellman key exchange of Figure 1.1. In particular, we look for optimizations for particular exponents (Frobenius power, exponentiations in particular subgroups of $\mathbb{F}_{p^k}^*$).

Notations. From now on, we denote $\mathbf{m}$ (resp. $\mathbf{s}$, $\mathbf{i}$) the cost of one multiplication (resp. a squaring, an inversion) in $\mathbb{F}_p$. For a field $\mathbb{F}_{p^k}$, we denote $\mathbf{m}_k$ (resp. $\mathbf{s}_k$, $\mathbf{i}_k$, $\mathbf{f}_k$, $\mathbf{c}_u$) to be the cost of a multiplication (resp. a squaring, an inversion, a Frobenius, an exponentiation to the power u). We estimate the arithmetic cost in extension fields by counting the number of multiplications in $\mathbb{F}_p$.

1.2.1 Multiplications

We consider prime fields and extension fields separately. For prime fields, we measure the $\mathbb{F}_p$ multiplications for various sizes of p using an efficient library, whereas we estimate $\mathbf{m}_k$ in terms of $\mathbf{m}$, and use our previous measurements to get a time estimation.

Measurements in prime fields

We represents elements of $\mathbb{F}_p$ as integers of size $\log_2(p)$. As for integer arithmetic, the cost of additions and multiplications by small constants are neglected with respect to the $\mathbb{F}_p$ multiplications.

The RELIC library [AG] provides an efficient finite field arithmetic for various sizes of primes. In particular, we are able to measure multiplications over $\mathbb{F}_p$ for various sizes of primes p. As we target the 128-bit security level for the discrete logarithm problem over $\mathbb{F}_{p^k}^*$, we will consider fields of size p^k in the range of the two examples of Section 1.1.4. Hence, we consider large 3072-bit primes p, and also smaller primes corresponding to $k > 1$, with four to eleven 64-bit machine words (i.e. $192 < \log_2(p) \leq 704$). The RELIC library provides measurements for integers of maximum 10 machine-words. We use a Intel Core i7-8700 CPU, 3.20GHz with TurboBoost disabled in order to get reproducible timings, and then estimate the cost of $\mathbf{m}$ in the case of eleven machine-words using the interpolation of Figure 1.4.

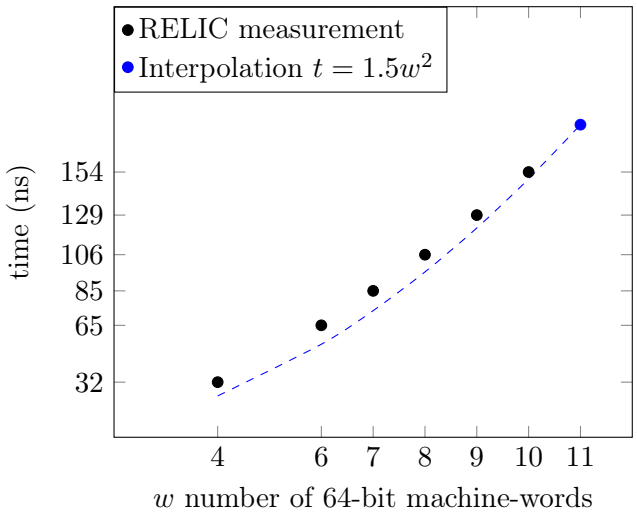


Figure 1.4: Measurement and interpolation of a multiplication modulo different bitlength primes.

For the larger size $\log_2(p) = 3072$, we use the same machine and measure a multiplication of 48 machine-word integers using the GNU MP library [Gt20]. We finally obtain that it costs 3800ns.

Estimations in extension fields

In order to provide a common comparison base, we give estimated costs for multiplications using Karatsuba-like formulas [DÓSD06, Mon05, CH07]. We treat squarings and multiplications separately because the formula differ slightly. In Chapters 4 and 5, we are interested in finite fields of degree $k \in \{5, 6, 7, 8, 12, 16\}$. Hence, we consider here extensions of degree 2, 3, 5 and 7 as in Figure 1.5.

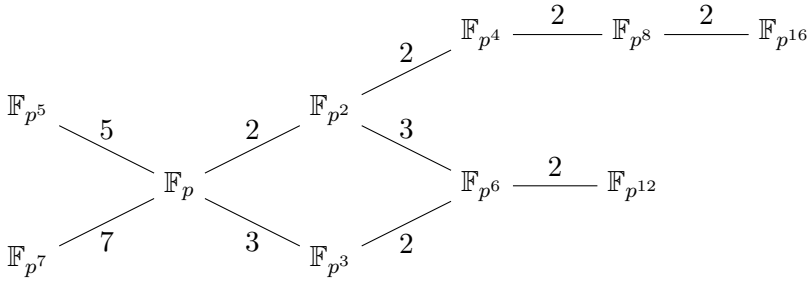


Figure 1.5: Towers of extension of degree 2, 3, 5 and 7 considered here.

Remark 1.4 (Binomial extensions). From now on, we suppose that an extension $\mathbb{F}_{q^k}$ of $\mathbb{F}_q$ is defined using a *binomial* irreducible polynomial $P = x^k - \alpha$ where $a \in \mathbb{F}_q$. Using this construction $\mathbb{F}_{q^k} = \mathbb{F}_q(x)$ where x is a root of P, $x^k = \alpha^k$ in $\mathbb{F}_{q^k}$ and several computations can be optimized (see Section 1.2.2). Over a prime field $\mathbb{F}_p$, $p \equiv 1 \bmod k$ is necessary in order to build the extension $\mathbb{F}_{p^k}$ over $\mathbb{F}_p$ with a binomial irreducible polynomial.

For degree 2 and 3 extensions, the costs come directly from the Karatsuba formulas while for prime extension degrees $k = 5$ and 7, we use the slightly different formulas from [Mon05]. We

neglect here the cost of multiplications by small constants.

$$\mathbf{m}_k = \begin{cases} 3\mathbf{m}_{k/2} & \text{if } k \text{ is even,} \\ 6\mathbf{m}_{k/3} & \text{if 3 divides } k, \\ 13\mathbf{m} & \text{if } k = 5, \\ 22\mathbf{m} & \text{if } k = 7, \end{cases} \qquad \mathbf{s}_k = \begin{cases} 2\mathbf{m}_{k/2} & \text{if } k \text{ is even,} \\ 2\mathbf{m}_{k/3} + 3\mathbf{s}_{k/3} & \text{if 3 divides } k, \\ 13\mathbf{m} & \text{if } k = 5, \\ 22\mathbf{m} & \text{if } k = 7 \end{cases}$$

Sparse multiplications

We also investigate the case of *sparse* element multiplications. In Chapters 4 and 5, we are interested in the cases of extension degrees divisible by 4 or 6. Elements of $\mathbb{F}_{q^d}$ ($d \in \{4, 6\}$) are represented with polynomials with d coefficients in $\mathbb{F}_q$. If both of the elements we multiply have coefficients alternating zero and non-zero integers, we estimate that it simply corresponds to a multiplication in $\mathbb{F}_{q^{d/2}}$. However, if only one element has few zeros, it is a bit trickier. We call these special cases *dense*×*sparse* multiplications, and consider the particular case where the sparse element has two or three non-zero coefficients. More precisely, we investigate the cases of interest of Chapters 4 and 5:

- In $\mathbb{F}_{q^4}$, we consider a dense element $a = a_0 + a_1x + a_2x^2 + a_3x^3$ and a sparse element with only two non-zero coefficients $b = b_1x^{i_1} + b_2x^{i_2}$, where $0 \leq i_1, i_2 \leq 3$, and $a_i, b_i \in \mathbb{F}_q$. A dense×sparse multiplication can be computed in 8 multiplication in $\mathbb{F}_q$ (instead of 9 with the two Karatsuba formulas).
- In $\mathbb{F}_{q^6}$, we consider (with similar notations) a sparse b with three zero coefficients in $\mathbb{F}_q$, and using [Gui13, page 92], the computations of $a \cdot b$ costs 13 multiplications in $\mathbb{F}_q$ instead of 18.

1.2.2 Frobenius power

A common computation in finite fields is the exponentiation to the power p (the characteristic of the field concerned). Obviously, the Frobenius is trivial over prime fields: $\omega_0^p = \omega_0$ for $\omega_0 \in \mathbb{F}_p$. Over extension fields, it is very useful to define the tower using a binomial polynomial in order to perform the Frobenius power. This is possible only when $p \equiv 1 \bmod k$ (see Remark 1.4). Let $w = \sum_{i=0}^{k-1} \omega_i x^i \in \mathbb{F}_{p^k} = \mathbb{F}_p[x]/(x^k - \alpha)$. Then, $w^{p^j} = \omega_0 + \sum_{i=1}^{k-1} \omega_i x^{ip^j}$. The terms x^{ip^j} do not depend on w and are precomputed. By Euclidean division by k, $x^{ip^j} = x^{u_j k+i} = \alpha^{u_j} x^i$. Therefore we have at most $\mathbf{f}_k = (k-1)\mathbf{m}$ for any p^j-th power Frobenius. Note that for k even, we have $x^{k/2\cdot(p^j-1)} = \alpha^{(p^j-1)/2} = \pm 1$ so that $x^{k/2\cdot p^j} = \pm x^{k/2}$, whence one multiplication can be saved.

1.2.3 Inversions

We estimate the cost of inversions in extension fields in terms of $\mathbf{m}$ as in the previous sections. We also estimate $\mathbf{i} \approx 25\mathbf{m}$; this is clearly implementation-dependent (as well as our measurements of Section 1.2.1). The computation of an inversion is closely related to efficient Frobenius powers:

$$a^{-1} = (\mathrm{Norm}_{\mathbb{F}_{q^k}/\mathbb{F}_q}(a))^{-1} \times a^q \times \cdots \times a^{q^{k-1}}.$$

Consequences of the above are given in [WS07], notably $\mathbf{i}_2 = 2\mathbf{m} + 2\mathbf{s} + \mathbf{i}$ and $\mathbf{i}_3 = 9\mathbf{m} + 3\mathbf{s} + \mathbf{i}$, neglecting additions. Recursive application yields $\mathbf{i}_k$ for $k = 2, 3, 4, 6, 8, 12, 16$. For $k = 5$, we compute $t := a^q \times \cdots \times a^{q^4} = ((a^q)^{1+q})^{1+q^2}$ and then use the fact that the norm of a is in $\mathbb{F}_p$ to get $\mathrm{Norm}_{\mathbb{F}_{p^5}/\mathbb{F}_p}(a) = a \times t = a_0t_0 + \alpha \sum_{j=1}^{k-1} a_i t_{k-i}$. We finally obtain that $\mathbf{i}_5 = 3\mathbf{f}_5 + 2\mathbf{m}_k + \mathbf{i} + 10\mathbf{m}$, and $\mathbf{i}_7$ is obtained in a similar way.

1.2.4 Exponentiations

Exponentiation is the basic operation of the Diffie–Hellman key exchange in a finite field, as presented in Figure 1.1. Exponentiations can be computed efficiently with Algorithm 1.1. It computes $\log_2(n) - 1$ squarings and $\mathrm{HW}(n) - 1$ multiplications, where $\mathrm{HW}(n)$ is the Hamming weight of n.

Many optimizations can improve this cost depending on the context, but the asymptotic complexity stays linear in the size of the exponent n. We detail here the non-adjacent form exponentiation.

Exponentiation in NAF. In some particular finite fields (for instance in $\mathbb{F}_{2^{127}-1}$), inversions can be computed efficiently so that the square-and-multiply algorithm is more efficient when designed with another representation than the binary form.

Definition 1.5 (Non-adjacent form, [HMV03, page 98])**.** *The non-adjacent form (NAF) of an integer n is the unique sequence $\{n_0, \ldots, n_d\}$ of 0, 1 and -1 such that $n = \sum_{i=0}^{d-1} n_i 2^i$, and two non-zero values cannot be adjacent.*

Non-adjacent form has more zeros than the binary form, and so an exponentiation can be performed more efficiently when inversions are cheap. Algorithm 1.2 costs $\log_2(n)$ squarings and $\mathrm{HW}_{\text{2-NAF}}(n)$ multiplications or (efficient) inversions.

Algorithm 1.1: Square-and-multiply(x, n)

Input. $x \in \mathbb{F}_{p^k}$,
n an integer.
Output. x^n.
Cost. $(\log_2(n) - 1)\mathbf{s}_k + (\mathrm{HW}(n) - 1)\mathbf{m}_k$.

Write $n = \sum_{i=0}^{d} b_i 2^i$ with $b_i \in \{0, 1\}$
$R \leftarrow 1$
for i from $d - 1$ downto 0 **do**
  $R \leftarrow R^2$
  if $b_i = 1$ **then**
    $R \leftarrow R \cdot x$
  end if
end for
return R

Algorithm 1.2: NAF-square-and-multiply(x, n)

Input. $x \in \mathbb{F}_{p^k}$,
n an integer.
Output. x^n.
Cost. $(\log_2(n)-1)\mathbf{s}_k + (\mathrm{HW}_{\text{2-NAF}}(n)-1)\mathbf{m}_k$.

Write n in NAF: $n = \sum_{i=0}^{d} b_i 2^i$ with
$b_i \in \{0, 1, -1\}$
$R \leftarrow 1$
for i from $d - 1$ downto 0 **do**
  $R \leftarrow R^2$
  if $b_i = 1$ **then**
    $R \leftarrow R \cdot x$
  else if $b_i = -1$ **then**
    $R \leftarrow R \cdot x^{-1}$
  end if
end for
return R

In Chapter 2, we investigate exponentiations where inversions are almost free so that the NAF is often used (called scalar multiplication in this context).

Exponentiation in base p. Exponentiations can be performed more efficiently using a base p decomposition together with the efficient Frobenius powers of Section 1.2.2. Suppose $x = \sum_{i=0}^{n} x_i p^i$. Then, a^x is computed as $a^x = \prod_{i=0}^{n} \left(a^{p^i}\right)^{x_i}$. If the prime p is special in the sense of Definition 1.3 (say $p = P(u)$), then a decomposition in base u leads to faster exponentiations.

We detail the cost of various exponentiations in Chapters 4 and 5, using special and non-special primes.

Cyclotomic subgroup exponentiation

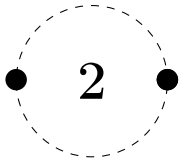
We go into detail on exponentiations in cyclotomic subgroup, following [GS10]. We now consider the subgroup $G_{\Phi_k(p)}$ of $\mathbb{F}_{p^k}^*$ of order $\Phi_k(p)$, where Φ_k is the k-th cyclotomic polynomial. We are interested in exponentiations in $G_{\Phi_k(p)}$. Granger and Scott obtain in [GS10, page 7] that a cyclotomic squaring in $\mathbb{F}_{p^k}$ where k is divisible by 6 costs $\mathbf{s}_k^{\text{cyclo}} = 3\mathbf{s}_{k/3}$ using the tower of extensions $\mathbb{F}_{q^{k/6}} \hookrightarrow \mathbb{F}_{q^{k/3}} \hookrightarrow \mathbb{F}_{q^k}$. Similarly, cyclotomic squarings apply in finite fields of even degree k. In [GS10, page 4], they obtain that $\mathbf{s}_k^{\text{cyclo}} = 2\mathbf{s}_{k/2}$. From that, an exponentiation of an element of $G_{\Phi_k(p)}$ to the power u costs $\mathbf{c}_u = (\log_2(u) - 1)\mathbf{s}_k^{\text{cyclo}} + (\text{HW}(u) - 1)\mathbf{m}_k$.

1.2.5 Summary

Table 1.2 summarises these estimated costs. We compared Table 1.2 with timings of the RELIC library [AG] for primes p of 6 to 8 machine-words and $k = 2, 6, 12$ on an Intel Core i5-4570 CPU, 3.20GHz. The accordance is satisfactory (within 10%), to the point that we use Table 1.2 as a base. Additionally, we also measured the relative costs of $\mathbf{i}$, $\mathbf{s}$, and $\mathbf{m}$ on the same platform, leading to $\mathbf{i} \approx 25\mathbf{m}$ and $\mathbf{s} \approx \mathbf{m}$. These approximations are clearly implementation-dependent.

k	1	2	3	5	6	7	8	12	16
$\mathbf{m}_k$	$\mathbf{m}$	$3\mathbf{m}$	$6\mathbf{m}$	$13\mathbf{m}$	$18\mathbf{m}$	$22\mathbf{m}$	$27\mathbf{m}$	$54\mathbf{m}$	$81\mathbf{m}$
$\mathbf{s}_k$	$\mathbf{m}$	$2\mathbf{m}$	$5\mathbf{m}$	$13\mathbf{m}$	$12\mathbf{m}$	$22\mathbf{m}$	$18\mathbf{m}$	$36\mathbf{m}$	$54\mathbf{m}$
$\mathbf{f}_k$	0	0	$2\mathbf{m}$	$4\mathbf{m}$	$4\mathbf{m}$	$6\mathbf{m}$	$6\mathbf{m}$	$10\mathbf{m}$	$14\mathbf{m}$
$\mathbf{s}_k^{\text{cyclo}}$					$6\mathbf{m}$		$12\mathbf{m}$	$18\mathbf{m}$	$36\mathbf{m}$
$\mathbf{i}_k - \mathbf{i}_1$	0	$4\mathbf{m}$	$12\mathbf{m}$	$48\mathbf{m}$	$34\mathbf{m}$	$104\mathbf{m}$	$44\mathbf{m}$	$94\mathbf{m}$	$134\mathbf{m}$
$\mathbf{i}_k$, with $\mathbf{i}_1 = 25\mathbf{m}$	$25\mathbf{m}$	$29\mathbf{m}$	$37\mathbf{m}$	$73\mathbf{m}$	$59\mathbf{m}$	$129\mathbf{m}$	$69\mathbf{m}$	$119\mathbf{m}$	$159\mathbf{m}$

Table 1.2: Relative cost of $\mathbf{m}_k$, $\mathbf{s}_k$ and $\mathbf{i}_k$ for finite field extensions of interest in Chapters 4 and 5.



Elliptic curves

In this section, we introduce elliptic curves and their application in cryptography. Points of an elliptic curve form a group for which the discrete logarithm problem is hard. The best known algorithms for solving the elliptic curve DLP have exponential complexity. Thus, this problem leads to many protocols such as the Elliptic Curve Diffie–Hellman (ECDH) key exchange and the Elliptic Curve Digital Signature Algorithm (ECDSA). Algebraic curves play an important role in cryptography and are widespread. More recently, particular maps between elliptic curves opened new doors for post-quantum cryptography. In order to understand these different protocols, we need to study the algebra behind these curves.

We introduce here the Weierstrass model representing an elliptic curve, together with formulas for computing the group law. From this group structure, we look at subgroups of the curve and morphisms of curves. In Chapter 3, the main tool will be particular morphisms called isogenies. We present here a few properties of these maps. We investigate further isogenies in Chapter 3. We also look at twists of curves, an important tool to obtain a gain of performance on the computation of pairings, the main subject of Chapter 4.

Finally, we give some details on the use of elliptic curves in applied cryptography. We introduce different models for representing the elliptic curve. These representations lead to different costs for the group law. The scalar multiplication, main step of elliptic-curve-based algorithms, has a variable cost depending on the model used. The last section of this chapter explains how to choose elliptic curve parameters from a cryptographic security point of view. The size of the parameters are important, but other properties matter, too. Twists of curves and small subgroups can lead to attacks on the discrete logarithm problem in some particular cases.

Summary

2.1 Definition

A general definition of an elliptic curve over a field K requires algebraic geometry theory.

Definition 2.1 (Elliptic curve)**.** *An elliptic curve over K is a smooth projective curve of genus 1 with a distinguished K-rational point.*

We will not need this abstract definition. The reader can look at [Har77] for a complete introduction to algebraic geometry. Beyond this definition, elliptic curves are varieties in the projective plane. Hence, an elliptic curve can be represented by an homogeneous polynomial equation in three variables. A point of the curve is a triple (X, Y, Z) satisfying the polynomial equation, and projective coordinates correspond to the class of points modulo the relation $(X, Y, Z) \sim (\lambda X, \lambda Y, \lambda Z)$ for all $\lambda \in K$. We now denote $(X : Y : Z)$ for the class of the point (X, Y, Z), and $\mathbb{P}^2(K)$ for the set of classes of points. Using substitutions, several polynomials lead to different models defining an elliptic curve with a polynomial equation. The usual model for introducing an elliptic curve is the Weierstrass model.

Definition 2.2 (Weierstrass model)**.** *An elliptic curve E over a field K can be defined by a non-singular equation of the form*

$$Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3 \tag{2.1}$$

where $a_1, \ldots, a_6 \in K$, together with a distinguished K-rational point.

We can represent the curve using an affine equation using the map $(X, Y, Z) \mapsto (X/Z, Y/Z, 1)$, plus an extra point $(0 : 1 : 0)$. The affine version of Equation (2.1) is

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6. \tag{2.2}$$

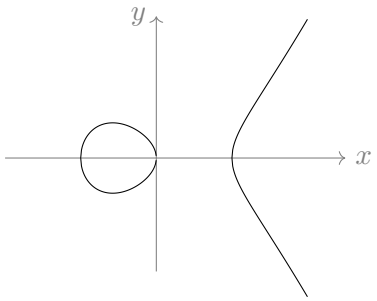


Figure 2.1: Affine part of the elliptic curve defined over $\mathbb{R}$ by $y^2 = x^3 - x$.

From now on, we consider elliptic curves defined over a field $\mathbb{F}_q$ of characteristic $p > 3$ (hence, q is a power of p). In such fields, 2 is invertible and the substitution $(x, y) \mapsto (x, (y - a_1x - a_3)/2)$ allows us to rewrite Equation (2.2) as

$$y^2 = 4x^3 + b_2x^2 + 2b_4x + b_6 \tag{2.3}$$

where

$$b_2 = a_1^2 + 4a_4 \qquad b_4 = 2a_4 + a_1a_3 \qquad b_6 = a_3^2 + 4a_6.$$

Finally, $3 \in \mathbb{F}_q$ is also invertible and the map $(x, y) \mapsto ((x - 3b_2)/36, y/108)$ allows us rewrite Equation (2.3) as a *short* Weierstrass equation:

$$y^2 = x^3 - 27c_4x - 54c_6 \tag{2.4}$$

where $c_4 = b_2^2 - 24b_4$ and $c_6 = -b_2^3 + 36b_2b_4 - 216b_6$.

Now, we consider elliptic curves in short Weierstrass form and note $E_{a,b}$ for the curve defined with $y^2 = x^3 + ax + b$. We define its j-invariant as $j(E_{a,b}) = 1728\frac{4a^3}{4a^3+27b^2}$. We will see at the end of Section 2.3 in which sense it is an invariant.

2.2 Group law

In this section, we present how to geometrically construct a group law on an elliptic curve. We now consider an elliptic curve in Weierstrass form $E : y^2 = x^3 + ax + b$ where $a, b \in \mathbb{F}_q$, together with the extra point $(0 : 1 : 0)$. We denote the geometrical curve $E := \{(x : y : z) \in \mathbb{P}^2(\bar{\mathbb{F}}_q), y^2z = x^3 + axz^2 + bz^3\}$ and for $k \in \mathbb{N}^*$, $E(\mathbb{F}_{q^k}) := \{(x : y : z) \in \mathbb{P}^2(\mathbb{F}_{q^k}), y^2z = x^3 + axz^2 + bz^3\}$. The Galois group $G(\bar{\mathbb{F}}_q/\mathbb{F}_{q^k})$ acts on E, and so $E(\mathbb{F}_{q^k})$ can be defined as the fixed points of E by the Galois group. We obtain a geometric group law using lines passing through points of the curve. Let P and Q be two points of E. The line passing through P and Q intersects the curve E in exactly three points (counted with multiplicity) by the Bézout theorem [ST15, §A.4]. The neutral element of our group law is by construction $0_E = (0 : 1 : 0)$. Now, we suppose that P and Q are affine points $(x_P, y_P, 1)$ and $(x_Q, y_Q, 1)$. If $(x_P, y_P) = (x_Q, -y_Q)$, then P and Q define a vertical line and its equation is simply $\ell_{P,Q}(x, y) = x - x_P$. Except this special case, the line (PQ) is not vertical, and its equation is given by

$$\ell_{P,Q}(x, y) = (y - y_P) + \lambda(x_P - x), \text{ where } \lambda = \begin{cases} \frac{y_Q - y_P}{x_Q - x_P} & \text{if } P \neq Q; \\ \frac{3x_P^2 + a}{2y_P} & \text{if } P = Q. \end{cases}$$

From this equation, and using the equation of the curve, we can explicitly determine the third point R of the intersection $E \cap (PQ)$:

$$\begin{aligned} x_R &= \lambda^2 - x_P - x_Q \\ y_R &= \lambda(x_P - x_R) - y_P. \end{aligned}$$

Now, we define a law of composition, written in additive notation: we define the point $P + Q$ to be the point $(x_R, -y_R)$, intersecting the vertical line $v_R(x, y) = x - x_R = 0$ with the curve E.

One can verify that for $k \in \mathbb{N}^*$, $E(\mathbb{F}_{q^k})$ form an abelian group. The tricky part is the associativity, that can be proved using symbolic computation. Another way is to obtain a bijection between the curve and a group and prove that it is actually a group isomorphism. The isomorphic group is the set of classes of degree 0 divisors modulo principal divisors. We introduce here the necessary definitions of the theory. We denote $\mathrm{Div}(E)$ the free abelian group generated by the points of E.

Definition 2.3 (Divisor)**.** *A divisor $D \in \mathrm{Div}(E)$ is a formal sum $D = \sum_{P \in E(\bar{\mathbb{F}}_q)} n_P(P)$, where $n_P \in \mathbb{Z}$ and $n_P = 0$ except for a finite set of points.*

The Galois group $G(\bar{\mathbb{F}}_q, \mathbb{F}_q)$ acts naturally on points and divisors. For $\sigma \in G(\bar{\mathbb{F}}_q, \mathbb{F}_q)$, $\sigma((x_P : y_P : z_P)) = (\sigma(x_P) : \sigma(y_P) : \sigma(z_P))$. With the notations of Definition 2.3,

$$\sigma(D) = \sum_{P \in E(\bar{\mathbb{F}}_q)} n_P(\sigma(P)).$$

We say that for $k \geq 1$, a divisor is *defined over* $\mathbb{F}_{q^k}$ if, and only if, $\sigma(D) = D$ for all $\sigma \in G(\bar{\mathbb{F}}_q, \mathbb{F}_{q^k})$.

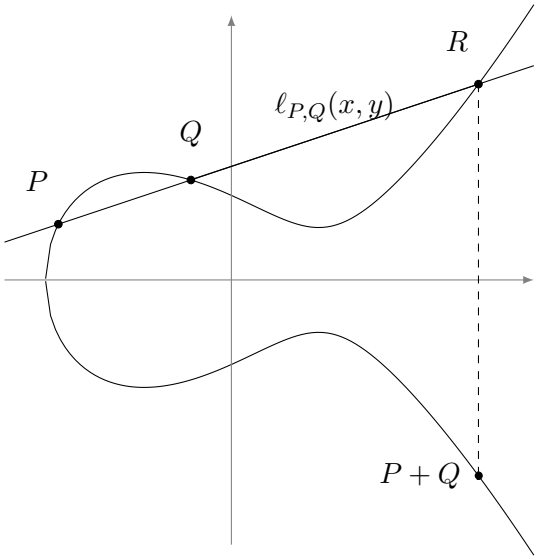


Figure 2.2: Geometric group law.

Definition 2.4 (Degree of a divisor)**.** *The degree of a divisor D is* $\deg D = \sum_{P \in E} n_P$.

We denote $\mathrm{Div}^0(E) = \{D \in \mathrm{Div}(E) / \deg D = 0\}$, and $\mathbb{F}_{q^k}(E)$ the set of rational functions over the curve E (as in [Sil86, page 3]). If $0 \neq f \in \mathbb{F}_{q^k}(E)$, we define $\mathrm{div}(f) := \sum_{P \in E(\bar{\mathbb{F}}_q)} \mathrm{ord}_P(f)(P)$, where $\mathrm{ord}_P(f)$ is the order of f at the point P, as defined in [Sil86, page 18].

Proposition 2.5. *Let $\ell_{P,Q}$ be a line as defined above. Then, by definition of the group law,* $\mathrm{div}(\ell_{P,Q}) = (P) + (Q) + (-(P+Q)) - 3(0_E)$. *Moreover,* $\mathrm{div}(\ell_{Q-P,P}) = \mathrm{div}(\ell_{P,-Q})$.

If $D = \sum_{P \in E(\bar{\mathbb{F}}_q)} n_P(P)$ is a divisor of degree 0 whose support is disjoint from the support of $\mathrm{div}(f)$, we define $f(D)$ to be $\prod_{P \in \bar{E}(\mathbb{F}_q)} f(P)^{n_P}$. A divisor D is principal if it is of the form $D = \mathrm{div}(g)$ for a non-zero $g \in \mathbb{F}_{q^k}(E)$. We obtain an equivalence relation

Definition 2.6. *Two divisors D_1 and D_2 are equivalent if $D_1 - D_2$ is a principal divisor.*

The Picard group $\mathrm{Pic}(E)$ is the quotient of $\mathrm{Div}(E)$ by the principal divisors. Finally, we denote $\mathrm{Pic}^0(E)$ the degree 0 part of the divisor class group of E. There is a group isomorphism between an elliptic curve E and $\mathrm{Pic}^0(E)$. This result is not obvious, and in our context, we only need the geometric construction. We refer to [Sil86, page 62] for details on this proof.

The opposite of a point P is $-P = (x_P, -y_P)$, and formulas are slightly different for a doubling step $[2]P$ or an addition $P + Q$:

$$\begin{aligned}
x_{[2]P} &= ((3x_P^2 + a)/(2y_P))^2 - 2x_P \\
y_{[2]P} &= 3x_P(3x_P^2 + a)/(2y_P) - ((3x_P^2 + a)/(2y_P))^3 - y_P \\
x_{P+Q} &= ((y_Q - y_P)/(x_Q - x_P))^2 - (x_Q + x_P) \\
y_{P+Q} &= (2x_P + x_Q)(y_Q - y_P)/(x_Q - x_P) - ((y_Q - y_P)/(x_Q - x_P))^3 - y_P.
\end{aligned}$$

We detail the cost of this algorithm and how to get more efficient formulas in Section 2.8.2.

2.3 Isogenies of elliptic curves

In this section, we introduce isogenies of elliptic curves, which are particular rational maps.

Definition 2.7 (Morphism)**.** *A morphism of curves is a rational map defined everywhere.*

In our context, every rational map of elliptic curves is a morphism. Over the algebraic closure, a rational map of elliptic curves is either constant or surjective.

Definition 2.8 (Isogeny)**.** *Let E and E' be two elliptic curves defined over $\mathbb{F}_q$ and $\mathbb{F}_{q^k}$ an extension of degree k. An isogeny $\phi : E \to E'$ is a non-constant rational map such that $\phi(0_E) = 0_{E'}$. An isogeny is said to be defined over $\mathbb{F}_{q^k}$ if the rational functions defining ϕ have coefficients in $\mathbb{F}_{q^k}$.*

Hence, over $\bar{\mathbb{F}}_q$, isogenies are surjective morphisms. One can prove that isogenies are actually group morphisms [Sil86, page 71]. If E and E' are elliptic curves in short Weierstrass form, an isogeny $\phi : E \to E'$ can be defined by an affine rational map of the form $\phi(x, y) = (u(x)/v(x), ys(x)/t(x))$, where $u, v, s, t \in \mathbb{F}_{q^k}[x]$ for an integer $k \geq 1$. An isogeny $\phi : E \to E'$ induces an injective morphism

$$\begin{array}{rccc} \phi^* : & \mathbb{F}_{q^k}(E') & \hookrightarrow & \mathbb{F}_{q^k}(E) \\ & f & \longmapsto & \phi \circ f. \end{array}$$

We obtain an extension field $\phi^*(\mathbb{F}_{q^k}(E')) \subset \mathbb{F}_{q^k}(E)$.

Definition 2.9 (Degree)**.** *The degree of an isogeny $\phi : E \to E'$ is the degree of the extension field $\mathbb{F}_{q^k}(E)/\phi^*(\mathbb{F}_{q^k}(E'))$. We also write ℓ-isogeny for an isogeny of degree ℓ.*

The structure of isogeny depends on the separability of the extension field $\phi^*(\mathbb{F}_{q^k}(E')) \subset \mathbb{F}_{q^k}(E)$.

Definition 2.10 (Separable, inseparable isogeny)**.** *An isogeny $\phi : E \to E'$ is separable if the extension $\mathbb{F}_{q^k}(E)/\phi^*(\mathbb{F}_{q^k}(E'))$ is separable, i.e. if every element in $\mathbb{F}_{q^k}(E)$ has a split minimal polynomial with simple roots.*
An isogeny is inseparable if it is not separable, and purely inseparable if the extension is.

Inseparable isogenies are closely related to the Frobenius isogeny, which is purely inseparable (recall that p is the characteristic of $\mathbb{F}_q$):

$$\begin{array}{rccc} \pi : & E_{a,b} & \longrightarrow & E_{a^p,b^p} \\ & (x, y) & \longmapsto & (x^p, y^p). \end{array}$$

An isogeny ϕ can be split into two isogenies: $\phi = \alpha \circ \pi^n$ where α is separable, and $n \geq 0$. The degree of a separable isogeny is simply the size of its kernel. Finally, given an isogeny $\phi : E \to E'$, there always exists a dual isogeny $\hat{\phi} : E' \to E$ such that $\phi \circ \hat{\phi} = [\deg(\phi)]$ [Sil86, Theorem 6.1].

Definition 2.11 (Isomorphism)**.** *An isogeny $\phi : E \to E'$ is an isomorphism if there exists an isogeny $\phi' : E' \to E$ such that $\phi \circ \phi' = \mathrm{Id}$.*

Isomorphisms are isogenies of degree one, and can be written of the form $(x, y) \mapsto (u^2x, u^3y)$ where $u \in \bar{\mathbb{F}}_q$. Two curves $E_{a,b}$ and $E_{a',b'}$ are isomorphic if, and only if, $a' = u^4a$ and $b' = u^6b$ for an element $u \in \bar{\mathbb{F}}_q$. The j-invariant defined at the end of Section 2.1 is an invariant of isomorphic curves: $j(E_{a,b}) = j(E_{u^4a,u^6b})$. Two curves are isomorphic over $\mathbb{F}_{q^k}$ if, and only if, the isomorphism is defined over $\mathbb{F}_{q^k}$ (i.e. $u^2, u^3 \in \mathbb{F}_{q^k}$).

2.4 Torsion

Given an elliptic curve $E_{a,b}$ (denoted E in this section) defined over $\mathbb{F}_q$ with $q = p^n$, we defined the Frobenius isogeny π. Composing n times this isogeny, we obtain $\pi^n : (x, y) \mapsto (x^q, y^q)$ which is an endomorphism because $E^{(p^n)} = E$. This endomorphism is often called the q-Frobenius. Its characteristic polynomial is $X^2 - tX + q$ and the trace t can be computed using a polynomial-time algorithm [Sch95]. The number of $\mathbb{F}_q$-rational points on the curve is related to the trace of the Frobenius [Sil86, page 141]:

$$\#E(\mathbb{F}_q) = q + 1 - t.$$

The Hasse bound [Sil86, page 138] shows that $|t| \leq 2\sqrt{q}$.

Given a point P of $E(\mathbb{F}_q)$, by the Lagrange theorem, $\ell = \text{ord}(P)$ divides $q + 1 - t$. The ℓ-torsion set, written $E[\ell]$, is the set of points of E (defined over the algebraic closure $\bar{\mathbb{F}}_q$) that satisfy $[\ell]P = 0_E$. We also denote $E(\mathbb{F}_{q^k})[\ell] := E[\ell] \cap E(\mathbb{F}_{q^k})$. The ℓ-th division polynomial of E is closely related to the ℓ-torsion. Division polynomials are defined recursively with the relations:

$$\psi_0 = 0 \qquad \psi_1 = 1 \qquad \psi_2 = 2y \qquad \psi_3 = 3x^4 + 6ax^2 + 12bx - a^2$$

$$\psi_4 = 4y(x^6 + 5ax^4 + 20bx^3 - 5a^2x^2 - 4abx - 8b^2 - a^3)$$

$$\begin{aligned} \psi_{2m+1} &= \psi_{m+2}\psi_m^3 - \psi_{m-1}\psi_{m+1}^3 && m \geq 2 \\ \psi_{2m} &= \left(\tfrac{\psi_m}{2y}\right)(\psi_{m+2}\psi_{m-1}^2 - \psi_{m-2}\psi_{m+1}^2) && m \geq 3 \end{aligned}$$

and indeed, if $\gcd(\ell, q) = 1$,

$$0_E \neq P \in E[\ell] \iff \begin{cases} \psi_\ell(x_P) = 0 & \ell \text{ is odd} \\ \psi_\ell(x_P)/y_P = 0 & \ell \text{ is even.} \end{cases}$$

Remark 2.12. $\psi_\ell \in \mathbb{Z}[x, a, b]$ (resp. $\mathbb{Z}[x, y, a, b]$) if ℓ is odd (resp. even). In practice, division polynomials can be computed recursively, but their degrees grow very fast so we can compute them only for small values of ℓ.

The ℓ-torsion set has a special structure:

$$E[\ell] = \begin{cases} \mathbb{Z}/\ell\mathbb{Z} \times \mathbb{Z}/\ell\mathbb{Z} & \text{if } \gcd(\ell, q) = 1 \\ \{0_E\} \text{ or } \mathbb{Z}/\ell\mathbb{Z} & \text{otherwise.} \end{cases}$$

Hence, ψ_ℓ has a number of roots which is $(\ell^2 - 1)/2$, $(\ell - 1)/2$ or 0 (given that P and $-P$ have the same x-coordinate). Looking at the torsion on the algebraic closure is interesting only from a theoretical point of view. In practice, when $\gcd(\ell, q) = 1$, we look at points on the smallest extension field such that the ℓ-torsion is fully rational. Usually, we call $\mathbb{G}_1$ and $\mathbb{G}_2$ two distinct subgroups of order ℓ defining the ℓ-torsion. A canonical choice can be done using the π^n endomorphism. Over $E[\ell]$, π^n is an endomorphism of the $\mathbb{Z}/\ell\mathbb{Z}$-module $E[\ell]$. Its eigenvalues are 1 and q. We can use the eigenspaces of π^n in order to make a choice of two subgroups of the ℓ-torsion: $\mathbb{G}_1 = E[\ell] \cap \ker(\pi^n - [1])$ and $\mathbb{G}_2 = E[\ell] \cap \ker(\pi^n - [q])$. In particular, with this choice, $\mathbb{G}_1$ is defined over $\mathbb{F}_q$ and $\mathbb{G}_2$ over $\mathbb{F}_{q^k}$ for an integer k which is called the *embedding degree*.

Definition 2.13 (Embedding degree)**.** *Let E be an elliptic curve defined over $\mathbb{F}_q$ and ℓ an integer coprime to q. The embedding degree of E with respect to ℓ is the smallest integer k such that $E(\mathbb{F}_{q^k})[\ell] = (\mathbb{Z}/\ell\mathbb{Z})^2$.*

Remark 2.14. We will give an equivalent definition in Chapter 4.

The structure of $E[p]$ allows us to split elliptic curves in two families: ordinary and supersingular curves.

Definition 2.15 (Ordinary and supersingular curves)**.** *An elliptic curve E over a field of characteristic p is ordinary if $E[p] = \mathbb{Z}/p\mathbb{Z}$ and supersingular if $E[p] = \{0_E\}$.*

Supersingular curves can always be defined over a quadratic finite field (i.e. $q = p^2$), and in this case, p divides the trace t. In particular, if E is a supersingular curve defined over a prime field $\mathbb{F}_p$, $\#E(\mathbb{F}_p) = p + 1$. One can prove that the embedding degree of a supersingular curve always divides 6. More precisely, over prime fields, supersingular elliptic curves have embedding degree 1, 2 or 3 [Gal05, page 199].

We now look at endomorphism rings of elliptic curves. The structure depends on the supersingularity of the curve.

2.5 Endomorphism rings of elliptic curves

Endomorphisms of elliptic curves are isogenies with same domain and codomain. In particular, given a curve E defined over $\mathbb{F}_q$, we have already seen an endomorphism: the q-Frobenius. The group law induces scalar multiplications endomorphisms $P \mapsto [n]P$. We develop the theory of scalar multiplication in Section 2.8.3. Hence, the lattice generated by [1] and π is included in $\mathrm{End}(E)$. Recall that π has characteristic polynomial $X^2 - tX + q$ so the lattice can be represented as $\mathbb{Z} + \mathbb{Z}\sqrt{-D}$ where D is the square-free part of $t^2 - 4q$. We now introduce orders of quadratic fields and quaternion algebras, closely related to the description of endomorphism rings of elliptic curves.

2.5.1 Orders in imaginary quadratic fields.

We introduce the needed tools for the theory of orders in imaginary quadratic fields. We refer to [Cox97] for a complete study of orders in number fields. Let K be a quadratic extension of $\mathbb{Q}$ defined with $x^2 + d$ for a positive square-free integer d. We define quadratic fields using their discriminant, always 0 or 1 modulo 4.

Definition 2.16 (Discriminant of K)**.** *The discriminant of K is*

$$\mathrm{Disc}(K) = \begin{cases} -d & \text{if } -d \equiv 1 \bmod 4 \\ -4d & \text{otherwise.} \end{cases}$$

To every discriminant corresponds an imaginary quadratic field. In the following, we consider a field of discriminant $-D$, and write $K = \mathbb{Q}(\sqrt{-D})$. Note that $\mathbb{Q}(\sqrt{-d})$ and $\mathbb{Q}(\sqrt{-D})$ are isomorphic even when $-D = -4d$: the isomorphism is simply $x + y\sqrt{-d} \mapsto x + y/2\sqrt{-D}$.

Orders are $\mathbb{Z}$-modules of K of rank 2 that are also subrings. Thus, we represent orders as a $\mathbb{Z}$-basis with two elements. As it is a subring of K, we often choose 1 as a basis vector, and write orders of the form $\mathbb{Z}[\alpha] = \{n + m\alpha, n, m \in \mathbb{Z}\}$ for a given α. The discriminant $\mathrm{Disc}(O)$ of an order O is the discriminant of the minimal polynomial of α. A common example of order is the ring of integers of K, denoted $\mathcal{O}_K$. In imaginary quadratic fields, it is of the form $\mathbb{Z}[\omega]$ where

$$\omega = \begin{cases} \sqrt{-d} & \text{if } -D \equiv 0 \bmod 4, \\ (1 + \sqrt{-d})/2 & \text{if } -D \equiv 1 \bmod 4. \end{cases}$$

The discriminant of the maximal order $\mathcal{O}_K$ is actually the discriminant $-D$ of the field: depending on the congruence class of d mod 4, ω has minimal polynomial $x^2 + d$ or $x^2 - x + (1 + d)/4$, so $\mathcal{O}_K$ has discriminant $-4d$ or $-d$, which is $\mathrm{Disc}(K)$. The ring of integers is actually the unique maximal order of K: every order is contained in $\mathcal{O}_K$. Thus, every order O of K is of the form $\mathbb{Z}[f\omega]$ for an integer f which satisfies $f = [\mathcal{O}_K : O]$.

Definition 2.17 (Conductor)**.** *Let $O_2 \subset O_1$ be two orders in K. The index $[O_1 : O_2]$ is called the conductor of O_2 in O_1.*

Remark 2.18. The discriminants of O_1 and O_2 satisfy $\mathrm{Disc}(O_2) = f^2 \mathrm{Disc}(O_1)$. If $O = \mathbb{Z}[f\omega]$ as above, then f is the conductor of O in $\mathcal{O}_K$.

2.5.2 Maximal orders in quaternion algebras

A quaternion algebra is a non-commutative $\mathbb{Q}$-algebra of dimension 4 defined with four generators: $H_{-a,-b} = \mathbb{Q}\mathbf{1} + \mathbb{Q}\mathbf{i} + \mathbb{Q}\mathbf{j} + \mathbb{Q}\mathbf{k}$ where $\mathbf{i}^2 + a = 0$, $\mathbf{j}^2 + b = 0$ and $\mathbf{k} = \mathbf{ij} = -\mathbf{ji}$. As in the case of quadratic fields, orders are full rank lattices that are also subrings of $H_{-a,-b}$, but maximal orders are not unique. In our context, we consider a quaternion algebra ramified only at p and ∞. It means that $H_{-a,-b} \otimes \mathbb{Q}_p$ and $H_{-a,-b} \otimes \mathbb{R}$ are division algebras. See [Voi20, Section 14.1] for details. Depending on the class of p mod 8, Pizer gives in [Piz80, page 368] a couple of integers (a, b) such that the algebra $H_{-a,-b}$ satisfies these properties of ramification. In particular,

- If $p = 3$ mod 4, $(a, b) = (1, p)$.
- If $p = 5$ mod 8, $(a, b) = (2, p)$.
- If $p = 1$ mod 8, $(a, b) = (r, p)$ where $r = 3$ mod 4 and $(p/r) = -1$. Assuming that the generalized Riemann hypothesis is true, there exists $r = O(\log_2^2(p))$ satisfying these conditions.

We will give some details on orders of quaternion algebra in Chapter 6. We refer to [Voi20] for a complete study of the theory of quaternion algebras.

2.5.3 Structure of endomorphism rings of elliptic curves

The structure of $\mathrm{End}(E)$ is closely related to the supersingularity of E.

Theorem 2.19 (Endomorphism ring structure [Sil86, page 102])**.** *Let E be an elliptic curve defined over $\mathbb{F}_q$ where q is a power of p. Then,*

- *If E is ordinary, then $\mathrm{End}(E)$ is an order in the imaginary quadratic field $\mathbb{Q}(\sqrt{t^2 - 4q})$.*
- *If E is supersingular, then $\mathrm{End}(E)$ is a maximal order in the quaternion algebra $H_{-a,-b}$ as above.*

Example 2.20. Let $E : y^2 = x^3 - x$ defined over a prime field $\mathbb{F}_p$. Then,

- If $p = 3$ mod 4, then -1 is not a square over $\mathbb{F}_p$, and assuming $\mathbb{F}_{p^2} = \mathbb{F}_p[x]/(x^2 + 1) = \mathbb{F}_p(i)$, $(x, y) \mapsto (-x, iy)$ is an endomorphism that anticommutes with the Frobenius π. E is supersingular and $\mathrm{End}(E)$ is a maximal order in a quaternion algebra.
- If $p = 1$ mod 4, the latter endomorphism now commutes with the Frobenius π. One can prove (by computing the trace for example) that the curve is ordinary in this case of congruence of p.

We go into detail on examples of endomorphism rings of ordinary curves in Section 3.1.1 and study the supersingular case in Chapter 6.

2.5.4 The Complex Multiplication (CM) method

Over a field of characteristic 0, the endomorphism ring is commutative. Hence, over $\mathbb{Q}$, an elliptic curve has an endomorphism ring which is either $\mathbb{Z}$ or an order of an imaginary quadratic field. It is possible to generate an elliptic curve over a number field of given endomorphism ring (specified by an order $\mathcal{O}_D$ of discriminant D) thanks to the CM method, closely related to the Hilbert class polynomial. We denote $\mathrm{Ell}(\mathcal{O})$ to be the set of isomorphism classes of elliptic curves with endomorphism ring the order $\mathcal{O}$.

Definition 2.21 (Hilbert class polynomial)**.** *The Hilbert class polynomial is the polynomial* $H_D(X) = \prod_{E \in \mathrm{Ell}(\mathcal{O}_D)}(X - j(E))$ *where* $\mathcal{O}_D$ *is the order of discriminant* $-D$*.*

Hilbert class polynomials are irreducible and have integer coefficients. Their study is related to the class group theory. We will give some details of this theory in Chapter 3. This polynomial can be computed modulo a prime p with space complexity $O(|D|^{1/2+\epsilon}\log_2(p))$ and $O(|D|^{1+\epsilon})$ time complexity using an algorithm presented in [Sut10]. The algorithm has been improved in [ES10] and is practical up to $D \approx 10^{16}$.

From the Hilbert class polynomial H_D, each root j_0 is a j-invariant of an elliptic curve over a number field of given endomorphism ring (of discriminant $-D$). Then, the elliptic curve equation can be recovered easily from j_0: if $j_0 = 0$ (resp. 1728), the curve $E : y^2 = x^3 + 1$ (resp. $E : y^2 = x^3 + x$) satisfy $j(E) = j_0$. Otherwise, one verifies that Equation (2.5) defines an elliptic curve of j-invariant j_0:

$$y^2 = x^3 + 3j_0x/(1728 - j_0) + 2j_0/(1728 - j_0). \tag{2.5}$$

We obtain an elliptic curve defined over $\mathbb{F}_p$ by reducing a curve defined over a number field modulo a prime p. The reduced curve is ordinary (resp. supersingular) if, and only if p splits (resp. is inert) in the order of discriminant $-D$. The case where p ramifies does not occur here as $p \gg D$. Inert and splitting primes are in the same proportion. Thus, reduced curves are ordinary for half of the primes.

In practice, the CM method produces particular curves with a small discriminant (their computation needs the knowledge of H_D). The question of generating an elliptic curve with a random (large discriminant) endomorphism ring will be detailed in Section 3.3.4. The CM method will be useful for generating ordinary elliptic curves in Chapter 4.

2.6 Automorphisms of elliptic curves

Automorphisms of elliptic curves are endomorphisms of degree 1. Using the bijection of Theorem 2.19, automorphisms are units of a common mathematical object representing the endomorphism ring. In imaginary quadratic number fields, a unit is a root of a cyclotomic polynomial. Hence, we look at cyclotomic polynomials of degree dividing 2: $T - 1$, $T + 1$, $T^2 + T + 1$, $T^2 + 1$ and $T^2 - T + 1$. Finally the units group is always a cyclic group of order 2, 4 or 6:

- If $D = 4$, integer units are $\{\pm 1, \pm i\} \simeq \mathbb{Z}/4\mathbb{Z}$ where $i = \sqrt{-1}$.
- If $D = 3$, integer units are $\{\pm 1, \pm j, \pm j^2\} \simeq \mathbb{Z}/6\mathbb{Z}$ where $j = (1 + \sqrt{-3})/2$.

- If $D \notin \{1, 3\}$, units are $\{\pm 1\} \simeq \mathbb{Z}/2\mathbb{Z}$.

A general study of units of quaternion algebras is given in [Voi20, Chapter 32]. The main difference with imaginary quadratic fields is the case of $H_{-1,-1}$ and $H_{-1,-3}$, where units are non-commutative subgroups of $SO_3(\mathbb{Q})$. In our context, endomorphism rings are maximal orders of quaternion algebras of the form $H_{-r,-p}$ where $r = 1$, 2 or $O(\log_2^2(p))$ and p is large. In these cases, the structure of units is the same as in the case of imaginary quadratic fields: $\mathbb{Z}/2\mathbb{Z}$, $\mathbb{Z}/4\mathbb{Z}$ or $\mathbb{Z}/6\mathbb{Z}$. Finally, an elliptic curve has only few automorphisms, and the structure of the automorphism group described in Theorem 2.22 lets us describe particular isomorphic curves, called twists.

Theorem 2.22 (Automorphisms of elliptic curves)**.** *Let E be an elliptic curve over $\mathbb{F}_q$ where q is a power of a prime $p > 3$. Then,* $\mathrm{Aut}(E) \simeq \mathbb{Z}/n\mathbb{Z}$ *where $n = 6$ if $j(E) = 0$, $n = 4$ if $j(E) = 1728$ and $n = 2$ if $j(E) \neq 0, 1728$.*

2.7 Twists of curves

Twists are particular isomorphisms of elliptic curves. Twisted curves arise naturally when we look at points of a curve $E_{a,b}$ defined over $\mathbb{F}_q$. Given $x \in \mathbb{F}_q$, it corresponds to the abscissa of a point of $E_{a,b}$ with probability $1/2 + 1/(q-1)$. When it is not the case, $(x^3 + ax + b)\sqrt{v}$ gives rise to a point of $E_{a,b}$ for any non-square $v \in \mathbb{F}_q^*$ so that we naturally have a map

$$\begin{array}{rcl} E_{a,b} & \longmapsto & E^t_{a,b} : vy^2 = x^3 + ax + b \\ (x, y) & \longmapsto & (x, \sqrt{v}y) \end{array}$$

which is an isomorphism defined over $\mathbb{F}_{q^2} = \mathbb{F}_q[x]/(x^2 - v)$. As we have seen at the end of Section 2.3, isomorphisms can be written in the form $(x, y) \mapsto (u^2x, u^3y)$ for $u \in \bar{\mathbb{F}}_q$. In our case, we can choose $u = \sqrt{v}$ and we obtain

$$\begin{array}{rcl} E_{a,b} & \longmapsto & E^t_{a,b} = E_{v^2a,v^3b} \\ (x, y) & \longmapsto & (\sqrt{v}^2x, \sqrt{v}^3y). \end{array}$$

The curve $E^t_{a,b}$ is called the *quadratic twist* of $E_{a,b}$.

Definition 2.23 (Twists of an elliptic curve)**.** *A twist of an elliptic curve E defined over $\mathbb{F}_q$ is an elliptic curve E', also defined over $\mathbb{F}_q$, such that E and E' are isomorphic over $\bar{\mathbb{F}}_q$. Two twists are equivalent if they are isomorphic over $\mathbb{F}_q$. We say that the twist is of degree d if the minimal extension defining the ismorphism is $\mathbb{F}_{q^d}$.*

From the structure of the automorphism group, [Sil86, page 343] obtains that there are only twists of degree dividing 6.

Theorem 2.24. *Let $E_{a,b}$ be an elliptic curve defined over $\mathbb{F}_q$ of characteristic $p > 3$. Then,*

- *If $j(E) \neq 0, 1728$, E has only one twist of degree 2 called the quadratic twist. If $v \in \mathbb{F}_q^*$ is a non-square, the isomorphism is given by $(x, y) \mapsto (vx, \sqrt{v}^3y)$.*
- *If $j(E) = 1728$, then E has one quadratic twist and two twists of degree 4, called quartic twists. If $w \in \mathbb{F}_q^*$ is such that $x^4 - w$ is irreducible, then we define $\mathbb{F}_{q^4} = \mathbb{F}_q(\sqrt[4]{w})$. Thus, one quartic twist is defined by $(x, y) \mapsto (\sqrt{w}x, \sqrt[4]{w}^3y)$ and the quadratic twist is defined by $(x, y) \mapsto (wx, \sqrt{w}^3y)$.*

- *If $j(E) = 0$, then E has two degree 6 twists, called sextic twists, two twists of degree 3 and one quadratic twist. Given $z \in \mathbb{F}_q^*$ such that $x^6 - z$ is irreducible, then a sextic twist is defined by $(x, y) \mapsto (\sqrt[3]{z}x, \sqrt{z}y)$.*

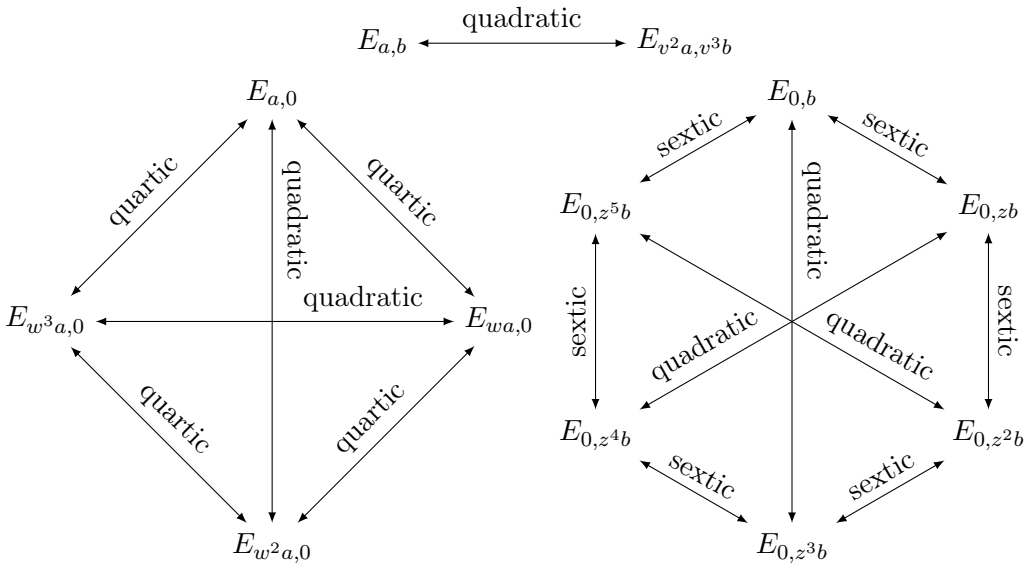


Figure 2.3: Twists of elliptic curves in the different cases of Theorem 2.24.

Remark 2.25. Twists of an elliptic curve defined over $\mathbb{F}_q$ are all defined over $\mathbb{F}_q$, but the isomorphism map is defined over $\mathbb{F}_{q^d}$ (where $d = 2$, 4 or 6).

From this construction of the quadratic twist at the begining of this section, we can see that each $x \in \mathbb{F}_q$ always gives two points of the set $\{E_{a,b}(\mathbb{F}_q), E^t_{a,b}(\mathbb{F}_q)\}$. Finally, $\#E_{a,b}(\mathbb{F}_q)$ and $\#E^t_{a,b}(\mathbb{F}_q)$ are closely related, and if t is the trace of the Frobenius as in Section 2.4, then

$$\#E_{a,b}(\mathbb{F}_q) = q + 1 - t \qquad \#E^t_{a,b}(\mathbb{F}_q) = q + 1 + t.$$

A similar approach lets us compute the trace of quartic and sextic twisted curves.

- If $E_{a,b}$ has j-invariant 1728, then $b = 0$ and the curve has quartic twists.

  $$\#E_{a,0}(\mathbb{F}_q) = q + 1 - t \qquad \#E_{wa,0}(\mathbb{F}_q) = q + 1 + 2y$$

  $$\#E_{w^2a,0}(\mathbb{F}_q) = q + 1 + t \qquad \#E_{w^3a,0}(\mathbb{F}_q) = q + 1 - 2y$$

  where y satisfies $t^2 - 4q = -4y^2$.

- If $E_{a,b}$ has j-invariant 0, then $a = 0$ and the curve has sextic twists.

  $$\#E_{0,b}(\mathbb{F}_q) = q+1-t \qquad \#E_{0,zb}(\mathbb{F}_q) = q+1-(-3y+t)/2 \qquad \#E_{0,z^2b} = q+1-(3y-t)/2$$

  $$\#E_{0,z^3b}(\mathbb{F}_q) = q+1+t \qquad \#E_{0,z^4b}(\mathbb{F}_q) = q+1-(-3y-t)/2 \qquad \#E_{0,z^5b} = q+1-(3y+t)/2$$

  where y satisfies $t^2 - 4q = -3y^2$.

Compression of the $\mathbb{G}_2$ elements

Twists can be used to represent elements of an elliptic curve in a compressed form. We refer to [Ver10] for details. Suppose that E is an elliptic curve defined over $\mathbb{F}_q$, with embedding degree $k = dk'$ with $d = 2$, 4 or 6 according to Theorem 2.24. Then, defining E over $\mathbb{F}_{q^{k/d}}$, E has a twist of degree d:

$$\begin{array}{rccc} \tau_d : & E & \longrightarrow & E' \\ & (x, y) & \longmapsto & (x\alpha^2, y\alpha^3) \end{array}$$

where $\alpha \in \mathbb{F}_{q^k}$ is a root of the irreducible polynomial $X^d - u \in \mathbb{F}_{q^{k/d}}[X]$ (for a given $u \in \mathbb{F}_{q^{k/d}}$). Thus, $E(\mathbb{F}_{q^k}) \simeq E'(\mathbb{F}_{q^k})$ and the subgroup $\mathbb{G}_2$, defined over $\mathbb{F}_{q^k}$, can be seen as $\tau_d^{-1}(\mathbb{G}_2')$ where $\mathbb{G}_2'$ is a subgroup of $E'(\mathbb{F}_{q^{k/d}})$. Hence, the coordinates of a point of $\mathbb{G}_2$ are represented using only k/d elements of $\mathbb{F}_q$.

Weierstrass representation with $a = -3$

As we will see in Section 2.8.2, elliptic curve cryptography can be more efficient when the elliptic curve is defined with the Weierstrass equation $y^2 = x^3 - 3x + c$. Suppose that $E_{a,b}$ is a an elliptic curve defined with a general equation $y^2 = x^3 + ax + b$ over $\mathbb{F}_q$.

- If $-3/a$ is a 4-th power in $\mathbb{F}_q$ (say $-3 = au^4$), then the $\mathbb{F}_q$-isomorphism $(x, y) \mapsto (xu^2, yu^3)$ allows us to choose the isomorphic elliptic curve $E_{-3,bu^6} : y^2 = x^3 - 3x + bu^6$. These two curves have the same torsion structure and twists structure.

- If $-3/a$ is not a 4-th power in $\mathbb{F}_q$, then we can use the quadratic twist in order to get a similar property. By construction of the quadratic twist, let v be a non-square in $\mathbb{F}_q$. Then, $E_{a,b}^t : y^2 = x^3 + av^2x + bv^3$ and v^2 is not a 4-th power (otherwise, v would be a square). Finally, $-3/(av^2)$ is a 4-th power and the property applies to $E_{a,b}^t$. Going back with the quadratic twist, $E_{a,b}$ is isomorphic to a curve of Weierstrass equation $y^2 = x^3 - 3w^2x + cw^3$, where $w \in \mathbb{F}_q$ is a non-square[1].

2.8 Elliptic curves in cryptography

In this section, we consider the discrete logarithm problem in elliptic-curve-based cryptography. We introduce efficient algorithms for computing the group law, and explain how insecure a curve can be when its structure has specificites. We will reuse the concepts introduced here in Chapter 4, when we look for new elliptic curves for pairing-based cryptography.

2.8.1 Discrete logarithm over an elliptic curve

As we have seen in Section 2.2, rational points of an elliptic curve form a group and we can consider the discrete logarithm problem on this group. Given a point $P \in E(\mathbb{F}_q)$ and $[s]P$, find the integer s. This problem is considered hard and the best algorithms breaking the DLP are actually generic algorithms. The baby-step–giant-step algorithm of Section 1.1.1 has a complexity $O(\sqrt{\#G})$ where G is the group used for DLP. This is in fact the best known complexity to date for solving the DLP. In our context, the group is not $E(\mathbb{F}_q)$, but the subgroup generated by P. Hence, in order to use the DLP over an elliptic curve, we need a large *prime* order subgroup. For a 128-bit security level, we will use an elliptic curve E defined over $\mathbb{F}_q$ such that $\#E(\mathbb{F}_q)$ has a

[1] w can be chosen small.

large prime factor $\approx 2^{256}$. From the Hasse bound described in Section 2.4, $\#E(\mathbb{F}_q) \approx q$ and so we look for $q \approx 2^{256}$ in order to obtain a safe curve.

In cryptographic applications, the security will be based on the DLP and so we often compute a scalar multiplication $[n]P = P + \ldots + P$ (n times) for a large (256 bits) integer n. Given a point $P = (x_P, y_P)$ in affine Weierstrass coordinates, the point $[n]P$ is determined by Equation (2.6), where ψ_n is the n-th division polynomial of E defined in Section 2.4.

$$[n]P = \left(x - \frac{\psi_{n-1}\psi_{n+1}}{\psi_n(x_P, y_P)^2}, \frac{\psi_{2n}(x_P, y_P)}{2\psi_n(x_P, y_P)^4} \right). \tag{2.6}$$

When we want to compute large scalar multiplications on the curve, we do not use these polynomials: as we have seen, their degrees grow very fast. Instead, we use Algorithm 2.1 that is an analogue of the square-and-multiply algorithm (called double-and-add in additive groups).

Algorithm 2.1: Scalar multiplication(n, P)

Input. n and integer,
P a point of $E(\mathbb{F}_q)$.
Output. $[n]P$.
Write $n = \sum_{i=0}^{d} b_i 2^i$ with $b_i \in \{0, 1\}$
$R \leftarrow P$
for i from $d-1$ downto 0 **do**
  $R \leftarrow \texttt{Dbl}(R)$
  if $b_i = 1$ **then**
    $R \leftarrow \texttt{Add}(R, P)$
  end if
end for
return R

This algorithm computes doubling (`Dbl`) and addition (`Add`) steps, and so we need an efficient group law in order to efficiently compute scalar multiplications. Using substitutions, many equations can represent an elliptic curve in the same way as the Weierstrass equation. Thus, using different models of elliptic curve, the group law is computed differently.

2.8.2 Formulas in different models

In Section 2.2, we have given formulas for computing the group law. In order to get more efficient addition and doubling steps, we can rewrite formulas in projective coordinates so that no inversion is needed. In Algorithms 2.2 and 2.3, a doubling step (resp. an addition step) costs 11 (resp. 14) multiplications (assuming $1\mathbf{s} = 1\mathbf{m}$). These algorithms are provided in [BL08].

Algorithm 2.2: Addition step in Weierstrass coordinates

Input. `P = (X1,Y1,Z1)`,
`Q = (X2,Y2,Z2)`.
Output. `P+Q = (X3,Y3,Z3)`.
Cost. 2**s** and 12**m**.

```
Y1Z2 = Y1*Z2
X1Z2 = X1*Z2
Z1Z2 = Z1*Z2
u = Y2*Z1-Y1Z2
uu = u**2
v = X2*Z1-X1Z2
vv = v**2
vvv = v*vv
R = vv*X1Z2
A = uu*Z1Z2-vvv-2*R
X3 = v*A
Y3 = u*(R-A)-vvv*Y1Z2
Z3 = vvv*Z1Z2
```

Algorithm 2.3: Doubling step in Weierstrass coordinates

Input. `P = (X1,Y1,Z1)`.
Output. `[2]P = (X3,Y3,Z3)`.
Cost. 6**s** and 5**m**.

```
XX = X1**2
ZZ = Z1**2
w = a*ZZ+3*XX
s = 2*Y1*Z1
ss = s**2
sss = s*ss
R = Y1*s
RR = R**2
B = (X1+R)**2-XX-RR
h = w**2-2*B
X3 = h*s
Y3 = w*(B-h)-2*RR
Z3 = sss
```

In Algorithm 2.1, the computation uses more doubling steps than additions. Hence, we can use a change of variables in order to get a faster doubling step. We present here the modified Jacobian and Montgomery coordinates. More models and their detailed group law costs are presented in [BL08] and at `https://hyperelliptic.org/`.

Definition 2.26 (Projective Jacobian model)**.** *The elliptic curve $E_{a,b}$ can be defined with a projective Jacobian equation $Y^2 = X^3 + aXZ^4 + bZ^6$ using the substitution from the short Weierstrass model $(x, y) \mapsto (xz^2, yz^3, z)$.*

Using this model, a doubling step (resp. addition step) now costs only 9 (resp. 16) multiplications. The *modified* Jacobian coordinates [CMO98] are simply a tweak where a fourth coordinate corresponds to the square of the Z-coordinate. Using this modification, addition costs two additional multiplications, but one multiplication is saved for the doubling step.

Remark 2.27. When the curve has a Weierstrass equation of the form $E_{-3\cdot u^2, b\cdot u^3}$ for a small u, a trick allows us to save one multiplication in Algorithm 2.3.

Algorithm 2.4: Addition step in modified Jacobian coordinates

Input. `P = (X1,Y1,Z1,T1)`, `Q = (X2,Y2,Z2,T2)`.
Output. `P+Q = (X3,Y3,Z3,T3)`.
Cost. 7**s** and 11**m**.

```
ZZ1 = Z1**2
ZZ2 = Z2**2
U1 = X1*ZZ2
U2 = X2*ZZ1
S1 = Y1*Z2*ZZ2
S2 = Y2*Z1*ZZ1
H = U2-U1
I = (2*H)**2
J = H*I
r = 2*(S2-S1)
V = U1*I
X3 = r**2-J-2*V
Y3 = r*(V-X3)-2*S1*J
Z3 = ((Z1+Z2)**2-ZZ1-ZZ2)*H
ZZ3 = Z3**2
T3 = a*ZZ3**2
```

Algorithm 2.5: Doubling step in modified Jacobian coordinates

Input. `P = (X1,Y1,Z1,T1)`.
Output. `[2]P = (X3,Y3,Z3,T3)`.
Cost. 5**s** and 3**m**.

```
XX = X1**2
A = 2*Y1**2
AA = A**2
U = 2*AA
S = (X1+A)**2-XX-AA
M = 3*XX+T1
X3 = M**2-2*S
Y3 = M*(S-X3)-U
Z3 = 2*Y1*Z1
T3 = 2*U*T1
```

Definition 2.28 (Montgomery model [CS18])**.** *Let $E_{a,b}$ be an elliptic curve defined over a finite field $\mathbb{F}_q$. If $\#E_{a,b}(\mathbb{F}_q) = 0 \bmod 4$ and $x^3 + ax + b$ has a root $\alpha \in \mathbb{F}_q$ such that $3\alpha^2 + a$ is a square in $\mathbb{F}_q$, then $E_{a,b}$ can be defined with a Montgomery equation*

$$By^2 = x^3 + Ax^2 + x,$$

where $B = 1/\sqrt{3\alpha^2 + a}$ and $A = 3\alpha B$, using the substitution $(x, y) \mapsto (B(x - \alpha), By)$.

In this model, the formula to compute the x-coordinate of $[2]P$ does not depend on y_P, and the x-coordinate of $P + Q$ can be computed from x_P, x_Q and x_{P-Q}:

$$x_{[2]P} = \frac{(x_P^2 - 1)^2}{4x_P(x_P^2 + Ax_P + 1)} \qquad x_{P+Q}x_{P-Q}(x_P - x_Q)^2 = (x_P x_Q - 1)^2.$$

We obtain *partial* doubling and differential addition steps in the sense that only the x-coordinate is computed, and the addition step needs x_{P-Q} as input. We will see that these partial computations let us compute efficiently a scalar multiplication. The cost of the partial doubling step (resp. partial addition step) is 4 multiplications (resp. 6 multiplications) using projective coordinates. As we compute only the x and z-coordinates of points in projective model, we use the notation $\mathbf{x}(P) = (x_P, z_P)$.

Algorithm 2.6: xAdd differential addition step in Montgomery coordinates

Input. $\mathbf{x}(P-Q)$ =(X1,Z1),
$\mathbf{x}(P)$ =(X2,Z2),
$\mathbf{x}(Q)$ =(X3,Z3).
Output. $\mathbf{x}(P+Q)$ =(X5,Z5).
Cost. 2**s** and 4**m**.

```
A = X2+Z2
B = X2-Z2
C = X3+Z3
D = X3-Z3
DA = D*A
CB = C*B
X5 = Z1*(DA+CB)**2
Z5 = X1*(DA-CB)**2
```

Algorithm 2.7: xDbl doubling step in Montgomery coordinates

Input. $\mathbf{x}(P)$ =(X1,Z1).
Output. $\mathbf{x}([2]P)$ =(X3,Z3).
Cost. 2**s** and 2**m**.

```
A = X1+Z1
AA = A**2
B = X1-Z1
BB = B**2
C = AA-BB
X3 = AA*BB
Z3 = C*(BB+(a+2)*C/4)
```

In order to obtain the entire point, an extra square-root computation is needed to get the y-coordinate. Most of the time, the x-coordinate is sufficient and we compute the y-coordinate only few times, without any square-root. More details will be given at the end of Section 2.8.3. We summarize in Table 2.1 the costs in terms of finite field multiplications. We use the same notations as in Chapter 1.

Coordinates	Doubling step	Alg.	Addition step	Alg.
Weierstrass projective	9**m**	2.2	16**m**	2.3
Modified Jacobian projective	8**m**	2.4	18**m**	2.5
Montgomery projective (x-only)	4**m**	2.6	6**m**	2.7

Table 2.1: Cost of the group law in different models of elliptic curves, from [BL08]

2.8.3 Scalar multiplication

As explained in Section 2.8.1, the main algorithm in elliptic curve cryptography is the scalar multiplication. Algorithm 2.1 is very similar to a square-and-multiply algorithm, and has a complexity linear in the size of the scalar ($O(\log_2(n))$). Writing $n = b_d 2^d + \ldots + b_1 2 + b_0$ in binary representation, Algorithm 2.1 reads the bits b_i of n, computes a doubling step, and if $b_i = 1$, an addition step. Hence, if we denote $\mathrm{HW}(n)$ the number of 1 in the binary representation of n, Algorithm 2.1 costs $(\log_2(n) - 1)$ doubling steps and $(\mathrm{HW}(n) - 1)/2$ addition steps in average. Asymptotically, Algorithm 2.1 has complexity $O(\log_2(n))$ curve operations. In practice the cost of the scalar multiplication can be improved by a constant factor.

Signed binary representation of n. Changing the representation basis of n can improve Algorithm 2.1. Representing n in base b for $b > 2$ does not improve the concrete complexity because the *multiplication by b* step (computed $\log_b(n) - 1$ times) becomes expensive when b grows. Similarly to the exponentiations of Section 1.2.4, the NAF representation of Definition 1.5 lets us obtain more zeros than in the binary representation (and hence less addition steps). Note that the addition step is now either an addition, either a subtraction, but a subtraction has the

same cost as an addition. In average, a scalar multiplication with the non-adjacent form needs $\log_2(n)$ doublings and $\log_2(n)/3$ additions/subtractions. For instance, to compute $[2^d - 1]P$, the NAF lets us compute the scalar multiplication in $d-1$ doublings and 1 subtraction instead of $d-2$ doublings and $d-2$ additions with a binary representation.

Sliding window. During a scalar multiplication $[n]P$, the scalar n is read bit by bit. The idea of the sliding window [HMV03, page 99] is to read these bits by blocks of w. This way, we compute the scalar multiplication using $\log_2(n)/w$ squarings and $3\log_2(n)/w$ multiplications (in average). This method needs the precomputations of $\{[i]P, 2 \leq i \leq 2^{w-1} - 1\}$. Depending on the context, this method can be efficient, for example if the point P is fixed.

GLV method. Another improvement of the scalar multiplication is the GLV method [GLV01]. Given a low degree endomorphism of the curve ψ with eigenvalue λ, the computation of $[n]P$ can be decomposed into $[n_1]P + [n_2]\psi(P)$, where n_1 and n_2 are determined by a (dimension 2) lattice reduction, writing the vector $(n, 0)$ in the basis of the lattice $\mathbb{Z} + \mathbb{Z}\lambda$. Finally, as ψ has a low degree, its evaluation on P is efficient. Using the lattice reduction, the scalars n_1 and n_2 are of the size of $\sqrt{n}$ and the two scalar multiplications $[n_1]P$ and $[n_2]\psi(P)$ can be computed in parallel. Once the precomputation of $\psi(P)$ and $\psi(P) + P$ is done, the GLV method leads to a reduction by a factor 2 of the cost of the scalar multiplication. In particular, this algorithm is efficient for curves of j-invariant 0 and 1728, for which a very fast endomorphism is known. The GLV method can be generalized into a four-dimensional GLV on $\mathbb{Q}$-curves [Smi16, LS14]. Using a particular prime p, the finite field arithmetic can be improved, leading to a very efficient group law [CL15, BDM18].

The case of Montgomery coordinates

Algorithm 2.1 computes `Dbl` and `Add` steps. Hence, it can be computed in Weierstrass or Jacobian coordinates, but not with the x-only arithmetic in Montgomery coordinates. In this case, we use Algorithm 2.8 to compute the x-coordinate of $[n]P$.

Algorithm 2.8: x-only scalar multiplication(n, P)

Input. n an integer,
$P = (x_P, y_P)$.
Output. $[n]P = (x_0, y_0)$.

Write $n = \sum_{i=0}^{d} b_i 2^i$ with $b_i \in \{0, 1\}$
$(x_0, x_1) \leftarrow (x_P, \text{xDbl}(x_P))$
for i from $d-1$ downto 0 **do**
  if $b_i = 1$ **then**
    $(x_0, x_1) \leftarrow (\texttt{xAdd}(x_0, x_1, x_P), \texttt{xDbl}(x_0))$
  else
    $(x_0, x_1) \leftarrow (\texttt{xDbl}(x_0), \texttt{xAdd}(x_0, x_1, x_P))$
  end if
end for
return (x_0, y_0) using Equation (2.7)

Denote R_0 (resp. R_1) the point of x-coordinate x_0 (resp. x_1). During the `for` loop of

Algorithm 2.8, $x_{R_0-R_1}$ is an invariant equal to x_P. This explains why `xAdd` third input is always x_P. Finally, x_0 (resp. x_1) is the x-coordinate of $[n]P$ (resp. $[n+1]P$). At the end of Algorithm 2.8, the y-coordinate y_0 of $[n]P$ is recovered using the following formula:

$$y_0 = \frac{(x_P x_0 + 1)(x_P + x_0 + 2A) - 2A - (x_P - x_0)^2 x_1}{2By_P}. \tag{2.7}$$

A detailed analysis of the cost of Algorithm 2.8 is given in [CS18]. In theory, the Montgomery model is the fastest model but as explained in Definition 2.28, an elliptic curve needs to satisfy specific properties in order to be represented in Montgomery coordinates. The modified projective model is a good choice for a generic curve (characteristic $p > 3$). Moreover, optimizations are available when the curve equation is $y^2 = x^3 - 3x + b'$: two multiplications can be saved in the addition algorithm. As we have seen in Section 2.7, it is possible half of the time.

2.8.4 Subgroup security

As we have seen in the begining of Section 2.8, for a cryptographic application, we use an elliptic curve whose order has a large subgroup of prime order r of 256 bits. Whenever the group $E(\mathbb{F}_q)$ has few small subgroups, then one can do a man-in-the-middle attack during particular protocols if implemented without adequate protection. For instance, this attack could work in a Diffie–Hellman key exchange, as in Figure 2.4. Suppose that Alice and Bob want to share a common key using their respective secrets a and b. Suppose also that the elliptic curve E has order $\#E(\mathbb{F}_q) = 2r$ where r is a 256-bit integer. Then, an attacker (Eve) can simply do a man-in-the-middle attack and send a point $P_2 \in E(\mathbb{F}_q)[2]$ instead of Alice's public key $P_a = [a]P$. Then, from the common *secret* key, Eve recovers the secret of Bob modulo 2, i.e. one bit of the secret. This attack generalizes for other small primes. Eve computes discrete logarithms in small subgroups as long as they are not too large. To counter this attack, Bob needs to check whether P_A is indeed in the large subgroup.

In the next chapters, we will use the following definitions which measure the number of small subgroups.

Definition 2.29 (η-subgroup-security)**.** *A curve E is η-subgroup-secure over $\mathbb{F}_q$ if all the factors of $E(\mathbb{F}_q)$ are at least as large as r, except those of size η.*

Definition 2.30 (ρ-value)**.** *The ρ-value is the ratio $\rho = \log_2(q)/\log_2(r)$. It is used to measure how far the curve parameters are from the optimal case where the curve is of prime order ($\rho = 1$ in this case).*

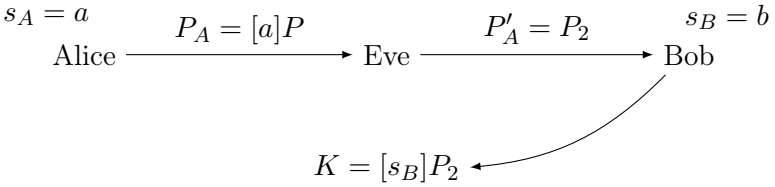


Figure 2.4: Man-in-the-middle attack when 2 divides $\#E(\mathbb{F}_q)$.

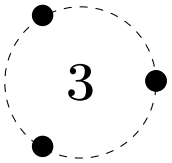
2.8.5 Twist security

The latter subgroup attack can also be used on the quadratic twist.

Definition 2.31 (Twist-subgroup-security)**.** *A curve is twist-subgroup-secure if its quadratic twist is subgroup-secure.*

In Figure 2.4, if Alice uses x-only arithmetic on Montgomery curve, then she needs to recover y_{P_A} from x_{P_A} before sending it to Bob. As we have seen in Section 2.7, x_{P_A} is also the x-coordinate of points of the quadratic twist of E. Hence, Eve can send P'_A a point of the twist E^t, instead of P_A. Then the same attack can be applied if P'_A has a smooth order. In order to prevent this attack, it is important to design elliptic curve cryptography where the security of E and its quadratic twist are equivalent or, alternatively, to implement (quite expensive) checks that peer-provided points belong to the expected curve.

Most (unfortunately, not all) standard elliptic curves have been designed in order to have twist- and subgroup-security. A study of several standards covering selection of curves for use in elliptic-curve cryptography is given at `https://safecurves.cr.yp.to/`. For instance, the curve `Ed25519` [Ber06] is subgroup- and twist-secure in the sense that $\#\texttt{Ed25519}(\mathbb{F}_p) = 2^3 \cdot r$ and $\#\texttt{Ed25519}^t(\mathbb{F}_p) = 2^2 \cdot r'$, where r (resp. r') is a prime integer of 252 (resp. 253) bits. In Chapter 4, we will design elliptic curves with particular properties. Among them, we will study their subgroup- and twist-security.



Isogenies in cryptography

In this chapter, we use isogenies of elliptic curves introduced in Chapter 2 to define several post-quantum cryptography protocols. The *hard problem* we use is not related to a discrete logarithm problem in a subgroup of points of an elliptic curve as in Chapter 2, but we will see in Chapter 7 that we can merge the two problems to get interesting cryptographic applications.

We begin this chapter with some details on isogenies. In particular, we present the graph of isogenies, which has a very different structure depending on the supersingularity of the elliptic curves we consider. On the one hand, ordinary curves are closely related to the class group theory. On the other hand, supersingular curves correspond to cosets of maximal orders in a quaternion algebra. In a second section, we dig into detail on the computation of isogenies in order to walk through the isogeny graph. From these isogenies, one can obtain post-quantum cryptography protocols, which are detailed in the last section.

Summary

3.1 Isogeny graphs

We consider now separable isogenies in the sense of Definitions 2.8 and 2.10, and curves up to an $\bar{\mathbb{F}}_p$-isomorphism. Such isogenies are fully defined from their kernel: from an elliptic curve E defined

over $\mathbb{F}_q$ and a finite subgroup $G \subset E(\bar{\mathbb{F}}_q)$, there exists a unique isogeny (up to an $\bar{\mathbb{F}}_p$-isomorphism of the target curve) $\phi : E \to E/G$ i.e., an isogeny from E with kernel G. Recall that for separable isogenies, $\deg(\phi) = \#G$. In the following, we consider curves up to an $\bar{\mathbb{F}}_q$-isomorphism, or up to an $\bar{\mathbb{F}}_q$-isogeny. From the considerations of Chapter 2, there exists for any ℓ-isogeny $\phi : E \to E'$ a unique ℓ-isogeny $\hat{\phi} : E' \to E$, called the *dual* of ϕ, such that $\phi \circ \hat{\phi} = [\ell]$ on E' and $\hat{\phi} \circ \phi = [\ell]$ on E. This shows that being *ℓ-isogenous* is a symmetric relation, and that being isogenous is an equivalence relation (as well as being isomorphic).

Definition 3.1 (Isogeny class, isomorphism class)**.** *An isogeny class is a set of $\bar{\mathbb{F}}_q$-isogenous elliptic curves. Similarly, an isomorphism class is the set of $\bar{\mathbb{F}}_q$-isomorphic elliptic curves. An $\mathbb{F}_q$-isogeny (resp. $\mathbb{F}_q$-isomorphism) class is the set of curves isogenous (resp. isomorphic) where the isogeny (resp. the isomorphism) is defined over $\mathbb{F}_q$.*

We consider the infinite graph of $\bar{\mathbb{F}}_q$-isogenous curves: vertices of the graph are $\bar{\mathbb{F}}_q$-isomorphism classes of elliptic curves, and edges correspond to isogenies between curves. We use the j-invariant to label the vertices of the graph: the j-invariant is an invariant for $\bar{\mathbb{F}}_q$-isomorphic elliptic curves. We also consider the subgraph of ℓ-isogenous curves (where ℓ is a prime different from the characteristic p of $\mathbb{F}_q$). It means that for a curve E (a representative of an isomorphism class, i.e. a vertex), we only look at isogenies of degree ℓ, i.e. at order-ℓ subgroups of $E(\bar{\mathbb{F}}_q)[\ell]$. From Section 2.4, the ℓ-torsion subgroup $E[\ell]$ is a 2-dimensional $\mathbb{Z}/\ell\mathbb{Z}$ vector space. Writing $E[\ell] = \langle P_\ell, Q_\ell \rangle$, the (cyclic) subgroups of order ℓ of $E[\ell]$ are generated by the $[i]P_\ell + Q_\ell$ ($0 \leq i \leq \ell - 1$) and P_ℓ. Hence there are exactly $\ell + 1$ isogenies (edges) of degree ℓ from a given curve (a vertex of the graph), and the ℓ-isogeny graph is $(\ell + 1)$-regular.

Looking only at isogenies defined over $\mathbb{F}_q$, the $\mathbb{F}_q$-isogeny class has particular properties. A theorem of Tate states that two curves are isogenous over $\mathbb{F}_q$ if and only if they have the same number of points over $\mathbb{F}_q$, thus in particular it preserves the trace and the supersingularity. Hence, a graph of $\mathbb{F}_q$-isogenous curves contains either ordinary curves or supersingular curves. Thus, we study the isogeny graph by considering the ordinary and the supersingular cases separately.

3.1.1 Ordinary curves

The theory of isogenous ordinary curves has been studied by Kohel in [Koh96]. The graph of ordinary curves is structured by the different endomorphism rings of the isogenous curves. As described in Theorem 2.19, endomorphism rings of ordinary curves defined over $\mathbb{F}_q$ with trace t are orders in the imaginary quadratic field $K = \mathbb{Q}(\sqrt{t^2 - 4q})$. We denote $-D < 0$ the discriminant of K (see Definition 2.16).

The endomorphism ring of an ordinary curve E defined over $\mathbb{F}_q$ is isomorphic to an order of K, and considering $\pi \in K$ such that $\pi^2 - t\pi + q = 0$, the suborder $\mathbb{Z}[\pi]$ is always contained in $\mathrm{End}(E)$: the Frobenius $(x, y) \mapsto (x^q, y^q)$ is an endomorphism corresponding to π in K, and scalar multiplications are endomorphisms corresponding to the integers $\mathbb{Z}$ in K. From the notations of Section 2.5.1, we write its discriminant $-D_\pi = t^2 - 4q$ and its conductor f_π: $-D_\pi = -Df_\pi^2$. Hence, $\mathbb{Z}[\pi] \subseteq \mathrm{End}(E) \subseteq \mathcal{O}_K$ and there is only a finite number of possible choices for endomorphism rings of ordinary curves defined over $\mathbb{F}_q$, corresponding to the conductors dividing f_π. As an illustration, we give the possible endomorphism rings for a small value of D_π in Figure 3.1, where the indices are written on the edges.

We now fix a prime ℓ coprime to q. If $\phi : E \to E'$ is an ℓ-isogeny, then one of the two endomorphism rings ($\mathrm{End}(E)$ or $\mathrm{End}(E')$) contains the other one, with index 1 or ℓ [Koh96, page 44]. From that, we define three different types of isogenies.

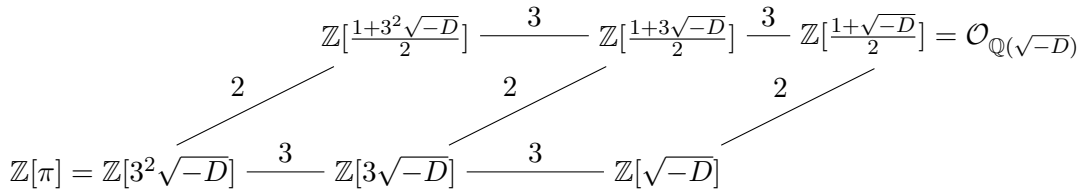


Figure 3.1: Possible endomorphism rings of ordinary curves when $t^2 - 4q = -2^2 \cdot 3^4 \cdot 71$.

Definition 3.2 (Horizontal, descending and ascending isogenies)**.** *Let $\phi : E \to E'$ be an ℓ-isogeny. Then,*

- *If* $\mathrm{End}(E) = \mathrm{End}(E')$*, ϕ is said to be* horizontal*,*
- *If* $[\mathrm{End}(E) : \mathrm{End}(E')] = \ell$*, ϕ is said to be* descending*,*
- *If* $[\mathrm{End}(E') : \mathrm{End}(E)] = \ell$*, ϕ is said to be* ascending*.*

In particular, an ℓ-isogeny graph corresponds to a chain of inclusions of orders where the conductors between them are ℓ. Kohel obtains in [Koh96, page 44] the following proposition which uses the Kronecker symbol $(\frac{-D}{\ell})$ (see also [DFHPS16]).

Proposition 3.3. *Let E be an ordinary elliptic curve such that*

$$\mathbb{Z}[\pi] \xrightarrow{f_\pi/f} \mathrm{End}(E) \xrightarrow{f} \mathcal{O}_K.$$

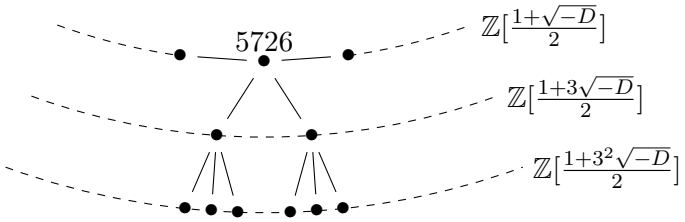
Then, for a prime $\ell \neq p$, the number of ℓ-isogenies from E is determined as follows:

- *If ℓ divides f but not f_π/f, then there is only one ascending isogeny,*
- *If ℓ divides both f and f_π/f, then there are one ascending isogeny and ℓ descending isogenies,*
- *If ℓ divides neither f nor f_π/f, then there are $1 + (\frac{-D}{\ell})$ horizontal isogenies,*
- *If ℓ does not divide f but divides f_π/f, then there are $1 + (\frac{-D}{\ell})$ horizontal isogenies and $\ell - (\frac{-D}{\ell})$ descending isogenies.*

The graph of ℓ-isogenies is often represented with different layers for each possible endomorphism ring, and is thus called a volcano. The graph has different connected components (see Figure 3.1 for an example).

We provide a concrete example corresponding to the cases of Figure 3.1. We choose a curve with endomorphism ring the maximal order of $\mathbb{Q}(\sqrt{-D})$ with $D = 71$, that is $\mathbb{Z}[\frac{1+\sqrt{-D}}{2}]$ because $-D \equiv 1 \bmod 4$. Let $p = 5851$ and $E_{1394,2040}$ defined over $\mathbb{F}_p$. This way, $E_{1394,2040}$ has j-invariant 5726 which is a root of the Hilbert class polynomial $H_{71}(x)$, and so $\mathrm{End}(E_{1394,2040}) = \mathbb{Z}[\frac{1+\sqrt{-D}}{2}]$. Moreover, the curve satisfies $t^2 - 4p = -2^2 \cdot 3^4 \cdot 71$ so that $\mathbb{Z}[\pi]$ has conductor $2 \cdot 3^2$, meaning that we are indeed in the case of Figure 3.1.
The ℓ-isogeny graph of ordinary curves is infinite, but if we restrict to curves defined over $\mathbb{F}_p$, it corresponds to endomorphism rings that are orders of Figure 3.1. All these curves have as a suborder $\mathbb{Z}[\pi] \subset \mathrm{End}(E)$. Looking at 3-isogenous curves, we can reach endomorphism rings of conductor a power of 3 in $\mathbb{Z}[\frac{1+\sqrt{-D}}{2}]$, namely $\mathbb{Z}[\frac{1+3\sqrt{-D}}{2}]$ and $\mathbb{Z}[\frac{1+3^2\sqrt{-D}}{2}]$.
This way, the volcano of 3-isogenies has three layers represented with the arcs below and we

know the number of ascending, descending and horizontal isogenies depending on the different conductors. For example, the $j = 5725$ curve corresponds to the maximal order $\mathcal{O}_K$ with conductor $f = 1$, and as $(\frac{-71}{3}) = 1$, there are $1 + 1$ horizontal isogenies and $3 - 1$ descending isogenies.



Practically, we reach the neighbor curves of $j = 5726$ using the eight points of order 3 defined over $\mathbb{F}_p$, which form four distinct subgroups. They lead to four isogenies of degree 3 from $E_{1394,2040}$. We obtain two curves ($j = 4481$ and 4341) with the same endomorphism ring $\mathcal{O}_K$, and two curves ($j = 4120$ and 3489) with endomorphism ring $\mathbb{Z}[\frac{1+3\sqrt{-D}}{2}]$. Finally, the structure of the graph gives the number of curves at each height (which corresponds to the degrees of $H_{3^4.71}$, $H_{3^2.71}$ and H_{71}). We get the connected component of the graph of 3-isogenies in the right hand side of Figure 3.2. If we begin with another curve whose j-invariant is a root of $H_{2^2.3^{2a}.71}(x)$ $(0 \leq a \leq 2)$, then it corresponds to the left hand side of Figure 3.2, which is similar to the right hand side because $\deg(H_{2^2.71}) = \deg(H_{71})$, $\deg(H_{2^2.3^2.71}) = \deg(H_{3^2.71})$ and $\deg(H_{2^2.3^4.71}) = \deg(H_{3^4.71})$. The curve whose j-invariant is 5508 has endomorphism ring $\mathbb{Z}[\sqrt{-D}]$ (conductor 4 in $\mathcal{O}_K$).

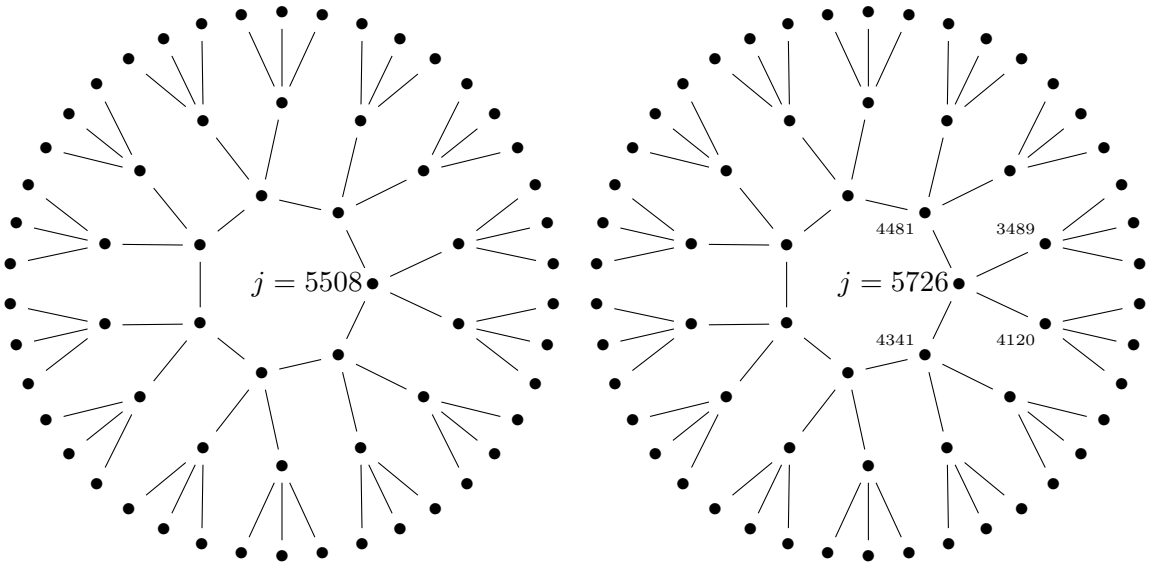


Figure 3.2: The connected components of the 3-isogeny graph corresponding to Figure 3.1

Class group action. In the ordinary case, isogenies can be translated in terms of a group action. We refer to [DF17, page 26] for a more complete description. Suppose that E is an elliptic curve of endomorphism ring an order $\mathcal{O}$. A (fractional) ideal I of $\mathcal{O}$ can be seen as a set of endomorphisms of E (up to a scalar). Hence, it makes sense to consider $E[I] := \{P \in E, \alpha(P) = 0_E \text{ for all } \alpha \in I\}$. From this subgroup of the curve E, we obtain an action of the fractional ideals on the curves of

endomorphism ring $\mathcal{O}$ by applying the isogeny of kernel $E[I]$. Principal ideals act trivially: the unique isogeny whose kernel is the subgroup $E[\alpha\mathcal{O}]$ is the endomorphism α, which is an isogeny from E to itself. Thus we look at the quotient group of fractional ideals by the principal ones, called the *class group*. It acts simply transitively on the set $\mathrm{Ell}(\mathcal{O})$ (defined in Section 2.5.4) as follows:

$$\begin{array}{rcl} \mathrm{Cl}(\mathcal{O}) \times \mathrm{Ell}(\mathcal{O}) & \longrightarrow & \mathrm{Ell}(\mathcal{O}) \\ (I, E) & \longmapsto & I * E = E/E[I]. \end{array}$$

In particular, the set of $\mathrm{Ell}(\mathcal{O})$ has $\#\,\mathrm{Cl}(\mathcal{O})$ elements, the computation of an isogeny is related to an ideal action. More precisely, an isogeny of prime degree ℓ corresponds to an ideal $\mathfrak{l}$ of norm ℓ in the class group. Hence, in order to get an efficient group action, it makes sense to decompose ideals of the class group into products of smooth ideals, which corresponds to isogenies of smooth degree (see Section 3.2 for details on the efficiency of the isogeny computation). We will see in Section 3.3 possible instantiations of an efficient group action.

3.1.2 Supersingular curves

Graphs of supersingular isogenies have been studied by Mestre [Mes86], Pizer [Piz90, Piz98], Kohel [Koh96], Delfs and Galbraith [DG16], among others. The story is very different from the ordinary case. Supersingular curves defined over a field of characteristic p can always be defined over $\mathbb{F}_{p^2}$ (see [Sil86, page 145]). Hence, we now consider supersingular elliptic curves defined over a quadratic finite field $\mathbb{F}_q$ where $q = p^2$. By the Hasse bound, the trace of t of the curve satisfies $|t| \leq 2\sqrt{q}$. By definition of the supersingularity, p divides t and finally, $t \in \{0, \pm p, \pm 2p\}$. Proposition 3.4 shows that the only interesting cases of isogeny graphs of supersingular curves are for trace $t = \pm 2p$.

Proposition 3.4. *Let E be a supersingular curve defined over $\mathbb{F}_{p^2}$. If the trace t of the curve is 0, p or $-p$, then the isogeny class contains only one point.*

Proof. The order $\mathbb{Z}[\pi]$ has discriminant $t^2 - 4q$. If $t = 0$ (resp. $t = \pm p$), then $t^2 - 4q = -4p^2$ (resp. $-3p^2$). It corresponds to the fundamental discriminant -4 (resp. -3), and hence $H_4(x) = (x - 1728)$ and $j_E = 1728$ (resp. $H_3(x) = x$ and $j_E = 0$). □

Remark 3.5. We consider in Proposition 3.4 the trace of the curve as seen over $\mathbb{F}_{p^2}$. Looking at supersingular curves defined over $\mathbb{F}_p$, there is more than one curve of trace (over $\mathbb{F}_p$) $t = 0$ on the $\mathbb{F}_p$-isogeny class. We investigate these curves in Section 3.1.2.

Elliptic curves with trace $t = \pm 2p$ have order $\#E(\mathbb{F}_{p^2}) = p^2 + 1 - t = (p \mp 1)^2$. These elliptic curves, defined over a finite field of characteristic p of the form $p = f\ell \pm 1$ always have a $\mathbb{F}_{p^2}$-rational subgroup of order ℓ: $\#E(\mathbb{F}_{p^2}) = f^2\ell^2$.

Graph of curves defined over $\mathbb{F}_{p^2}$

The study of the graph of supersingular curves is closely related to the endomorphism rings of elliptic curves. From Theorem 2.19, we know the structure of endomorphism rings of supersingular elliptic curves. Thus, we do not encounter the concept of horizontal isogenies anymore: every endomorphism ring is isomorphic to a maximal order in a quaternion algebra, and to every maximal order corresponds exactly a pair of ($\mathbb{F}_{p^2}/\mathbb{F}_p$-Galois conjugate) supersingular curves. Contrary to the theory of orders in imaginary quadratic fields, there are many different maximal orders in quaternion algebras. We investigate this correspondence explicitly in Chapter 6.

Isogenous curves have the same Frobenius characteristic polynomial and hence the same trace t. Hence, supersingular curves of trace $t = 2p$ and $-2p$ produce two distinct isogeny classes. The number of curves (up to isomorphism) in each class is given by

$$\lfloor p/12 \rfloor + \begin{cases} 0 & \text{if } p \equiv 1 \bmod 12 \\ 1 & \text{if } p \equiv 5, 7 \bmod 12 \\ 2 & \text{if } p \equiv 11 \bmod 12. \end{cases}$$

The two graphs (of curves with $t = 2p$ and $t = -2p$) are isomorphic in the sense that each curve of trace $2p$ is isomorphic to a curve of trace $-2p$ (and the isomorphism is the quadratic twist, defined over a quadratic extension of $\mathbb{F}_{p^2}$). We typically speak of supersingular graphs over $\bar{\mathbb{F}}_p$ and over $\mathbb{F}_{p^2}$ indistinctly.

We now provide an example of supersingular isogeny graph. Let $p = 2^2 \cdot 3^3 - 1$ and $\mathbb{F}_{p^2} = \mathbb{F}_p(i)$ where $i^2 = -1$. Consider the elliptic curve $E_{102i+81,10i+53}$ defined over $\mathbb{F}_{p^2}$. Its trace is $t = -2p$ and $\#E(\mathbb{F}_{p^2}) = (p+1)^2 = 2^4 \cdot 3^6$. There are $\lfloor p/12 \rfloor + 2 = 10$ curves in the isogeny class containing this curve, and the graph of 2-isogenies is 3-regular (except at the ramification vertices $j = 0, 1728$).

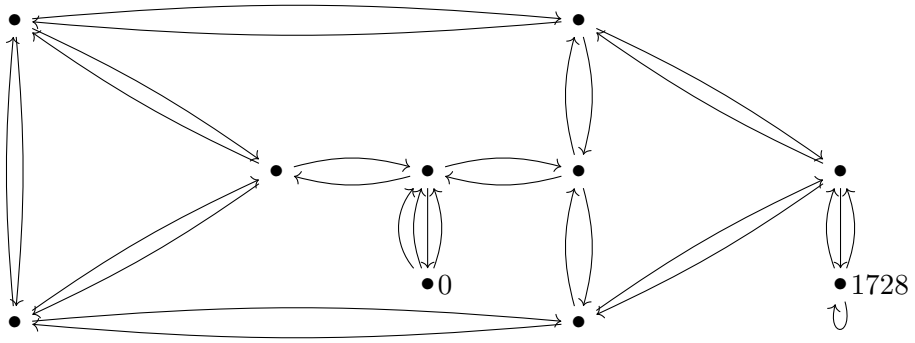


Figure 3.3: Supersingular 2-isogeny graph over $\mathbb{F}_{p^2}$ where $p = 2^2 \cdot 3^3 - 1$ and $t = -2p$.

In the next paragraph, we consider the graph of $\mathbb{F}_p$-isomorphism classes of supersingular curves defined over $\mathbb{F}_p$, and isogenies defined over $\mathbb{F}_p$. We may find inside the graph of curves defined over $\mathbb{F}_{p^2}$ a sub-structure inherited from the graph of curves defined over $\mathbb{F}_p$. One may be tempted to think that the $\mathbb{F}_{p^2}$-graph contains the $\mathbb{F}_p$-graph as a subgraph, however the situation is slightly subtler: indeed, supersingular curves defined over $\mathbb{F}_p$ are isogenous to their quadratic twists, thus the $\mathbb{F}_p$-graph contains pairs of vertices that become isomorphic in $\mathbb{F}_{p^2}$. Hence, the ℓ-isogeny graph of curves and isogenies defined over $\mathbb{F}_p$ is a double cover (outside the ramification points at $j = 0, 1728$) of the $\mathbb{F}_p$-subgraph contained in the ℓ-isogeny graph over $\mathbb{F}_{p^2}$. The $\mathbb{F}_p$-graph corresponding to Figure 3.3 is given in Figure 3.4.

Graphs of curves defined over $\mathbb{F}_p$

Delfs and Galbraith showed that one obtains the same kinds of undirected graphs as for ordinary curves using horizontal, ascending and descending isogenies. In this context, the ring of endomorphisms of E defined over $\mathbb{F}_p$ is denoted $\mathrm{End}_p(E)$ and is an order of an imaginary quadratic field, as in the ordinary case. In the same way, $\mathbb{Z}[\pi]$ is an order of discriminant $t^2 - 4p = -4p$, which leads to only two possible $\mathbb{F}_p$-endomorphism rings: $\mathbb{Z}[\sqrt{-p}]$ and $\mathbb{Z}[\frac{1+\sqrt{-p}}{2}]$ depending on the congruence of p modulo 4. In this context, we say that an isogeny $\phi : E \to E'$ is *horizontal* whenever

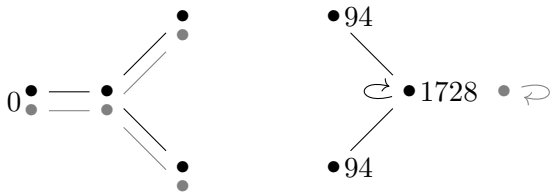


Figure 3.4: Supersingular 2-isogeny $\mathbb{F}_p$-graph over $\mathbb{F}_p$ where $p = 2^2 \cdot 3^3 - 1$. Gray vertices correspond to the quadratic twist of the black vertices

$\mathrm{End}_p(E) \simeq \mathrm{End}_p(E')$. We define the horizontal isogeny classes as in Definition 3.1, but restricting to horizontal isogenies (equivalently, it is the set of curves with same $\mathbb{F}_p$-endomorphism ring). Delfs and Galbraith showed that there are one or two horizontal isogeny classes of supersingular curves over $\mathbb{F}_p$, according to whether $p = 1$ or 3 mod 4. Precisely:

- If $p = 1 \bmod 4$, then $\mathrm{End}_p(E) \simeq \mathbb{Z}[\sqrt{-p}]$ for all curves.
- If $p = 3 \bmod 4$, then $\mathrm{End}_p(E)$ is isomorphic to one of $\mathbb{Z}[\sqrt{-p}]$ or $\mathbb{Z}[\frac{1+\sqrt{-p}}{2}]$; the horizontal isogeny class associated to $\mathbb{Z}[\frac{1+\sqrt{-p}}{2}]$ is called the *surface*, and the horizontal isogeny class associated to $\mathbb{Z}[\sqrt{-p}]$ is called the *floor*.

We apply Proposition 3.3 in a particular case where the only possible conductor f of $\mathrm{End}_p(E)$ in the maximal order of $\mathbb{Q}(\sqrt{-p})$ is either 1 or 2.

Proposition 3.6 (from [Koh96, page 44])**.** *Let E be a supersingular elliptic curve defined over $\mathbb{F}_p$. Then, the number of ℓ-isogenies defined over $\mathbb{F}_p$ depends on the divisibility of f and f/f_π by ℓ:*

- *If ℓ is an odd prime, then ℓ does not divide neither f or f_π/f and and there are $1 + \left(\frac{-D}{\ell}\right)$ horizontal isogenies. If $\left(\frac{-p}{\ell}\right) = -1$, there is no ℓ-isogeny of supersingular curves defined over $\mathbb{F}_p$, i.e., the ℓ-isogeny graph is made of isolated vertices. If $\left(\frac{-p}{\ell}\right) = 1$, every curve has exactly two horizontal ℓ-isogenies, thus each horizontal isogeny class is partitioned into a finite number of cycles.*
- *If $\ell = 2$, then the structure depends on the maximality of $\mathrm{End}_p(E)$ which is closely related to the congruence of p modulo 4.*
  - *If $p = 1 \bmod 4$, $\mathrm{End}_p(E)$ is maximal in $\mathbb{Q}(\sqrt{-p})$ and has discriminant $-4p$, and every curve has exactly $1 + \left(\frac{-4p}{2}\right) = 1$ horizontal ℓ-isogeny.*
  - *If $p = 3 \bmod 4$, then every curve on the floor corresponds to a conductor $f = 2$ and so has exactly one non-horizontal ℓ-isogeny going to a curve on the surface, whereas for curves on the surface, the number of isogenies is related to the congruence of p mod 8. If $p = 7 \bmod 8$, they have exactly two horizontal ℓ-isogenies, plus one non-horizontal going to the floor (dual to the one coming from the floor); if $p = 3 \bmod 8$, they have three non-horizontal isogenies going to three curves on the floor (dual to the ones coming from the floor).*

Figure 3.5 describes the different possible ℓ-isogenies defined over $\mathbb{F}_p$ for a given supersingular curve (defined over $\mathbb{F}_p$) represented with the symbol $\circ$. In cryptographic applications, we will only be interested in cycles of horizontal isogenies, thus either the second case or the fourth case (on the

surface) of Figure 3.5. In particular, we present in Section 3.3.2 a key exchange [CLM+18] that focuses on $\ell \neq 2$ and $\left(\frac{-p}{\ell}\right) = 1$. We investigate in Chapter 7 the case of $\ell = 2$ and $p = 7 \bmod 8$.

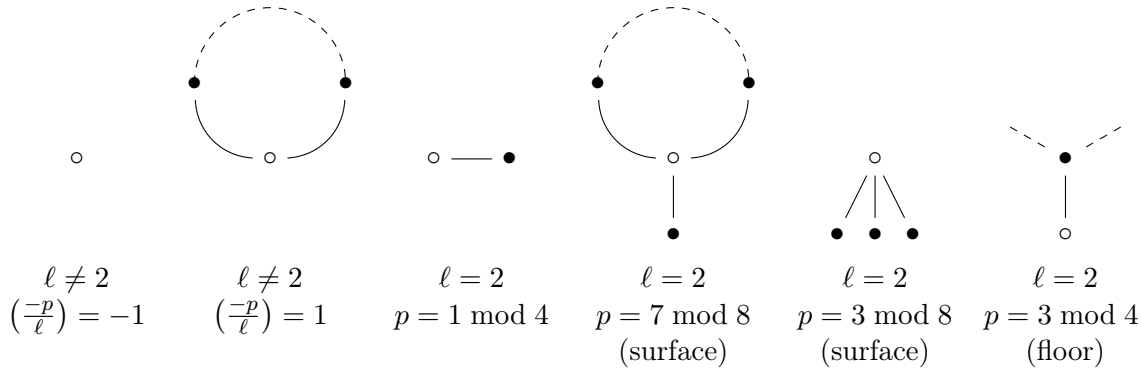


Figure 3.5: ℓ-isogenies defined over $\mathbb{F}_p$ from a supersingular curve.

3.2 Isogeny computation

We consider here separable isogenies in the sense of Definitions 2.8 and 2.10. The *degree* of an isogeny ϕ is the degree of the field extension induced by ϕ^* (see Definition 2.9), but in the case of separable isogenies, it corresponds to the size of its kernel. A separable isogeny is said to be *cyclic* if its kernel is; we will mostly deal with cyclic isogenies in Chapter 7. We first investigate formulas for computing ℓ-isogenies in $O(\ell)$, called Vélu's formulas.

3.2.1 Vélu's formulas

In [Vél71], Vélu introduces formulas for computing a separable isogeny, given its kernel. The complexity of the algorithm is linear in the size of the kernel (i.e. the degree).

Proposition 3.7 (Vélu's formulas)**.** *Let $E_{a,b}$ be an elliptic curve defined over a field K, and let $G \subset E(\bar{K})$ be a finite subgroup. We denote $G^* = G - \{0_E\}$. The separable isogeny*

$$\phi : P \longmapsto \left(x_P + \sum_{Q \in G^*} x_{P+Q} - x_Q, y_P + \sum_{Q \in G^*} y_{P+Q} - y_Q \right)$$

has kernel G and codomain $E_{a',b'}$ where

$$a' = a - 5 \sum_{Q \in G^*} (3x_Q^2 + a) \qquad b' = b - 7 \sum_{Q \in G^*} (5x_Q^3 + 3ax_Q + b).$$

From the formulas, the complexity of the algorithm is linear in the size of the subgroup G: it computes sums of points for all elements of G. Hence, Vélu's formulas are inefficient for large degrees. From now on, we consider that ℓ is a small prime, and that ϕ is an isogeny of degree a power of ℓ. As we consider separable isogenies, it also corresponds to the order of the kernel G. If $\#G = \ell^n$, then Vélu's formulas become very expensive as n grows. We usually speak about *a walk on the ℓ-isogeny graph* instead of a ℓ^n-isogeny. Indeed, in order to compute a ℓ^n-isogeny, we start at a given curve, and compute n isogenies of degree ℓ and stop at another curve.

In practice, we use slightly different formulas in order to get efficient algorithms:

- As we look at curves up to an isomorphism (defined over $\bar{\mathbb{F}}_p$ or $\mathbb{F}_p$ depending on the context), we can choose the codomain curve representing the isomorphism class coset in order to improve the efficiency.
- From the considerations of Table 2.1, the elliptic curve group law arithmetic is more efficient in x-only Montgomery coordinates.

From now on, we assume that the elliptic curves we consider can be written in Montgomery coordinates (with an equation $By^2 = x^3 + Ax^2 + x$). An $\bar{\mathbb{F}}_p$-isomorphism class representative can be chosen with $B = 1$ so that we consider only the A coefficient in the following. We adapt Proposition 3.7 in order to get x-only formulas in the Montgomery model. As an example, we investigate the case of 2-isogenies.

Isogenies of degree 2. Cyclic isogenies are closely related to their kernel, so we need to study the 2-torsion of Montgomery curves.

Proposition 3.8. *Let E be an elliptic curve defined by a Montgomery equation $By^2 = x^3+Ax^2+x$. Then, $E[2] = \{0_E, (0,0), (\alpha,0), (1/\alpha,0)\}$ where α is a root of $x^2 + Ax + 1$.*

From now on, we represent elliptic curves with an equation $M_\alpha : y^2 = x^3 - (\alpha + 1/\alpha)x^2 + x$. Applying the formulas for the order 2 subgroup $G = \{0_E, (\alpha, 0)\}$, we get the rational maps defining a degree 2 isogeny. Efficient formulas have been obtained in order to preserve the Montgomery model and improve the efficiency of the 2-isogeny evaluation. We refer to [JD11] and [Ren18] for details.

Proposition 3.9 (Derived from [Ren18, page 10])**.** *Let $\alpha \neq 0$ and M_α be an elliptic curve in Montgomery model. Define*

$$f_0(x) = \frac{(x+1)^2}{4\alpha x} \qquad f_\alpha(x) = \frac{x(\alpha x - 1)}{x - \alpha} \qquad f_{1/\alpha}(x) = \frac{x(x-\alpha)}{\alpha x - 1}.$$

Then, f_α defines an isogeny of kernel $\langle(\alpha, 0)\rangle$

$$\begin{array}{rccc} \phi_\alpha : & M_\alpha & \longrightarrow & M_{2\alpha(\alpha+\sqrt{\alpha^2-1})-1} \\ & (x,y) & \longmapsto & (f_\alpha(x), \sqrt{\alpha} y f'_\alpha(x)). \end{array}$$

Using this codomain curve, the two other isogenies of kernel generated by $(0,0)$ and $(1/\alpha,0)$ are defined by $(x,y) \mapsto (f_0(x), c_0 y f'_0(x))$ and $(f_{1/\alpha}(x), c_{1/\alpha} y f'_{1/\alpha}(x))$ for two constants c_0 and $c_{1/\alpha}$ we do not explicit here.

This codomain isomorphism class representative $M_{2\alpha(\alpha+\sqrt{\alpha^2-1})-1}$ is chosen so that $\phi_\alpha(0,0) = \phi_\alpha(1/\alpha,0) = (0,0)$. As $\hat{\phi} \circ \phi = [2]$, the kernel of $\hat{\phi}$ is $(0,0)$ and $\hat{\phi}(x,y) = (f_0(x), \ldots)$. We now investigate isogenies of degree a power of 2 as a composition of isogenies of degree 2. In the following, Vélu(P) denotes the isogeny of kernel P, computed using Vélu's formulas (or Proposition 3.9 if $\ell = 2$).

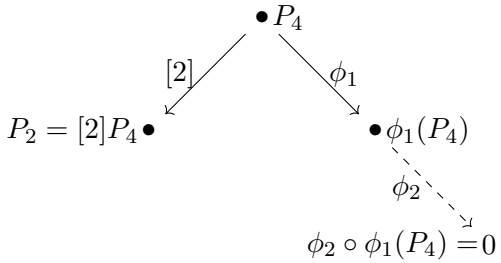
We now focus on the case of isogenies of degree ℓ^n where ℓ is a prime. Recall that we look at separable isogenies, i.e. defined with their kernel. Using the Chinese Remainder Theorem on the kernel subgroup, we can always split an isogeny as a composition of isogenies of prime power degree.

3.2.2 Isogenies of degree a power of ℓ

In Chapter 7, we aim to evaluate isogenies of degree a power of 2 at points, and compute j-invariant of the codomain curve. Vélu's formulas can be adapted for a group of order ℓ^n but become quickly impractical as the complexity is linear in the degree.

De Feo, Jao and Plût decompose an isogeny of degree ℓ^n into n isogenies of degree ℓ in [DFJP14]. We first investigate the computation of a 4-isogeny as an example. Such an isogeny has a cyclic kernel generated by a point P_4 of order 4. One can compute the isogeny by splitting the computation into two isogenies of degree 2 as follows:

1. Compute $P_2 = [2]P_4$, a point of order 2.
2. Use Proposition 3.9 to compute the 2-isogeny ϕ_1 whose kernel is $\langle P_2 \rangle$, and evaluate it at P_4. The point $\phi_1(P_4)$ has order 2:
$$[2]\phi_1(P_4) = \phi_1(P_2) = 0.$$
3. Compute the 2-isogeny ϕ_2 of kernel $\langle \phi_1(P_4) \rangle$, again using Proposition 3.9. $\phi_2 \circ \phi_1$ is an isogeny of kernel $\langle P_4 \rangle$ as required.



This algorithm can be adapted to decompose an isogeny of degree ℓ^n as n isogenies of degree ℓ. Several variants (also presented in [DFJP14]) compute an ℓ^n-isogeny by decomposing it into n isogenies of prime degree ℓ. Let ϕ be an isogeny of degree ℓ^n. The kernel of ϕ is a subgroup of order ℓ^n. As in the example above, let $\ker(\phi) = \langle P \rangle$ for a point P of order ℓ^n. Then, the isogeny ϕ can be computed using an algorithm of [JD11] that requires a strategy. In the following, we explain three strategies of [JD11]. All these strategies compute some scalar multiplications $[\ell]$ and some ℓ-isogenies. From now on, we denote $\mathbf{c}_{[\ell]}$ the cost of a scalar multiplication by ℓ and $\mathbf{c}_{\ell\text{-isog}}$ the cost of an isogeny of degree ℓ.

The comb strategy

The first algorithm we present is probably the simplest way of splitting ϕ into n isogenies of degree ℓ. Algorithm 3.1 comes from [JD11] and computes the ℓ^n-isogeny with mostly scalar multiplications by ℓ. Figure 3.6 explains how the algorithm works: the top vertex ($\bullet$) is the point P defining the kernel of the target isogeny ϕ. The isogeny splits as $\phi_n \circ \ldots \circ \phi_1$. Each isogeny kernel is computed using mostly scalar multiplications by ℓ:

- The kernel of ϕ_1 is $[\ell^{n-1}]P$, computed using $n-1$ scalar multiplications by ℓ.
- From this point, ϕ_1 is computed using the formulas of Proposition 3.9. $\phi_1(P)$ is a point of order 2^{n-1} as $[2^{n-1}]\phi_1(P) = \phi_1([2^{n-1}]P) = 0$.
- $\ker(\phi_2)$ is obtained in the same way using $\phi_1(P)$ and $n-2$ scalar multiplications by ℓ, and so on.

Finally, Algorithm 3.1 computes $(n-1)n/2$ scalar multiplications by ℓ and n isogenies of degree ℓ using Vélu's formulas, leading to a cost of $\frac{n(n-1)}{2}\mathbf{c}_{[\ell]} + n\mathbf{c}_{\ell\text{-isog}}$.

Algorithm 3.1: ComputeIsogenyComb(P) from [JD11]

Input. P a point of order a power of ℓ.
Output. A list of ℓ-isogenies corresponding to the isogeny of kernel P.

if $P = 0$ **then**
  return Id
end if
$P_\ell \leftarrow P$
while $[\ell]P \neq 0$ **do**
  $P_\ell \leftarrow [\ell]P_\ell$
end while
$\phi_1 \leftarrow$ Vélu(P_ℓ)
return ComputeIsogenyComb($\phi_1(P)$) and ϕ_1

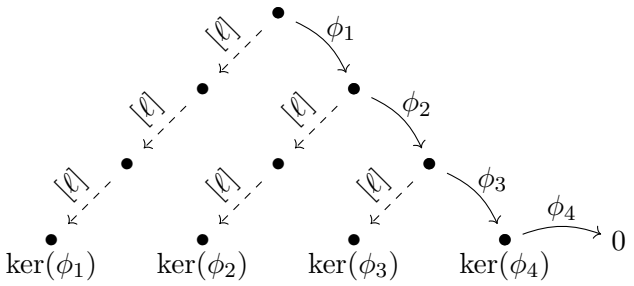


Figure 3.6: Computational structure of Algorithm 3.1 when $n = 4$.

Remark 3.10. Another comb strategy can be obtained reversing the roles of isogenies and scalar multiplications (see [JD11]). Once the first kernel point $[\ell^{n-1}]P$ is obtained, one computes ϕ_1 at all the intermediate points (i.e. the $[\ell^j]P$) and iterates replacing P by $\phi_1(P)$ which is of order ℓ^{n-1}. Figure 3.7 describes this variant in the case $n = 3$. It computes more isogenies than scalar multiplications, and its cost is $(n-1)\mathbf{c}_{[\ell]} + \frac{n(n-1)+2}{2}\mathbf{c}_{\ell\text{-isog}}$.

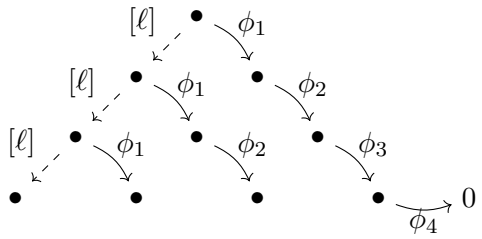


Figure 3.7: Computational structure of the mirror of Algorithm 3.1 when $n = 4$.

For fixed ℓ (so that $\mathbf{c}_{[\ell]} = O(1)$), the complexity is quadratic in n. It can be improved to $O(n \log_2(n))$ using a symmetric strategy.

The symmetric strategy

The two previous strategies compute either mostly multiplications, either mostly isogenies. The symmetric strategy is more balanced and computes as many isogenies as scalar multiplications.

Moreover, this strategy really improves the complexity as the isogeny of degree ℓ^n is computed in $O(n \log_2(n))$ steps (isogenies or scalar multiplications). In Algorithm 3.2, we suppose that n is a power of 2 for simplicity. The cost of Algorithm 3.2 is precisely $\frac{n \log_2(n)}{2} \mathbf{c}_{[\ell]} + (\frac{n \log_2(n)}{2} + 1)\mathbf{c}_{\ell\text{-isog}}$. As before, Figure 3.8 is clearer for understanding the algorithm.

Algorithm 3.2: ComputeIsogenySymmetric(n, P)

Input. n an integer,
P a point of order ℓ^n.
Output. A list of n isogenies of degree ℓ corresponding to the isogeny of kernel P.

if $n = 1$ **then**
  return Vélu(P)
end if
$P' \leftarrow [\ell^{n/2}]P$
$\phi \leftarrow$ ComputeIsogenySymmetric($n/2, P'$)
return ComputeIsogenySymmetric($n/2, \phi(P)$) and ϕ

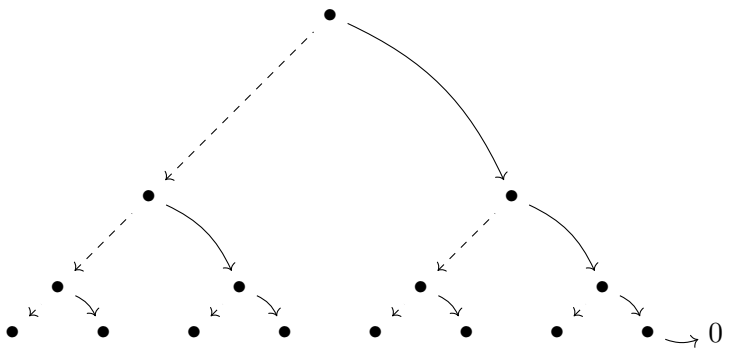


Figure 3.8: Computational structure of Algorithm 3.2 when $n = 8$.

The optimal strategy

For very small ℓ, a multiplication by ℓ costs roughly as much as an ℓ-isogeny. But when ℓ grows, $\mathbf{c}_{[\ell]} \ll \mathbf{c}_{\ell\text{-isog}}$. We finally present the optimal strategy that computes the trade-off between the number of scalar multiplications $[\ell]$ and the number of ℓ-isogenies (given the cost of $\mathbf{c}_{[\ell]}$ and $\mathbf{c}_{\ell\text{-isog}}$ as parameters). A binary tree corresponding to a strategy is coded in a list of integers representing the number of leaves at the right of each node, begining at the top node and walking in depth-first left-first. Algorithm 3.3 is taken from [JAC+19, Algorithm 46] and computes a good strategy to optimize the total cost of an ℓ^n-isogeny, given as inputs the number n of ℓ-isogenies, the cost p of a scalar multiplication $[\ell]$ and the cost q of an ℓ-isogeny.

The algorithms presented above can be translated using this representation:

- The comb strategy of Figure 3.6 is coded with `[3,2,1]` and its mirror in Figure 3.7 with `[1,1,1]`.
- The symmetric strategy of Figure 3.8 corresponds to `[4,2,1,1,2,1,1]`.

From a strategy s output from Algorithm 3.3, Algorithm 3.4 computes the evaluation of the isogeny of kernel generated by P, at a point Q (using the strategy s). We provide here the derecursified version.

Algorithm 3.3: GenerateStrategy(n, p, q)

Input. n an integer,
$p = \mathbf{c}_{[\ell]}$,
$q = \mathbf{c}_{\ell\text{-isog}}$.
Output. A strategy optimizing the cost of the corresponding isogeny.

$S \leftarrow$ a dictionary where the value at 1 is an empty list
$C \leftarrow$ a dictionnary where the value at 1 is 0
for i from 2 to $n+1$ **do**
  $b \leftarrow \text{argmin}_{0<b<i}(C[i-b] + C[b] + bp + (i-b)q)$
  $S[i] \leftarrow$ the concatenation of b, $S[i-b]$ and $S[b]$
  $C[i] \leftarrow C[i-b] + C[b] + bp + (i-b)q$
end for
return $S[n+1]$

Asymptotic complexity

We end this section with the asymptotic cost of isogenies. If ϕ is an ℓ^n-isogeny, then the cost of the computation depends clearly on the size and the smoothness of the degree. More precisely, the strategy output by Algorithm 3.3 is closely related to the size of the prime degree ℓ. Algorithm 3.3 finds an optimal trade-off between the number of scalar multiplications $[\ell]$ and the number of ℓ-isogenies. Asymptotically, Algorithm 2.1 computes a scalar multiplication $[\ell]$ in $O(\log_2(\ell))$ base-field multiplications. Algorithms for ℓ-isogenies have exponential complexity when ℓ increase. The recent work [BFLS20] of Bernstein, De Feo, Leroux and Smith relates isogeny computations with resultants and polynomial evaluations. Using the formulas of the group law in Montgomery coordinates together with a well-chosen index system, they reach the $\tilde{O}(\sqrt{\ell})$ asymptotic complexity. This work leads to improvements on practical cost of the key exchange of Section 3.3.2. The general complexity of an ℓ^n-isogeny can be estimated as follows:

- When ℓ is fixed and small, the cost of Vélu's formulas is roughly as large as scalar multiplications, and the optimal strategy is very close to Algorithm 3.2 whose cost is $O(n \log_2(n))$ base field multiplications.
- When ℓ becomes large, the cost of Vélu's formulas is exponentially larger than a scalar multiplication, and the optimal strategy is the comb strategy of Algorithm 3.1. The total cost is $\tilde{O}(n^2 \log_2(\ell) + n\sqrt{\ell})$. For a fixed ℓ, the cost is dominated by $O(n^2)$ base field multiplications.

3.3 Isogeny-based cryptography

Isogenies have been introduced in cryptography with a focus on ordinary curves in 1997 by Couveignes [Cou06]. Independently, Rostovstev and Stolbunov [RS06] studied a similar approach in 2006. The underlying protocol is a key exchange that we denote now CRS. It was not efficient and hence did not interest the cryptography community until the potential threat of quantum computers. De Feo and Jao obtain in 2011 a quantum-resistant Diffie–Hellman key exchange [JD11], leading to one of the NIST Post-Quantum competition candidate for key exchange, called SIKE [JAC+19]. More recently, isogenies of ordinary curves and $\mathbb{F}_p$-supersingular

Algorithm 3.4: ComputeIsogenyFromStrategy(s, P, Q)

Input. s a strategy,
P a point generating a kernel isogeny,
Q another point of the curve.
Output. R the image of Q by the isogeny of kernel $\langle P \rangle$.

$l \leftarrow \text{size}(s), i \leftarrow 0$
$\mathcal{Q} \leftarrow \{(l, P)\}$
$R \leftarrow Q$
while $\mathcal{Q}$ is not empty **do**
  $(h, T) \leftarrow \mathcal{Q}$.popLast()
  if $h = 1$ **then**
    $\phi \leftarrow \text{Vélu}(T)$
    $\mathcal{Q}' \leftarrow \{\}$
    while $\mathcal{Q}$ is not empty **do**
      $(h', T') \leftarrow \mathcal{Q}$.popFirst()
      Append $(h' - 1, \phi(T'))$ to $\mathcal{Q}'$
    end while
    $\mathcal{Q} \leftarrow \mathcal{Q}'$
    $R \leftarrow \phi(R)$
    $l = l - 1$
  else if $0 < s[i] < h$ **then**
    Append (h, T) to $\mathcal{Q}$
    Append $(h - s[i], [\ell^{s[i]}]T)$ to $\mathcal{Q}$
    $i \leftarrow i + 1$
  end if
end while
return R

(Apply ϕ at $\mathcal{Q}$ and reverse the queue)

curves have been studied in [DKS18, CLM+18]. In consequence, a key exchange has been designed and is called CSIDH. It uses supersingular curves defined over $\mathbb{F}_p$, and is closely related to the CRS original idea. This variant has led to many applications such as signatures [DG19, BKV19].

Isogeny-based cryptography has become interesting because of the following hard problem: given two isomorphism classes of elliptic curves, find an isogeny between them. Even if isogenies have a lot of structure, the best known algorithm for solving the problem for supersingular curves defined over $\mathbb{F}_{p^2}$ has exponential complexity. Sub-exponential algorithms apply in the case of ordinary curves and supersingular curves defined over $\mathbb{F}_p$. In the following sections, we describe three key exchange protocols in the three cases studied above: ordinary curves, supersingular curves defined over $\mathbb{F}_p$, and finally supersingular curves defined over $\mathbb{F}_{p^2}$.

3.3.1 The CRS key exchange and its improvements

Key exchanges based on isogenies of ordinary curves have been introduced with the works of Couveignes [Cou06] and Rostovstev and Stolbunov [RS06]. It can be summarized as a group action on a set of isomorphism classes of curves. More precisely, suppose that E is an ordinary elliptic curve defined over $\mathbb{F}_p$ of order $\mathcal{O}$ (so $\mathrm{End}(E) = \mathcal{O}$) representing an isomorphism class. Then, using the action of the class group $\mathrm{Cl}(\mathcal{O})$ (see Section 3.1.1), we obtain a key exchange as follows:

1. Alice chooses her secret ideal $\mathfrak{a} \in \mathrm{Cl}(\mathcal{O})$ and sends to Bob $E_\mathfrak{a} := \mathfrak{a} * E$.
2. Bob chooses his secret ideal $\mathfrak{b} \in \mathrm{Cl}(\mathcal{O})$ and sends to Alice $E_\mathfrak{b} := \mathfrak{b} * E$.
3. They share the common secret $E_{\mathfrak{ab}} := (\mathfrak{a} * \mathfrak{b}) * E = \mathfrak{a} * E_\mathfrak{b} = \mathfrak{b} * E_\mathfrak{a}$.

In order to compute the group action, Couveignes [Cou06] proposes to split the ideal as a product of small norm ideals, and Rostovstev and Stolbunov [RS06] compute the action of small ideals using isogenies. Practically, the action of an ideal $\mathfrak{a} = \mathfrak{l}_2\mathfrak{l}_3$ corresponds to an isogeny of degree 2 and an isogeny of degree 3. A key exchange is described in Figure 3.9, where Alice chooses $\mathfrak{a} = \mathfrak{l}_2\mathfrak{l}_3$ and Bob chooses $\mathfrak{b} = \mathfrak{l}_5\mathfrak{l}_7$.

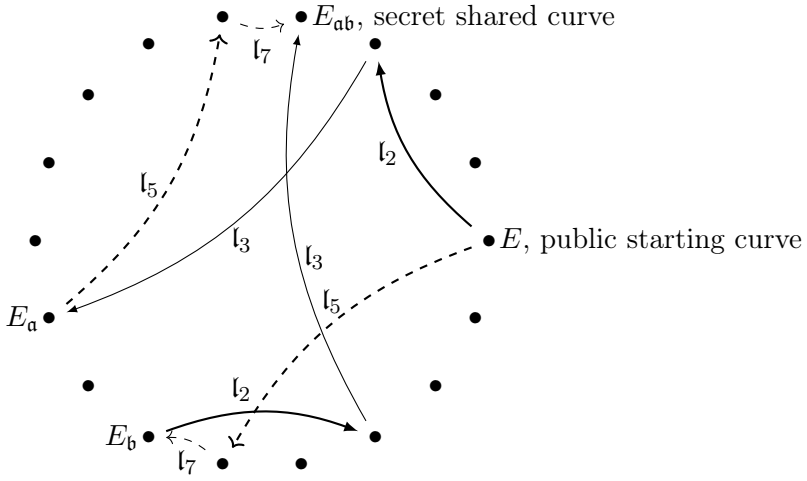


Figure 3.9: The CRS key exchange in the case of $\mathfrak{a} = \mathfrak{l}_2\mathfrak{l}_3$ and $\mathfrak{b} = \mathfrak{l}_5\mathfrak{l}_7$

De Feo, Kieffer and Smith [DKS18] improve the efficiency by selecting curves with a particular torsion over $\mathbb{F}_p$. Looking at an ideal $\mathfrak{a} = \prod_i \mathfrak{l}_i^{e_i}$, the action of the $\mathfrak{l}_i$ can be efficiently computed

using Vélu's formulas if the norm of $\mathfrak{l}_i$ divides $\#E(\mathbb{F}_p)$ (or its quadratic twist). Thus, they select an ordinary curve whose order (or its quadratic twist order) has as many small prime factors as possible. They obtain an elliptic curve defined over $\mathbb{F}_p$ with

$$p = 7 \left(\prod_{2 \leq \ell \leq 380,\ \ell \text{ prime}} \ell \right) - 1.$$

The primes of the set $S = \{3, 5, 7, 11, 13, 17, 103, 523, 821, 947, 1723\}$ divide either $\#E(\mathbb{F}_p)$ or $\#E^t(\mathbb{F}_p)$. From an ideal $\mathfrak{a}$ factoring into $\mathfrak{a} = \prod_i \mathfrak{l}_i^{e_i}$,

- if $N(\mathfrak{l}_i)$ is in S, then the action of $\mathfrak{l}_i$ is computed using Vélu's formulas (and if $N(\mathfrak{l}_i)$ divides $p+1$, $\bar{\mathfrak{l}}_i$ can also be computed at no additional cost),
- Otherwise, the isogeny corresponding to $\mathfrak{l}_i$ is computed using another algorithm called the *Elkies step*.

Unfortunately, the latter is more expensive and the author of [DKS18] are not able to generate a curve with enough small prime divisors so that it contains all $\mathfrak{l}_i$ norms. Finally, the key exchange is computed in roughly 5 minutes for a 128-bit security level. We refer to [DKS18] for more details on this protocol.

In a security point of view, the CRS system relies on the assumptions that given two curves of endomorphism ring $\mathcal{O}$, it is hard to find a (smooth degree) isogeny $E \to E'$, and that the curves are really uniformly sampled in $\mathrm{Ell}(\mathcal{O})$. The protocol is threatened by a quantum algorithm related to the dihedral hidden subgroup problem, based on the work of Kuperberg [Kup05] and Regev [Reg04]. Its complexity is subexponential ($L_q(1/2, c)$ for a constant $c > 0$). The practical cost is not clearly determined, and is an active field of research. In the same time, Martindale et al. [CLM+18] obtain a similar protocol using supersingular elliptic curves defined over $\mathbb{F}_p$, called CSIDH.

3.3.2 CSIDH

Chronologically, CSIDH [CLM+18] appears after the SIDH key exchange of Section 3.3.3, but structurally, it is closely related to the CRS primitive. The efficiency of the key exchange of [DKS18] is affected by the fact that the curve order does not have enough small divisors (and so Vélu's formulas do not always apply). Instead of working over ordinary curves, Martindale et al. choose to work with supersingular curves defined over a prime field $\mathbb{F}_p$. For these curves, the trace (over $\mathbb{F}_p$) is 0 and its order is $\#E(\mathbb{F}_p) = p + 1$. Fixing a prime $p = \prod_i \ell_i - 1$, the factors of the curve order are precisely the ℓ_i. Using enough primes ℓ_i, Vélu's formulas can be computed for all ideals as desired.

Supersingular curves defined over $\mathbb{F}_{p^2}$ do not have a class group action so it does not make sense to consider a protocol as in Section 3.3.1. However, from the considerations of Section 3.1.2, the structure of the graph of curves and isogenies defined *only* over $\mathbb{F}_p$ is very similar to the case of ordinary curves. Thus, $\mathrm{End}_p(E)$ is either $\mathbb{Z}[\sqrt{-p}]$ or $\mathbb{Z}[\frac{1+\sqrt{-p}}{2}]$, and if $p + 1 = \prod_i \ell_i$, one expects that the ideals of $\mathrm{Cl}(\mathrm{End}_p(E))$ split as products of prime-norm ideals $\prod_i \mathfrak{l}_i^{e_i}$ so that the action is efficient with Vélu's formulas. The security of CSIDH relies on the same assumptions as in the ordinary case, and so it leads to a subexponential complexity. The authors of [CLM+18] provide an example of setting for a 128-bit security level. Let $p = 4 \cdot l_1 \cdots l_{74} - 1$ where l_1 through l_{73} are the smallest 73 odd primes and $l_{74} = 587$. Then, the action of $\mathrm{Cl}(\mathbb{Z}[\sqrt{-p}])$ leads to an

efficient key exchange. Finally, the CSIDH key exchange is much more efficient than the protocol with ordinary curves in [DKS18], and leads to a computation in roughly 100ms.

CSIDH also leads to other cryptographic applications such as signatures. Briefly, the class group order $\mathrm{Cl}(\mathbb{Z}[\sqrt{-p}])$ has been computed in [BKV19] and it has been proved that it is a cyclic group leading to an efficient Fiat-Shamir signature called CSI-FiSh. The CSIDH construction has many variants. For example, [CD20] works on the surface of the volcano in order to improve the efficiency. One of the functions designed in Chapter 7 is closely related to the CSIDH setting, where the isogeny walk is entirely done over the $\mathbb{F}_p$-graph.

3.3.3 SIDH

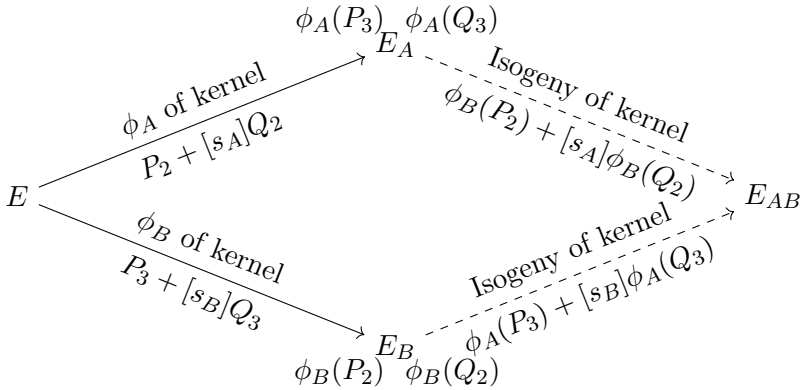
The SIDH key exchange [JD11] has nothing to do with CSIDH in the sense that it computes isogenies of supersingular curves without staying on an invariant order of the full endomorphism ring. Thus, it does not use a commutative group action and the security of the scheme is based on a structure which is not threatened by subexponential attacks. In consequence, the public key sizes are smaller for SIDH than for CSIDH. We investigate the pros and the cons of SIDH in this section.

Let E be an elliptic curve defined over $\mathbb{F}_p$, where $p = 2^e3^f - 1$, and such that $\#E(\mathbb{F}_{p^2}) = (2^e3^f)^2$. Choose P_2, Q_2 a basis of $E[2^e]$, and P_3, Q_3 a basis of $E[3^f]$. In this context, the SIDH key exchange proceeds as follows:

1. Alice chooses her secret integer $s_A \in \mathbb{Z}/2^e\mathbb{Z}$ and deduces her isogeny $\phi_A : E \to E_A$ whose kernel is $P_2 + s_AQ_2$. She computes her public key which is composed of $j(E_A)$, and the additional points $\phi_A(P_3)$, $\phi_A(Q_3)$.

2. Bob chooses his secret integer $s_B \in \mathbb{Z}/3^f\mathbb{Z}$ and deduces his isogeny $\phi_B : E \to E_B$ whose kernel is $P_3 + s_BQ_3$. He computes his public key which is composed of $j(E_B)$, and the additional points $\phi_B(P_2)$, $\phi_B(Q_2)$.

3. Alice (resp. Bob) computes the isogeny $E_B \to E_{AB}$ (resp. $E_A \to E_{BA}$) whose kernel is $\phi_B(P_2) + s_A\phi_B(P_3)$ (resp. $\phi_A(P_3) + s_B\phi_A(Q_3)$). They end at the same isomorphic class of curve and share the common secret $j(E_{AB})$.



Security estimation. The security of the SIDH key exchange is based on the fact that it is hard to recover an isogeny given two isomorphism classes of curves on the graph. More precisely, given two supersingular elliptic curves, the best (time complexity) algorithm for finding

a path of isogenies between the curves is a meet-in-the-middle algorithm. Its complexity is exponential in the size of the graph. Using classical computers, the asymptotic complexity is $O(\sqrt{\max(2^e, 3^f)}) = O(\sqrt[4]{p})$. On a quantum computer, the meet-in-the-middle attack can be done in $O(\sqrt[3]{\max(2^e, 3^f)}) = O(\sqrt[6]{p})$ [Tan09], but also requires $O(\sqrt[6]{p})$ storage. A conservative estimate for the parameter size is to choose a prime p of bitlength 6 times the security level. However, from a practical point of view, this massive storage is not feasible for a 128-bit security level. The van Oorschot-Wiener golden collision finding algorithm [vW99] would have a better efficiency [ACC+19] and in conclusion, $\log_2(p) = 448$ is sufficient in order to reach a 128-bit security level.

Remark 3.11. Note that the additional data sent in the SIDH key exchange is not exploited in the cryptanalysis above. It is still an open question to know if these data can leak crucial information on the secret isogeny.

Key encapsulation. The SIDH key exchange is secure only with ephemeral keys. Alice is able to get information on Bob's secret by trying different well-chosen *fake* public keys [GPST16]. Moreover, there is currently no known method for public key validation in the case of SIDH. In consequence, SIDH with static public keys is not recommended and a key-encapsulation is preferred. The isogeny-based key encapsulation is called SIKE [JAC+19] and proceeds as follows (where H and H' are hash functions with respective codomain $\mathbb{Z}/3^f\mathbb{Z}$ and $\{0,1\}^B$, B fixed):

1. Alice chooses her secret integer $s_A \in \mathbb{Z}/2^e\mathbb{Z}$ and deduces her isogeny $\phi_A : E \to E_A$ whose kernel is $P_2 + s_A Q_2$. She computes her public key $P_A = \{j(E_A), \phi_A(P_3), \phi_A(Q_3)\}$.

2. Bob uses Alice's public key and an randomly chosen message m to get his secret $s_B = H(P_A, m) \in \mathbb{Z}/3^f\mathbb{Z}$, and deduces his isogeny $\phi_B : E \to E_B$ whose kernel is $P_3 + s_B Q_3$. He computes the isogeny $E_A \to E_{BA}$ whose kernel is $\phi_A(P_3) + s_B\phi_A(Q_3)$ and sends to Alice $H'(j(E_{BA})) \oplus m$ together with his public key $P_B = \{j(E_B), \phi_B(P_2), \phi_B(Q_2)\}$.

3. Alice uses P_B and her secret key to compute $j(E_{AB})$ and then $H'(j(E_{AB}))$ in order to recover Bob's initial random value m. She can also recompute Bob's secret key and check that Bob has not acted maliciously.

SIKE [JAC+19] provides explicit parameters to reach different levels of security using a key encapsulation.

3.3.4 An open problem

We end this chapter with an open problem related to the generation of *random* supersingular curves. More precisely, we would like to be able to generate a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$ whose endomorphism ring is unknown.

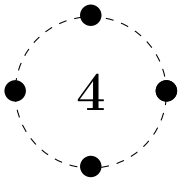
For a fixed prime p, we have obtained the number of $\bar{\mathbb{F}}_p$-isomorphism classes of supersingular curves defined over $\mathbb{F}_{p^2}$. For a fixed trace $t = 2p$, this number is $\lfloor p/12 \rfloor + \epsilon_p$ for $\epsilon_p \in \{0, 1, 2\}$. More generally, there are $O(p)$ $\bar{\mathbb{F}}_p$-isomorphism classes of supersingular curves among the $O(p^2)$ classes of curves defined over $\mathbb{F}_{p^2}$ (represented as j-invariants in $\mathbb{F}_{p^2}$). In consequence, a curve chosen at random (for instance by choosing randomly $a, b \in \mathbb{F}_{p^2}$ to define a curve $E_{a,b}$) is supersingular with probability $\approx 1/p$ which is negligible when p is of cryptographic size.

In all the protocols we described above such as SIDH or CSIDH, the initial elliptic curve is a curve of *small* discriminant. It is often simplified by choosing a curve with $j = 0$ or 1728 which corresponds to $-D = -3$ or -4, but one could use the CM method in order to get another

supersingular curve. By fixing a CM curve as in Section 2.5.4 and reducing modulo a prime p, we get a curve which is supersingular or ordinary depending on the prime decomposition in the order $\mathcal{O}_D$ where $-D$ is the discriminant of the curve. Inert and splitting primes are in the same proportion and so we are able to generate supersingular curves in this way, but only with a small discriminant (recall that the largest computed Hilbert class polynomials have discriminant $\approx 10^{15}$ [ES10]). Then, from a small discriminant, it is possible to determine (and evaluate) all the endomorphisms of $\mathrm{End}(E)$. See Chapter 6 for details.

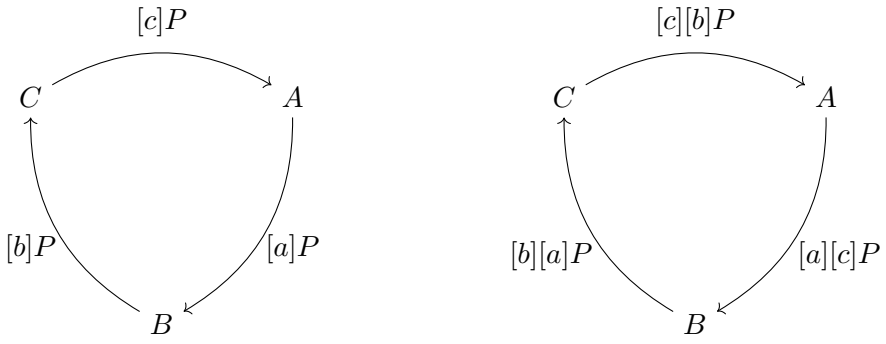
Finally, the only way to get a supersingular curve of unknown endomorphism ring is to begin with a curve such as a CM curve, and compute a random walk on the isogeny graph. However, the knowledge of the walk brings information that lets us compute once again $\mathrm{End}(E)$. The only way to get a supersingular curve E without knowing $\mathrm{End}(E)$ is to "forget" the walk, which means either to compute the walk using multiparty computation, or give to compute the walk by a *third party*, which should be trusted.

Hence, the problem of generating a supersingular elliptic curve without knowing its endomorphism ring is still an open problem. A potential solution would lead to interesting properties for the tools we introduce in Chapter 7.



Pairing-friendly curves

In this chapter, we consider new situations where the elliptic curve DLP is not sufficient. For instance, we consider the straightforward generalization of the Diffie–Hellman key exchange with three participants, called tripartite key exchange. Using the classical Diffie–Hellman over an elliptic curve with a point P of large prime order, Alice, Bob and Charlie can share a common key in two rounds. First, from her secret a, Alice sends the point $[a]P$ to Bob. Then, using the point $[c]P$ received from Charlie, Alice sends $[a][c]P$ to Bob. The two other participants send similar points so that they share a common secret $[abc]P$. This protocol is explained in Figure 4.1.



First round. Second round.

Figure 4.1: Two-round Diffie–Hellman with three participants.

In this chapter, we introduce a mathematical object that lets us generalize this key exchange in one round. This tool is called a *pairing* and was introduced in the early 2000's by Joux [Jou04]. In 2004, Boneh et al. obtained short signatures [BLS04], and more recently, new applications have been proposed, e.g. zero-knowledge proofs [BCCT12] used in the Zcash cryptocurrency and electronic voting.

We first define what a pairing is, and explain how to compute variants on an elliptic curve. Pairing-based cryptography relies on the hardness of the discrete logarithm problem in three groups: the multiplicative group of a finite field (studied in Chapter 1), and two groups of

points of an elliptic curve (studied in Chapter 2). We present here methods for generating *pairing-friendly* elliptic curves, i.e. elliptic curves for which a pairing is efficiently computable. Families of pairing-friendly curves induce a discrete logarithm over a finite field threatened by the recent NFS variants presented in Chapter 1. Hence, for these curves, the size of the parameters needs to be increased in order to obtain a target security level. In consequence, we present how to generate new elliptic curves that are immune to these attacks. Finally, we estimate the cost of a pairing computation. We clarify the trade-off between the available optimizations and the size of the parameters.

Summary

4.1 Pairing constructions

4.1.1 Definition

We begin with the general definition of a pairing, with general groups. In practice, pairings are instantiated over an elliptic curve and a finite field.

Definition 4.1 (Pairing)**.** *A pairing is an application*

$$\begin{array}{rccc} e: & G \times G' & \longrightarrow & G'' \\ & (g, g') & \longmapsto & e(g, g') \end{array}$$

between three groups G, G' and G'' of prime exponent that is

- non-degenerate*:*

$$e(g, g') = 1_{G''} \text{ for all } g \in G \Longrightarrow g' = 1_{G'},$$
$$e(g, g') = 1_{G''} \text{ for all } g' \in G' \Longrightarrow g = 1_{G}.$$

- bilinear*: for $g, h \in G$ and $g', h' \in G'$,*

$$e(g \cdot h, g') = e(g, g') \cdot e(h, g') \text{ and } e(g, g' \cdot h') = e(g, g')e(g, h').$$

- efficiently computable *(computing the application has a polynomial complexity and is efficient in practice).*

Pairings have been introduced with an instantiation using an elliptic curve and a finite field. In this context, G and G' are groups of points of an elliptic curve and G'' will be a group of r-th roots of unity in a finite field. With this notation, Alice, Bob and Charlie can share a common secret in one round using the protocol explained in Figure 4.2. The group law is denoted additively for G and G', and multiplicatively for G''. From the bilinearity of the pairing, Alice computes $e([b]P, [c]Q)^a = e(P, Q)^{abc}$. Using similar computations, Bob and Charlie are also able to compute $e(P, Q)^{abc}$, the common secret key.

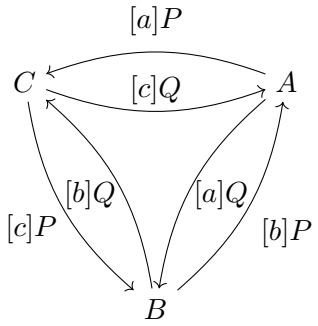


Figure 4.2: Tripartite one-round Diffie–Hellman.

Several protocols and applications were introduced few years later. We mention here the BLS signature [BLS04] which produces short signatures. As in Chapter 2, we compute scalar multiplications of points of an elliptic curve which has particular properties (these curves are called *pairing-friendly* and this definition is detailed in Section 4.2). Thus, let E be a pairing-friendly elliptic curve defined over a prime field $\mathbb{F}_p$. The secret key in the BLS signature is a scalar $s < r$ where r is the exponents of the three groups involved in the pairing above. The public key is a pair of points $(P, [s]P) \in G$. To sign a message m, the signer computes a hash $Q = H(m) \in G'$ and gives back the signature $[s]Q$. The verifier then checks that $e(P, [s]Q) = e([s]P, Q)$. We now investigate the details on the three groups G, G' and G''. In practice, these groups depend on the choice of the pairing, but the signature is always a point of the curve, and so it leads to short signatures (compared to ECDSA or RSA signatures). These signatures benefit from other properties (such as the signature aggregation [BGLS03]) but we do not detail them here. BLS signatures are closely related to the framework we introduce in Section 7.3.

Given a field $\mathbb{F}_q$ and a prime integer r such that $\gcd(r, q) = 1$, we denote μ_r the group of r-th roots of unity in $\bar{\mathbb{F}}_q$. This group is of exponent r, defined over a finite extension of $\mathbb{F}_q$. The following theorem gives an equivalent definition of the embedding degree, defined in Definition 2.13.

Theorem 4.2 (Balasubramanian and Koblitz, [Gal05, page 192])**.** *Let E be an elliptic curve defined over $\mathbb{F}_q$ and let r be a prime dividing $\#E(\mathbb{F}_q)$. Suppose that r does not divide $(q-1)$ and that $\gcd(r, q) = 1$. Then, k is the embedding degree of E with respect to r if, and only if, $\mathbb{F}_{q^k}$ is the smallest extension of $\mathbb{F}_q$ containing μ_r.*

Let E be an elliptic curve defined over $\mathbb{F}_q$ with a prime integer r coprime to q dividing $\#E(\mathbb{F}_q)$. Let k be the embedding degree with respect to r. We present groups of exponents r useful for defining elliptic-curve-based pairings:

- The r-torsion subgroup $E[r]$, isomorphic to $\mathbb{Z}/r\mathbb{Z} \times \mathbb{Z}/r\mathbb{Z}$.
  By definition of the embedding degree, $E[r]$ is fully rational over $\mathbb{F}_{q^k}$: $E(\mathbb{F}_{q^k})[r] = E[r]$. We will use the notations introduced in Section 2.4: $E[r] = \mathbb{G}_1 \times \mathbb{G}_2$ where $\mathbb{G}_1 = E[r] \cap \ker(\pi - [1])$ and $\mathbb{G}_2 = E[r] \cap \ker(\pi - [q])$.

- The quotient group $E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k})$, isomorphic to $\mathbb{Z}/r\mathbb{Z} \times \mathbb{Z}/r\mathbb{Z}$.
  In a cryptographic context, cosets can be represented by the r-torsion points. Indeed, there is no point of order r^i $(i > 1)$ over $E(\mathbb{F}_{q^k})$. Hence, by the finite abelian group theorem, there is an isomorphism $\iota : E(\mathbb{F}_{q^k}) \xrightarrow{\sim} H \times \mathbb{Z}/r\mathbb{Z} \times \mathbb{Z}/r\mathbb{Z}$, where H is a group of order coprime to r. Now, if $Q \in E(\mathbb{F}_{q^k})$, $\iota(Q) = (Q_0, Q_1, Q_2)$ and so $Q - \iota^{-1}((0, Q_1, Q_2)) = \iota^{-1}((Q_0, 0, 0)) \in E(\mathbb{F}_{q^k})[r]$. Finally, in the quotient group $E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k})$, $Q \sim \iota^{-1}((0, Q_1, Q_2))$ which is a point of r-torsion.

- The quotient group $\mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^r$, isomorphic to $\mathbb{Z}/r\mathbb{Z}$.
  This group is also isomorphic the group of r-th roots of unity, by the map $x \mapsto x^{(q^k-1)/r}$, and is denoted $\mathbb{G}$ in the following.

Before going into the details of pairing constructions, we need to introduce the Miller function, which will be one of the main computations of this chapter.

4.1.2 The Miller function

The Miller function is a function that, given a point, returns a function of the curve that meets certain conditions. We will not need to compute Miller functions, but only their evaluations at points.

Definition 4.3 (Miller function)**.** *Let $Q \in E(\mathbb{F}_{q^k})$ and $n \in \mathbb{Z}$. The Miller function is an element $f_{n,Q} \in \mathbb{F}_{q^k}(E)$ such that* $\mathrm{div}(f) = n(Q) - ([n]Q) - (n-1)(0_E)$.

Note that $f_{n,Q}$ is determined uniquely up to a multiplication by an element of $\mathbb{F}_{q^k}^*$. Such a function can be computed using the *Miller algorithm*, presented in Algorithm 4.1. The main idea of the algorithm is to compute the point $[n]Q$ and save the group law line computations. Recall that from $[i]Q$ and $[j]Q$, the point $[i+j]Q$ is computed using the line $\ell_{[i]Q,[j]Q}(x, y)$ of Section 2.2, and a vertical line $v_{[i+j]Q}(x, y) = x - x_{[i+j]Q}$. Indeed, we observe that

$$\mathrm{div}(f_{i+j,Q}) = \mathrm{div}(f_{i,Q}) + \mathrm{div}(f_{j,Q}) + ([i]Q) + ([j]Q) - ([i+j]Q) - (0_E)$$

and the extra term $([i]Q) + ([j]Q) - ([i+j]Q) - (0_E)$ is closely related to the computation of $[i+j]Q$ from $[i]Q$ and $[j]Q$:

$$\mathrm{div}(\ell_{[i]Q,[j]Q}) = ([i]Q) + ([j]Q) + (-[i+j]Q) - 3(0_E),$$

$$\mathrm{div}(v_{[i+j]Q}) = ([i+j]Q) + (-[i+j]Q) - 2(0_E).$$

Finally, we obtain

$$\mathrm{div}(f_{i+j,Q}) = \mathrm{div}(f_{i,Q}) + \mathrm{div}(f_{j,Q}) + \mathrm{div}(\ell_{[i]Q,[j]Q}) - \mathrm{div}(v_{[i+j]Q})$$

and so the Miller functions satisfy (up to a constant multiplication)

$$f_{i+j,Q} = f_{i,Q} f_{j,Q} \frac{\ell_{[i]Q,[j]Q}}{v_{[i+j]Q}}. \tag{4.1}$$

Using Equation (4.1), we can compute $f_{n,Q}$ using a square-and-multiply-like algorithm. Instead of computing the whole function $f_{n,Q}$ (which has very large coefficients in practice) and then evaluating it at a point P, we compute and evaluate at the same time during the algorithm. Thus, we compute only multiplications over $\mathbb{F}_{q^k}$ instead of $\mathbb{F}_{q^k}(E)$. At the end, the Miller function evaluation is very similar to an exponentiation in $\mathbb{F}_{q^k}$. Algorithm 4.1 details the computation of $f_{n,Q}$, and its evaluation at the same time at a point P.

Algorithm 4.1: MillerLoop(n, P, Q) – Compute $m = f_{n,Q}(P)$.

Input. n an integer,
P a point of the curve E.
Output. $Q \in E[r]$.

$m \leftarrow 1$
$S \leftarrow Q$
for b from the second most significant bit of n to the least **do**
  $m \leftarrow m^2 \cdot \ell_{S,S}(P)/v_{2S}(P)$
  $S \leftarrow [2]S$
  if $b = 1$ **then**
    $m \leftarrow m \cdot \ell_{S,Q}(P)/v_{S+Q}(P)$
    $S \leftarrow S + Q$
  end if
end for
return m

Although the Miller function evaluation in Algorithm 4.1 is more expensive than an exponentiation x^n in $\mathbb{F}_{q^k}$, the complexity of both algorithms is linear in the size of n. Depending on the curve structure, optimizations are possible. We investigate them in Section 4.3. We are now in a position to define elliptic-curve-based pairings, using the groups of exponent r of Section 4.1.1.

4.1.3 The Weil pairing

Let $P, Q \in E[r]$. Let $f_{r,P}$ (resp. $f_{r,Q}$) $\in \mathbb{F}_{q^k}(E)$ be the Miller function defined in Definition 4.3. Note that $[r]P = [r]Q = 0_E$ so $\mathrm{div}(f_{r,P}) = r(P) - r(0_E)$ and $\mathrm{div}(f_{r,Q}) = r(Q) - r(0_E)$. Let D_P (resp. D_Q) be a degree zero divisor equivalent to $(P) - (0_E)$ (resp. $(Q) - (0_E)$), defined over $\mathbb{F}_{q^k}$, whose support is disjoint with $\{Q, 0_E\}$ (resp $\{P, 0_E\}$); one can simply sample $S \in E(\mathbb{F}_{q^k})$ until $D_P := (P + S) - (S)$ suits.

The Weil pairing is defined using the r-torsion group $E[r]$ and the group of r-th roots of unity $\mu_r \subset \mathbb{F}_{q^k}^*$.

Theorem 4.4 (Weil pairing [Gal05, page 185])**.** *The map*

$$\begin{array}{rccc} e_r : & E(\mathbb{F}_{q^k})[r] \times E(\mathbb{F}_{q^k})[r] & \longrightarrow & \mu_r \\ & (P, Q) & \longmapsto & f_{r,P}(D_Q)/f_{r,Q}(D_P) \end{array}$$

is non-degenerate and bilinear.

As defined in Section 2.2, for $D_Q = (Q+S)-(S)$, the value of $f_{r,P}(D_Q)$ is $f_{r,P}(Q+S)/f_{r,P}(S)$. Thus, the Weil pairing is computed by evaluating Miller functions at points instead of divisors. Using four times Algorithm 4.1, we are able to compute the Weil pairing. We investigate the cost of Algorithm 4.1 and its optimizations in Section 4.3. It applies in particular for the Weil pairing. In practice, these Miller functions are expensive. In consequence, the *Tate pairing* or its variants are preferred for efficiency. We present these new pairings in the next sections.

Remark 4.5. The Weil pairing is also defined for $r = \ell^e$ with a small prime ℓ. Moreover, if $P \neq 0_E$, then Q lies in the subgroup generated by P if, and only if, $e_{\ell^e}(P,Q)^{\ell^{e-1}} = 1$. From this result, one can obtain the matrix representation of an endomorphism f on the ℓ^e-torsion using the Weil pairing: from a basis $\{P_1, P_2\}$ of $E[\ell^e]$,

$$f(P_j) = a_{1,j}P_1 + a_{2,j}P_2 \qquad j \in \{1,2\}$$

and the coefficients $(a_{i,j})_{i,j}$ can be obtained by computing discrete logarithms:

$$a_{1,1} = \mathrm{DL}_{e_r(P_1,P_2)}(e_r(f(P_1),P_2)), \qquad a_{2,1} = \mathrm{DL}_{e_r(P_2,P_1)}(e_r(f(P_1),P_1)),$$

$$a_{1,2} = \mathrm{DL}_{e_r(P_1,P_2)}(e_r(f(P_2),P_2)), \qquad a_{2,2} = \mathrm{DL}_{e_r(P_2,P_1)}(e_r(f(P_2),P_1)).$$

Note that these computations are efficient in a group of smooth order ℓ^e.

4.1.4 The Tate pairing

The Tate pairing is slightly more complicated to define because it uses the quotient groups $E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k})$ and $\mathbb{F}^*_{q^k}/(\mathbb{F}^*_{q^k})^r$. In practice, we will see that representatives of cosets in these two groups can also be chosen in $E[r]$ and μ_r. Moreover, we will see that this pairing is more efficient than the Weil pairing.

Let $P \in E[r]$ as in the Weil pairing, together with the Miller function $f_{r,P}$. Let $Q \in E(\mathbb{F}_{q^k})$ and denote D_Q a degree zero divisor equivalent to $(Q) - (0_E)$ whose support is disjoint with $\{P, 0_E\}$. In the same way as for the Weil pairing, one can compute $f_{r,P}(D_Q) \in \mathbb{F}_{q^k}$. Moreover, for $Q' \in Q + rE(\mathbb{F}_{q^k})$, one can prove that $f_{r,P}(Q') \equiv f_{r,P}(Q) \bmod (\mathbb{F}^*_{q^k})^r$. From this fact, we are able to define the Tate pairing.

Theorem 4.6 (Tate pairing [Gal05, page 185])**.** *The map*

$$\begin{array}{rccc} \tilde{t}_r : & E(\mathbb{F}_{q^k})[r] \times E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k}) & \longrightarrow & \mathbb{F}^*_{q^k}/(\mathbb{F}^*_{q^k})^r \\ & (P,Q) & \longmapsto & f_{r,P}(D_Q) \end{array}$$

is non-degenerate and bilinear.

The Weil and the Tate pairings have several properties [Gal05, page 187]. In particular, they are compatible with isogenies. If $\phi : E \to E'$ is an isogeny with dual $\hat{\phi}$, $P \in E(\mathbb{F}_{q^k})[r]$ and $Q \in E'(\mathbb{F}_{q^k})[r]$, then

$$e_r(\phi(P), Q) = e_r(P, \hat{\phi}(Q)).$$

From Section 4.1.1, we can choose $Q \in E[r]$ in order to represent a coset in $E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k})$. The Miller algorithm computes $f_{r,P}(D_Q) \in \mathbb{F}^*_{q^k}/(\mathbb{F}^*_{q^k})^r$. In cryptographic applications, we expect that two choices of coset representatives $Q \sim Q'$ in $E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k})$ lead to the same coset representative in $\mathbb{F}^*_{q^k}/(\mathbb{F}^*_{q^k})^r$. To get this property, we define the *reduced* Tate pairing, which computes a coset representative as a value of the group of r-th roots of unity:

Definition 4.7 (Reduced Tate pairing)**.** *The reduced Tate pairing is the non-degenerate bilinear application*

$$\begin{array}{rccc} t_r: & E(\mathbb{F}_{q^k})[r] \times E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k}) & \longrightarrow & \mu_r \\ & (P,Q) & \longmapsto & f_{r,P}(D_Q)^{(q^k-1)/r}. \end{array}$$

We will investigate the cost of raising to the power $(q^k-1)/r$ as well as the optimizations of Algorithm 4.1 in Section 4.3. As for the Weil pairing, if $D_Q = (Q+S)-(S)$, then $f_{r,P}(D_Q) = f_{r,P}(Q+S)/f_{r,P}(S)$ is computed using Algorithm 4.1 twice. Nevertheless, we will see later that in the case of the *reduced* Tate pairing, this can be done using only *one* computation of Algorithm 4.1.

Finally, we introduce a variant of the Tate pairing.

4.1.5 The ate pairing

We present the *ate pairing* following [HSV06]. Raising t_r to the power m for an integer m coprime to r corresponds to a permutation of the roots of unity, and hence leads to another pairing which is a *variant* of the Tate pairing. Moreover, one can prove that

$$f_{ab,Q} = f_{a,Q}^b \cdot f_{b,[a]Q}. \tag{4.2}$$

From Equation (4.2) and using $f_{m,[r]P} = f_{m,0_E} = 1$, we obtain that $t_r(P,Q)^m = f_{mr,P}(D_Q)^{(q^k-1)/r}$. Finally, choosing a particular integer m and swapping P and Q, this pairing can be efficiently computed. Let T be an integer that is a k-th root of unity mod r. By definition of the embedding degree, $q^k = 1 \bmod r$ and so we can write $T = q^j \bmod r$ for a given integer j. Let $d = \gcd(k,j)$. Remark that r divides $T^{k/d}-1$ and define $m = (T^{k/d}-1)/r$. Then,

$$t_r(Q,P)^m = f_{T^{k/d}-1,Q}(D_P)^{(q^k-1)/r} = f_{T^{k/d},Q}(D_P)^{(q^k-1)/r}.$$

Using Equation (4.2) together with the fact that $[T]Q = [q^j]Q$, we obtain

$$f_{T^{k/d},Q} = f_{T,Q}^{T^{k/d-1}} \cdot f_{T,[q]Q}^{T^{k/d-2}} \cdots f_{T,[q^{k/d-1}]Q} = f_{T,Q}^{\sum_{i=0}^{k/d-1} T^{k/d-1-i}q^{ij}}.$$

As $T = q^j \bmod r$, this last exponent modulo r is $q^{j(k/d-1)}k/d$, which is not a multiple of r (otherwise, k would not be the embedding degree with respect to r). Thus, this exponentiation is a permutation of the r-th roots of unity in $\mathbb{F}_{q^k}$ and Hess, Smart and Vercauteren [HSV06] define the reduced ate pairing:

$$\begin{array}{rccc} A_{r,T}: & \mathbb{G}_2 \times \mathbb{G}_1 & \longrightarrow & \mu_r \\ & (Q,P) & \longmapsto & f_{T,Q}(D_P)^{(q^k-1)/r}, \end{array}$$

for a k-th root of unity T.

4.1.6 Optimal pairings

Weil, Tate and ate pairing efficiency depends on the size of the Miller loop. As T is a k-th root of unity, writing $T = q^j \bmod r$ and $d = \gcd(k,j)$, r divides $\Phi_{k/d}(T)$. Hence, it gives a lower bound for the size of T: $\log_2(T) \geq \log_2(r)/\varphi(k/d) \geq \log_2(r)/\varphi(k)$, where φ is the Euler totient function. From this bound, Vercauteren defines *optimal pairings* in [Ver10].

Definition 4.8 (Optimal pairing, [Ver10, page 5]). *Let $e : G \times G' \to G''$ be a pairing with $|G| = |G'| = |G''| = r$ and $G'' \subset \mathbb{F}_{q^k}$. Then the pairing is called an optimal pairing if it can be computed in $\log_2(r)/\varphi(k) + \epsilon(k)$ basic Miller iterations, with $\epsilon(k) \leq \log_2(k)$.*

From this definition, the ate pairing can be turned into an optimal ate pairing by choosing T with small coordinates in base q.

Theorem 4.9 ([Ver10, page 6]). *Let $T = mr$ with $r \not\equiv 0 \bmod m$ and $T = \sum_{i=0}^{l} c_i q^i$. Then,*

$$\begin{aligned} a_{r,[c_0,\ldots,c_l]} : \quad \mathbb{G}_2 \times \mathbb{G}_1 &\longrightarrow \mu_r \\ (Q, P) &\longmapsto \left(\prod_{i=0}^{l} f_{c_i,Q}^{q^i}(D_P) \cdot \prod_{i=0}^{l-1} \frac{\ell_{[s_{i+1}]Q,[c_i q^i]Q}(P)}{v_{[s_i]Q}(P)} \right)^{(q^k-1)/r} \end{aligned}$$

with $s_i = \sum_{j=i}^{l} c_j q^j$, defines a pairing if $mkq^{k-1} \not\equiv ((q^k - 1)/r) \cdot \sum_{i=0}^{l} i c_i q^{i-1} \bmod r$.

In practice, we use elliptic curves for which k-th roots of unity have only few non-zero coordinates in base q. We will see in the next section several examples of curves together with their optimal ate version. For instance, if an elliptic curve has a small trace t, an optimal pairing can be obtained from choosing $T = t - 1$ (recall that $q + 1 - t = 0 \bmod r$ so $t - 1$ is a primitive k-th root of unity).

4.2 Pairing-friendly elliptic curves

In this section we explain how to generate pairing-friendly elliptic curves, i.e. elliptic curves for which a pairing is efficiently computable. From now on, we focus on elliptic curves defined over a prime field denoted $\mathbb{F}_p$. We begin this section with curves that are *not* pairing-friendly.

4.2.1 Elliptic curves not designed for pairings

Random elliptic curves. Given a random elliptic curve defined over $\mathbb{F}_p$ (one can sample $a, b \in \mathbb{F}_p$ until $4a^3 + 27b^2 \neq 0$, and choose $E_{a,b}$), its embedding degree k with respect to a prime divisor r of $\#E_{a,b}(\mathbb{F}_p)$ is roughly of size $\log_2(r)$. Recall that $\log_2(r) \geq 256$ for a 128-bit security target. Thus, the target finite field $\mathbb{F}_{p^k}$ is too large for practical computations. Random elliptic curves have huge embedding degrees and are hence *not* pairing-friendly.

Elliptic curves defined over a small characteristic field. Elliptic curves defined over a small characteristic field are *not recommended*: an algorithm for computing the discrete logarithm problem in small characteristic in quasi-polynomial time was proposed in 2014 [BGJT14]. This algorithm was heuristic until 2019, when Kleinjung and Wesolowski proved it [KW19]. In consequence, in order to reach a security level, the size of the finite field $\mathbb{F}_{p^k}$ needs to be much larger than in the case of large characteristic. For efficient pairing-based applications, we look for elliptic curves defined over a large characteristic prime field $\mathbb{F}_p$.

Supersingular elliptic curves. The first suggested pairings in [Jou04] used supersingular elliptic curves, because they were the first known way to produce curves with a small embedding degree. More precisely, we have seen in Section 2.4 that these curves have an embedding degree $k = 2$ or 3 in large characteristic. The target finite field is $\mathbb{F}_{p^2}^*$ or $\mathbb{F}_{p^6}^*$, and from the considerations of Chapter 1, we need to choose a large prime p in order to obtain a given security level. In a practical point of view, the cost of the pairing is affected when p increases.

Remark 4.10. we will see in Chapter 7 that supersingular curves become interesting for applications that mix isogeny-based cryptography together with pairing-based cryptography.

4.2.2 Generation of pairing-friendly elliptic curves

In the context of pairing-based cryptography, we leave supersingular curves and focus on ordinary curves. The only known methods for generating pairing-friendly elliptic curves are based on the CM method, described in Section 2.5. Instead of generating parameters a, b of a curve $E_{a,b}$, we look at parameters p, r and t requiring the properties of a pairing-friendly elliptic curve. Then, using the Hilbert class polynomial, we are able to generate an elliptic curve E defined over $\mathbb{F}_p$, for which r divides $p + 1 - t = E(\mathbb{F}_p)$. Hence, we are able to generate only small discriminant pairing-friendly ordinary elliptic curves. Parameters of the curves we expect to generate need to satisfy the following properties.

- The integers r and p are (large) primes.
  We look for an elliptic curve E defined over $\mathbb{F}_p$ and for pairings defined with groups of prime order r.

- r divides $p + 1 - t$.
  We want a subgroup of order r in $E(\mathbb{F}_p)$.

- The square-free part $-D$ of $t^2 - 4p$ is smaller in absolute value than 10^{15}.
  The method for generating the curve coefficients is based on the Hilbert class polynomial, which can be efficiently computed modulo p for a discriminant $D \leq 10^{15}$ [Sut10].

- The trace t is non-zero.
  Supersingular curves defined over $\mathbb{F}_p$ have trace $t = 0$ (see Section 3.1.2).

- The smallest integer $k \geq 1$ such that r divides $p^k - 1$ is *relatively small.*
  We look for pairing-friendly curves, i.e. curves with a small embedding degree.

- p^k is large enough to resist discrete logarithm computations.

A complete survey of the different methods for generating pairing-friendly curves is given in [FST10].

The two first approaches were proposed by Dupont, Enge and Morain in [DEM05], and Cocks and Pinch in [CP01]. Dupont et al. gave a method where t and r are simultaneously computed using resultants, whereas Cocks and Pinch proposed a more flexible algorithm. Both of these methods produce elliptic curves with a large trace $t \approx r$, and hence do not fit with an efficient ate pairing. Recall that the Miller loop T in the ate pairing is a k-th root of unity, which is closely related to $t - 1$: $(t-1)^k = q^k = 1 \bmod r$. We detail in Algorithm 4.2 the Cocks-Pinch method, usable with arbitrary embedding degree. Even if it produces an expensive ate pairing, we will see that the prime p generated with this algorithm is not *special* in the sense of Definition 1.3. In Chapter 5, we will modify this algorithm in order to obtain elliptic curves resistant to S(T)NFS variants, together with an efficient (optimal) ate pairing.

Algorithm 4.2: CocksPinch($k, -D$)

Input. k an embedding degree,
$-D$ a discriminant.
Output. p, r, t and y parameters of a pairing-friendly elliptic curve.

while True **do**
  Let r be a random prime such that k divides $r-1$ and $(-D/r)=1$.
  Let T be a random primitive k-th root of unity in $(\mathbb{Z}/r\mathbb{Z})^*$. Let $t' = T+1$.
  Let $y' = (t'-2)/\sqrt{-D} \bmod r$.
  Let $t \in \mathbb{Z}$ (resp. $y \in \mathbb{Z}$) be congruent to $t' \bmod r$ (resp. $y' \bmod r$).
  $p = (t^2 + Dy^2)/4$.
  if p is prime **then**
    return p, r, t, y
  end if
end while

Then, polynomial methods were also proposed by Barreto, Lynn, and Scott [BLS03], and independently by Brezing and Weng [BW05]. These methods parameterize the characteristic p, the trace t, and the curve subgroup order r by polynomials. We need to define polynomials *representing* primes.

Definition 4.11 (from [FST10, page 11])**.** *Let $f(x)$ be a polynomial with rational coefficients. f represents primes if $f(x)$ is non-constant and irreducible, has positive leading coefficient, and satisfies $f(x) \in \mathbb{Z}$ for some $x \in \mathbb{Z}$, and also* $\gcd(\{f(x) \text{ such that } x, f(x) \in \mathbb{Z}\}) = 1$.

The most general construction is stated by Brezing and Weng.

Theorem 4.12 (from [FST10])**.** *Fix a positive integer k and a positive square-free integer D. Execute the following steps:*

1. *Find an irreducible polynomial $r(x) \in \mathbb{Z}[x]$ with positive leading coefficient such that $K = \mathbb{Q}[x]/(r(x))$ is a number field containing $\sqrt{-D}$ and the k-th cyclotomic field.*
2. *Choose a primitive k-th root of unity $\zeta_k \in K$.*
3. *Let $t(x) \in \mathbb{Q}[x]$ be a polynomial mapping to $\zeta_k + 1$ in K.*
4. *Let $y(x) \in \mathbb{Q}[x]$ be a polynomial mapping to $(\zeta_k - 2)/\sqrt{-D}$ in K.*
5. *Let $p(x) \in \mathbb{Q}[x]$ be given by $(t(x)^2 + Dy(x)^2)/4$.*

Suppose $p(x)$ represents primes and $y(x_0) \in \mathbb{Z}$ for some $x_0 \in \mathbb{Z}$. Then, the triple $(t(x), r(x), y(x))$ parameterizes a family of elliptic curves with embedding degree k and discriminant $-D$.

Barreto, Lynn and Scott [BLS03] gave a construction where $r(x)$ is a cyclotomic polynomial Φ_k, and Brezing and Weng extended it with $r(x) = \Phi_\ell(x)$ where ℓ is a multiple of the embedding degree k. Several families were obtained with this method, and are surveyed in [FST10]. In some cases, non-cyclotomic polynomials lead to more effective constructions. One method to obtain such extensions is to evaluate $\Phi_\ell(x)$ at some polynomial $u(x)$. When $\Phi_\ell(u(x))$ factors, we may gain some advantage.

As we target the 128-bit security level, the study of the discrete logarithm problem over finite fields of Chapter 1 gives a lower bound on the size of the target finite field: $\log_2(p^k) > 3000$. From Chapter 2, we already know that $\log_2(p) > 256$, and choosing a small embedding degree $k \leq 3$ leads to a large prime p and hence slow pairing computations. We present here three families of curves of embedding degree $k = 12$ and 16, constructed from the polynomial method described above. We will see in Section 4.3 how efficient is the ate pairing on these curves.

The Barreto-Naehrig family. Barreto and Naehrig provided in [BN06] a family of pairing-friendly elliptic curves of embedding degree $k = 12$ and discriminant $D = 3$. The construction is based on Theorem 4.12 where $r(x)$ comes from a factor of $\Phi_{12}(6x^2)$, rescaled in order to have integer coefficients. Using this construction, r, t and p are parameterized by

$$\begin{aligned} r(x) &= 36x^4 + 36x^3 + 18x^2 + 6x + 1, \\ t(x) &= 6x^2 + 1, \\ p(x) &= 36x^4 + 36x^3 + 24x^2 + 6x + 1. \end{aligned}$$

BN curves have ρ-value $\rho = 1$ in the sense that $p(x) + 1 - t(x) = r(x)$. The quadratic twisted curve order is parameterized by $p(x)+1+t(x)$ which is also an irreducible polynomial. An optimal ate pairing is derived in [Ver10, page 10]. Setting $T = \lambda - \lambda^2 + \lambda^3 + 6x + 2 = r \cdot (6x^2 - 6x + 2)$ (where λ is a primitive 12-th root of unity) in Theorem 4.9, the ate pairing can be computed as $a_{r,[6x+2,1,-1,1]}(Q, P)$.

Before the last NFS improvements [BGK15, FGHT17, JP14, KB16], the parameters of these curves fit very well for a 128-bit security level: setting $x = -2^{62} - 2^{55} - 1$, the primes p and r were 256-bit integers (which is the minimal size to reach a 128-bit security in an elliptic curve), and the finite field $\mathbb{F}_{p^{12}}$ was large enough to reach the 128-bit security level. Unfortunately, the BN parameters fit also very well with the recent variants of NFS: the prime is parameterized by the polynomial $p(x)$ which has small coefficients and degree 4, and the field has several subfields. To reach the 128-bit security level, we need to study the NFS variant applied to the field $\mathbb{F}_{p(x)^{12}}$, for a given x. We estimated in Section 1.1.4 that the finite field $\mathbb{F}_{p^{12}}$ needs to be 5500-bit long in order to reach the 128-bit security level. Setting a seed $x = 2^{114} + 2^{101} - 2^{14} - 1$, [BD19] obtain a curve which satisfies $\log_2(p) = \log_2(r) = 462$. Hence, the target finite field is 5535-bit long, large enough to be STNFS-resistant. The quadratic twisted curve is also of prime order so that the curve is twist- and subgroup-secure.

The BLS family. We introduce another family which is called the Barreto-Lynn-Scott (BLS) family [BLS03]. These curves have embedding degree 3^i and $2^i \cdot 3$ ($i \geq 0$), and are very popular for embedding degree $k = 12$. In this case, BN and BLS12 curves are very similar for several aspects. First, r, t and p are also parameterized with polynomials:

$$\begin{aligned} r(x) &= x^4 - x^2 + 1, \\ t(x) &= x + 1, \\ p(x) &= (x^6 - 2x^5 + 2x^3 + x + 1)/3. \end{aligned}$$

As $p(x) + 1 - t(x) = r(x) \cdot (x-1)^2/3$, BLS12 curves have a ρ-value $\rho > 1$, but in practice, we will see that this cofactor is relatively small for a 128-bit security level. Both of BN and BLS12 curves have discriminant $D = 3$ and hence sextic twists. The pairing computation is slightly simpler in the case of BLS curves: choosing $T = t - 1 = x$, the ate pairing is optimal in the sense that

$\log_2(x) \leq \log_2(r(x))/\varphi(12)$. We will investigate the cost of this pairing in Section 4.3. The target finite field is $\mathbb{F}_{p(x)^{12}}$ as in the case of BN curves. In consequence, the study of the NFS variants on this field is closely related to the story of the BN curves. For a 128-bit target security, one needs to choose x such that $\mathbb{F}_{p(x)^{12}}$ is 5500-bit long. For instance, Barbulescu and Duquesne have chosen $x = -2^{77} + 2^{50} + 2^{33}$ [BD19], we obtain an elliptic curve defined over $\mathbb{F}_p$ with a prime $p = p(x)$ of 461 bits, and a subgroup of order $r(x)$ with $\log_2(r(x)) = 308$. The target finite field $\mathbb{F}_{p(x)^{12}}$ is 5525-bit long, which resists the recent variants of NFS.

The KSS family. The last family we present in this section is the Kachisa-Schaefer-Scott (KSS) family of curves [KSS08]. This curve construction is available for embedding degree $k \in \{8, 16, 18, 32, 36, 40\}$, but we present here only the instantiation for $k = 16$. The parameterization also comes from the Brezing-Weng method:

$$\begin{aligned} r(x) &= (x^8 + 48x^4 + 625)/61250, \\ t(x) &= (2x^5 + 41x + 35)/35, \\ p(x) &= (x^{10} + 2x^9 + 5x^8 + 48x^6 + 152x^5 + 240x^4 + 625x^2 + 2398x + 3125)/980. \end{aligned}$$

When $x \equiv 25$ or $45 \bmod 70$, these polynomials have integer coefficients. When also $p(x)$ is prime, then a pairing-friendly elliptic curve of discriminant $D = 4$ defined over $\mathbb{F}_{p(x)}$ can be derived. As for BN and BLS12 curves, the NFS variants apply: $\mathbb{F}_{p(x)^{16}}$ has subfields and the prime $p(x)$ is special so that the STNFS algorithm becomes efficient. A STNFS-secure instantiation is given in [BD19] for $x = -2^{35} - 2^{32} - 2^{18} + 2^8 + 1$. This way, the prime p satisfies $\log_2(p) = 339$ and $\mathbb{F}_{p^{16}}$ is 5424-bit long. The optimal ate pairing on KSS16 curves is given by

$$\Big(\big(f_{x,Q}(P) \cdot \ell_{[x]Q,[p]Q}(P) \big)^{p^3} \cdot \ell_{Q,Q}(P) \Big)^{(p^{16}-1)/r}.$$

4.3 Pairing cost in the general case

In the previous section, we have introduced three families of elliptic curves for which an optimal ate pairing is available. We now count the number of operations over $\mathbb{F}_p$ to compute it. As in Chapter 1, we denote by $\mathbf{m}_k$, $\mathbf{s}_k$, $\mathbf{i}_k$ and $\mathbf{f}_k$ the costs of multiplication, squaring, inversion, and p-th power Frobenius in $\mathbb{F}_{p^k}$, and by $\mathbf{m}$ the cost of a multiplication in $\mathbb{F}_p$. We neglect additions and multiplications by small constants. We recall the notations for the Hamming weight in base 2 and in non-adjacent form (NAF) introduced in Chapter 1: $\mathrm{HW}(x)$ (resp. $\mathrm{HW}_{\text{2-NAF}}(x)$) is the number of non-zero coefficients in the representation of x in base 2 (resp. in the 2-NAF representation of x).

4.3.1 Curve subgroup choices

In all the pairings introduced in Section 4.1, the points P and Q are chosen in $E[r]$. By definition of the embedding degree, $E[r]$ is fully rational over $\mathbb{F}_{p^k}$. Hence, P and Q have coordinates defined over $\mathbb{F}_{p^k}$. An efficient optimization is to choose particular subgroups of $E[r]$ in order to get points with *sparse* coordinates. Thus, Algorithm 4.1 computes some multiplications over $\mathbb{F}_{p^k}$ faster when only few coordinates are non-zero.

By construction, pairing-friendly curves defined over $\mathbb{F}_p$ have a subgroup of order r, called $\mathbb{G}_1 = E[r] \cap \ker(\pi - [1])$. One can choose[2] $P \in \mathbb{G}_1$ and get a point with very sparse coordinates

[2]It is actually necessary for the ate pairing, defined by construction for $(Q, P) \in \mathbb{G}_2 \times \mathbb{G}_1$.

(embedding x_P and y_P in $\mathbb{F}_{p^k}$, $k-1$ of their coefficients are zeros). The second point Q can also be sparse when the extension field $\mathbb{F}_{p^k}$ is well-chosen: suppose that the embedding degree k is divisible by $d \in \{2,3,4,6\}$, and that $\mathbb{F}_{p^k} = \mathbb{F}_{p^{k/d}}(\alpha)$ is defined using a binomial polynomial $X^d - w$ irreducible over $\mathbb{F}_{p^{k/d}}$ (hence $\alpha^d = w$). Then, if E has twists of degree d, then one twisted curve E' can be defined with

$$\begin{array}{rccc} \tau_d : & E_{a,b} & \longrightarrow & E' = E_{a\alpha^4, b\alpha^6} \\ & (x,y) & \longmapsto & (x\alpha^2, y\alpha^3). \end{array}$$

From the considerations of Section 2.7, $\mathbb{G}_2 = E[r] \cap \ker(\pi - [p]) \simeq \tau_d^{-1}(\mathbb{G}_2')$ where $\mathbb{G}_2' \subseteq E'(\mathbb{F}_{p^{k/d}})$. Suppose $Q' = (x', y') \in \mathbb{G}_2'$ (so $x', y' \in \mathbb{F}_{p^{k/d}}$). Then, $\tau_d^{-1}(Q') = (x'/\alpha^2, y'/\alpha^3) = (x'/w \cdot \alpha^{d-2}, y'/w \cdot \alpha^{d-3})$ has coordinates with only one non-zero coefficient in $\mathbb{F}_{p^{k/d}}$, instead of the d expected.

Example 4.13 (Compression of $\mathbb{G}_2$). Let E be an elliptic curve of embedding degree $k = 12$ and discriminant $D = 3$. In particular, E has sextic twists and its Weierstrass equation is $y^2 = x^3 + b$ for a given $b \in \mathbb{F}_p$. Choosing an irreducible polynomial $X^6 - z \in \mathbb{F}_{p^2}[X]$, the extension field $\mathbb{F}_{p^{12}}$ can be defined as $\mathbb{F}_{p^{12}} = \mathbb{F}_{p^2}(\sqrt[6]{z})$. A twisting isomorphism is given by $\tau_6 : (x,y) \in E \mapsto (\sqrt[3]{z}x, \sqrt{z}y) \in E'$ where E' is a sextic twisted curve to E. Then, choosing $Q \in \mathbb{G}_2$, we obtain a compression of factor 6 of the coordinates of Q: there exists $Q' = (x', y') \in \mathbb{G}_2'$ such that $Q = \tau_6^{-1}(Q') = (x'/z \cdot \sqrt[6]{z}^4, y'/z \cdot \sqrt[6]{z}^3)$. Writing the coordinates of Q in the basis of $\mathbb{F}_{p^{12}}$ over $\mathbb{F}_p$, only two elements among the twelve are non-zero.
A similar trick can be applied for an elliptic curve of embedding degree $k = 16$ and discriminant $D = 4$. Indeed, such a curve has quartic twists and in particular, defining $\mathbb{F}_{p^{16}} = \mathbb{F}_{p^4}(\sqrt[4]{w})$, coordinates of points of $\mathbb{G}_2$ (in the basis of $\mathbb{F}_{p^{16}}$) have only four non-zero elements and twelve zeros.

Reduced pairings with only one Miller function. As we have seen in Section 4.1.3, the Weil pairing is computed using Algorithm 4.1 four times: $f_{r,P}$ at $Q+S$ and at S, and $f_{r,Q}$ at $P+S'$ and at S'. In the case of *reduced* pairing, we also compute an extra final exponentiation to the power $(p^k-1)/r$ which has a non-negligible cost. Nevertheless, this structure of r-th root of unity lets us compute only one Miller evaluation instead of two. In the reduced Tate pairing, [BKLS02, page 7] remarks that $f_{r,P}(D_Q)^{(p^k-1)/r}$ can be computed as follows: instead of sampling a point S until the divisor $D_Q = (Q+S) - (S)$ has support disjoint with $(P) - (0_E)$, one can keep $(Q) - (0_E)$ and replace $f_{r,P}$ by another function $g_{r,P}$ of divisor whose support is disjoint from $\{Q, 0_E\}$. One can choose $R \notin \{0_E, -P\}$ and define $g_{r,P}$ to be a function of divisor $\mathrm{div}(g_{r,P}) = r(P+R) - r(R) \sim \mathrm{div}(f_{r,P})$. Then, the reduced Tate pairing is computed as $g_{r,P}((Q) - (0_E))^{(p^k-1)/r} = (g_{r,P}(Q)/g_{r,P}(0_E))^{(p^k-1)/r}$. Among these two Miller functions evaluations, $g_{r,P}(0_E) \in \mathbb{F}_p^*$ when $P \in E(\mathbb{F}_p)$ (which is the case here). Finally, $p-1$ divides $(p^k-1)/r$ and so this factor $g_{r,P}(0_E)$ vanishes after the final exponentiation: $g_{r,P}(0_E)^{(p^k-1)/r} = 1$. To conclude, one computes the reduced Tate pairing using the formula $t_r(P,Q) = f_{r,P}(Q)^{(p^k-1)/r}$. This trick also applies to the reduced ate and its optimal versions. From now on, we compute the reduced Tate and ate pairings as follows:

$$t_r(P,Q) = f_{r,P}(Q)^{(p^k-1)/r} \qquad A_{r,T}(Q,P) = f_{r,Q}(P)^{(p^k-1)/r}$$

As we have seen in Section 4.2.2, an optimal ate pairing can be derived using Theorem 4.9. In practice, the cost of this pairing (at least on BN, BLS12 and KSS16 curves) is dominated by a Miller function evaluation and a final exponentiation. We treat the cost of the pairing in the two following sections.

4.3.2 Miller step

In this section, we present a second version of Algorithm 4.1. The main difference between Algorithms 4.1 and 4.3 is the number of finite field inversions. In Algorithm 4.3, numerators and denominators are split into two variables in order to compute a single inversion at the end of the algorithm. Moreover, the integer T can be represented in non-adjacent form (NAF, see Section 2.8.3) in order to compute as little as possible Addition steps. Note that the case $b = -1$ in Algorithm 4.3 requires a specific treatment [EM14] because Equation (4.1) is slightly different with $j = 1$ and $j = -1$:

$$f_{i+1,Q} = f_{i,Q} \underbrace{f_{1,Q}}_{=1} \frac{\ell_{[i]Q,Q}}{v_{[i+1]Q}} \qquad f_{i-1,Q} = f_{i,Q} \underbrace{f_{-1,Q}}_{=1/v_Q} \frac{\ell_{[i]Q,-Q}}{v_{[i-1]Q}}.$$

Algorithm 4.3: MillerLoop(T, P, Q)

Input. T an integer,
$P \in E(\mathbb{F}_p)$,
$Q \in E(\mathbb{F}_{p^k})$.
Output. $f_{T,Q}(P)$.

$(\mu_{n,0}, \mu_{d,0}) \leftarrow v_Q(P)$ ▷in the NAF case;
$(m_n, m_d) \leftarrow (1, 1)$; $S \leftarrow Q$;
for b from the second most significant bit of $|T|$ to the least **do**
  $(\lambda_n, \lambda_d) \leftarrow \ell_{S,S}(P)$; $S \leftarrow [2]S$ ▷DoubleLine;
  $(\mu_n, \mu_d) \leftarrow v_S(P)$ ▷VerticalLine;
  $(m_n, m_d) \leftarrow (m_n^2 \lambda_n \mu_d, m_d^2 \lambda_d \mu_n)$ ▷Update1;
  if $b = \pm 1$ **then**
    $(\lambda_n, \lambda_d) \leftarrow \ell_{S,bQ}(P)$; $S \leftarrow S + bQ$ ▷AddLine;
    $(\mu_n, \mu_d) \leftarrow v_S(P)$ ▷VerticalLine;
    $(m_n, m_d) \leftarrow (m_n \lambda_n \mu_d, m_d \lambda_d \mu_n)$ ▷Update2;
    if $b = -1$ **then**
      $(m_n, m_d) \leftarrow (m_n \mu_{d,0}, m_d \mu_{n,0})$ ▷Update3;
    end if
  end if
end for
if $T < 0$ **then**
  $(m_n, m_d) \leftarrow (m_d, m_n)$
end if
return m_n/m_d

Before going into the details, one can already see that the cost of Algorithm 4.3 is given by the following formula, where the notation $\mathbf{c}_X$ denotes the cost of step X, or algorithm X, and $\mathrm{Nb}_{-1,T}$ denotes the number of -1 in the NAF representation of T.

$$\begin{aligned}\mathbf{c}_{\text{MillerLoop}} =& (\log_2(T) - 1)\,(\mathbf{c}_{\text{DoubleLine}} + \mathbf{c}_{\text{VerticalLine}}) \\ &+ (\log_2(T) - 2)\mathbf{c}_{\text{Update1}} \\ &+ (\mathrm{HW}_{\text{2-NAF}}(T) - 1)(\mathbf{c}_{\text{AddLine}} + \mathbf{c}_{\text{VerticalLine}} + \mathbf{c}_{\text{Update2}}) \\ &+ \mathrm{Nb}_{-1,T}(\mathbf{c}_{\text{Update3}}) + \mathbf{i}_k. \end{aligned} \tag{4.3}$$

We now focus on the cost of each step $\mathbf{c}_X$ in Equation (4.3). Before going into the details of the line computations and the cost of updates, we present a nice trick that will simplify a lot the line computations.

Useless factors in subfields. We have seen that only one single Miller function is computed during a reduced pairing. This comes from the fact that $g_{r,P}(0_E) \in \mathbb{F}_p^*$ vanishes after the final exponentiation. This trick is be generalized in [BKLS02] to all factors in proper subfields of $\mathbb{F}_{p^k}^*$. Suppose that $x \in \mathbb{F}_{p^l}$ with $l < k$ a divisor of k. Then, $p^l - 1$ divides $p^k - 1$ and more precisely, $p^l - 1$ divides the factor $(p^k - 1)/r$. Thus, $x^{(p^k-1)/r} = 1$. In consequence, all the factors in $\mathbb{F}_{p^l}^*$ do not need to be computed.

This trick, together with the particular choice of subgroups of $E[r]$ and the existence of twists defined over a subfield of $\mathbb{F}_{p^k}$, simplifies a lot the computations of the lines.

Vertical lines. When E has *even* embedding degree $k = 2k'$, no vertical line need to be computed. Recall that a vertical line at a point $Q \in \mathbb{G}_2$, evaluated at a point $P \in \mathbb{G}_1$, is written $v_Q(P)$ and is simply $x_P - x_Q$. Defining E over $\mathbb{F}_{p^{k'}}$ and $\mathbb{F}_{p^k} = \mathbb{F}_{p^{k'}}(\sqrt{w})$ for $w \in \mathbb{F}_{p^{k'}}$, the quadratic twist of E over $\mathbb{F}_{p^{k'}}$ can be defined using $\sqrt{w}$. Finally, $Q = (x'_Q/w, y'_Q/\sqrt{w}^3)$ for $x'_Q, y'_Q \in \mathbb{F}_{p^{k'}}$. Hence, $v_Q(P) = x_P - x'_Q/w$ is a factor in the subfield $\mathbb{F}_{p^{k/2}}$. Finally, when the embedding degree k is even, the vertical line computations always vanish after the final exponentiation and do not need to be taken into account. Moreover, it simplifies a lot Algorithm 4.3 because there is no denominator. In particular, Update3 does not need to be computed as it corresponds to vertical lines. All the curves considered in this chapter have this property so that we consider here $\mathbf{c}_{\text{Update3}} = 0$. However, in Chapter 5, we will consider curves where $\mathbf{c}_{\text{Update3}} > 0$.

Addition and doubling lines. The subfield factors in doubling and addition steps also vanish after the final exponentiation. Hence, these steps can be rewritten more efficiently. In particular, the line computations have been well studied in the case of $j = 0$ and 1728 curves. In this case, twists of degree $d \in \{4, 6\}$ lead to a point $Q = (x'_Q/w \cdot \alpha^{d-2}, y'_Q/w \cdot \alpha^{d-3})$ with $x'_Q, y'_Q \in \mathbb{F}_{p^{k/d}}$. Hence, the computations involve multiplications of sparse elements of $\mathbb{F}_{p^k}$, which correspond to multiplications in $\mathbb{F}_{p^{k/d}}$. In [AKL+11, page 12], the cost of DoubleLine and AddLine are detailed for curves of embedding degree k divisible by 6 and $D = 3$:

$$\mathbf{c}_{\text{DoubleLine}} = 3\mathbf{m}_{k/6} + 6\mathbf{s}_{k/6} + (k/3)\mathbf{m}$$
$$\mathbf{c}_{\text{AddLine}} = 11\mathbf{m}_{k/6} + 2\mathbf{s}_{k/6} + (k/3)\mathbf{m}.$$

The output of these functions is an element of $\mathbb{F}_{p^k}$ which has only three non-zero coefficients in $\mathbb{F}_{p^{k/6}}$. Similarly, [CLN10, pages 9,10] details the cost in the case of $k = 0 \bmod 4$ and $D = 4$:

$$\mathbf{c}_{\text{DoubleLine}} = 2\mathbf{m}_{k/4} + 8\mathbf{s}_{k/4} + (k/2)\mathbf{m}$$
$$\mathbf{c}_{\text{AddLine}} = 9\mathbf{m}_{k/4} + 5\mathbf{s}_{k/4} + (k/2)\mathbf{m}.$$

In this case, the output of these functions is an element of $\mathbb{F}_{p^k}$ with two non-zero coefficients in $\mathbb{F}_{p^{k/4}}$.

Updates. In the general case, Update1 (resp. Update2) costs $4\mathbf{m}_k + 2\mathbf{s}_k$ (resp. $4\mathbf{m}_4$). Nevertheless, from the considerations of vertical lines, no denominator is computed when the

embedding degree k is even. Moreover, when the curve has twists defined over $\mathbb{F}_{p^{k/d}}$, the outputs of the DoubleLine and AddLine steps are sparse element of $\mathbb{F}_{p^k}$. When $d = 6$, multiplying a *dense* element of $\mathbb{F}_{p^k}$ with a sparse one (3 non-zero coefficients in $\mathbb{F}_{p^{k/6}}$) costs $13\mathbf{m}_{k/6}$ as stated in Section 1.2.1. Similarly, when $d = 4$, multiplying an element of $\mathbb{F}_{p^k}$ with an element with only two non-zero coefficients in $\mathbb{F}_{p^{k/4}}$ costs $8\mathbf{m}_{k/4}$ instead of $9\mathbf{m}_{k/4}$ in the general case. Finally, we obtain that for these curves,

$$\mathbf{c}_{\text{UPDATE1}} = \begin{cases} 13\mathbf{m}_{k/d} + \mathbf{s}_k & \text{if } d = 6 \\ 8\mathbf{m}_{k/d} + \mathbf{s}_k & \text{if } d = 4 \end{cases} \qquad \mathbf{c}_{\text{UPDATE2}} = \begin{cases} 13\mathbf{m}_{k/d} & \text{if } d = 6 \\ 8\mathbf{m}_{k/d} & \text{if } d = 4. \end{cases}$$

4.3.3 Final exponentiation

Recall that in the cases of reduced (Tate, ate and optimal ate) pairings, we need to raise to the power $(p^k - 1)/r$ the output of the Miller function. This exponent splits into two parts using the formula

$$\frac{p^k - 1}{r} = \frac{p^k - 1}{\Phi_k(p)} \cdot \frac{\Phi_k(p)}{r}.$$

First part. The first part of the exponentiation uses few Frobenius powers and inversions and its cost depends on the value of $\Phi_k(p)$. Its computation is very efficient because of Frobenius powers (see Section 1.2.2). In particular, for $x \in \mathbb{F}_{p^k}$ with k even, $x^{p^{k/2}}$ is almost free: it is simply the conjugate of x seen in a quadratic extension of $\mathbb{F}_{p^{k/2}}$.

Second part. The second part of the exponentiation is more expensive than the first one. Notice that the output of the first exponentiation is an element of the cyclotomic subgroup: $\left(x^{(p^k-1)/\Phi_k(p)}\right)^{\Phi_k(p)} = x^{p^k-1} = 1$. Hence, the cyclotomic squarings (see Section 1.2.4) can be used in the second part of the exponentiation. The cost estimation of raising to this second exponentiation is specific to each curve. The key ingredient is the base-p representation of the exponent, since Frobenius powers p^i are computed efficiently. The idea is to choose a polynomial which is a multiple of $\Phi_k(p)/r$ so that the representation in base p has particular coefficients. Later, we will treat the case of BN, BLS12 and KSS16 curves for which the primes p and r are parameterized with polynomials. Finally, in these three cases, the cost is dominated by exponentiations by x, which can be computed efficiently with the formula of Section 1.2.4. We will detail the case of our three families in the next section.

4.4 Pairing cost in the case of three families of curves

In this section, we investigate the explicit cost of the pairing for the three families of curves which have been presented in Section 4.2.2: BN, BLS12 and KSS16 elliptic curves. Each of them splits into a Miller step, possible extra lines computations and a final exponentiation. Several estimates below differ marginally from [BD19], which uses a different estimated cost for inversions and multiplications of sparse elements. We provide a script that automatically computes the cost of the pairing on each curve. The code repository is publicly accessible at:

`https://gitlab.inria.fr/smasson/cocks-pinch-variant`.

4.4.1 Barreto-Naehrig curves.

On BN curves, an optimal pairing is obtained choosing a 12-th root of unity λ and setting $T = \lambda - \lambda^2 + \lambda^3 + 6x + 2 = r \cdot (6x^2 - 6x + 2)$ in Theorem 4.9. Finally, $a_{r,[6x+2,1,-1,1]}(Q,P)$ is computed as follows:

$$\left(f_{6x+2,Q}(P) \cdot \ell_{[p^3-p^2+p]Q,[6x+2]Q}(P) \cdot \ell_{[p^3-p^2]Q,[p]Q}(P) \cdot \ell_{[p^3]Q,[-p^2]Q}(P)\right)^{\frac{p^k-1}{r}}.$$

Miller loop. The Miller function computation is $f_{6x+2,Q}(P)$. BN curves have embedding degree $k = 12$ and $D = 3$ so the optimizations of Section 4.3.2 are available. In particular, we benefit from the sextic twists and the embedding degree is even, so that no denominator is computed. Updates also benefit of the sparsity of the elements (see Section 1.2.1). Finally, the steps of Algorithm 4.3 are efficiently computed using subfield multiplications:

$$\begin{aligned} \mathbf{c}_{\text{DoubleLine}} &= 3\mathbf{m}_2 + 6\mathbf{s}_2 + 4\mathbf{m}, & \mathbf{c}_{\text{AddLine}} &= 11\mathbf{m}_2 + 2\mathbf{s}_2 + 4\mathbf{m}, \\ \mathbf{c}_{\text{Update1}} &= 13\mathbf{m}_2 + \mathbf{s}_{12}, & \mathbf{c}_{\text{Update2}} &= 13\mathbf{m}_2. \end{aligned}$$

We obtain the final cost of the Miller step:

$$\begin{aligned} \mathbf{c}_{\text{MillerLoop}} =& (\log_2(6x+2)-1)(3\mathbf{m}_2 + 6\mathbf{s}_2 + 4\mathbf{m}) + (\log_2(6x+2)-2)(13\mathbf{m}_2 + \mathbf{s}_{12}) \\ &+ (\text{HW}_{\text{2-NAF}}(6x+2)-1)(11\mathbf{m}_2 + 2\mathbf{s}_2 + 4\mathbf{m} + 13\mathbf{m}_2). \end{aligned}$$

Extra lines. The optimal pairing $a_{r,[6x+2,1,-1,1]}$ requires few extra line computations. The points used in these line computations are actually easy to get. The first factor $\ell_{[p^3-p^2+p]Q,[6x+2]Q}(P)$ is a vertical line: $p^3 - p^2 + p = -(6x+2) \bmod r$. Hence, this factor can be omitted. The two other factors simplify using the trick presented in Proposition 2.5:

$$\begin{aligned} \ell_{[p^3-p^2]Q,[p]Q} &= \ell_{-[6x+2]Q-[p]Q,[p]Q} = \ell_{[p]Q,[6x+2]Q} \\ \ell_{[p^3]Q,-[p^2]Q} &= \ell_{-[p+6x+2]Q-[-p^2]Q,[-p^2]Q} = \ell_{[-p^2]Q,[p+6x+2]Q}. \end{aligned}$$

We now write $Q' = [6x+2]Q$, which is output from the Miller algorithm, and $Q_1 = [p]Q$ and $Q_2 = [p]Q_1$, computed using Frobenius exponentiations on the (sparse) coordinates of Q and Q_1 (4 times $2\mathbf{m}_2$). Using these notations, the two latter line computations are $\ell_{Q_1,Q'}(P)$ and $\ell_{-Q_2,Q_1+Q'}(P)$, costing one addition line of cost $\mathbf{c}_{\text{AddLine}}$ and one *light* addition line. We estimate that this last operation costs $4\mathbf{m}_2 + 4\mathbf{m}$ as the point does not need to be computed. Multiplying them together costs only $7\mathbf{m}_2$ using their sparsity. We finally update the Miller function output with one dense×sparse multiplication ($13\mathbf{m}_2$, see Secton 1.2.1) and obtain:

$$\mathbf{c}_{\text{ExtraLines}} = (4 \cdot 2\mathbf{m}_2) + (11\mathbf{m}_2 + 2\mathbf{s}_2 + 4\mathbf{m}) + (4\mathbf{m}_2 + 4\mathbf{m}) + (7\mathbf{m}_2) + (13\mathbf{m}_2)$$

The precise cost of these operations can probably be improved but it is negligible compared to the total cost of the pairing computation.

Final exponentiation. The final exponentiation splits into $(p^6-1)(p^2+1)(p^4-p^2+1)/r$. As claimed in Section 4.3.3, the first two factors are easy to compute with Frobenius and inversions: raising to the power p^6 is simply the conjugate of the element seen in a quadratic extension of $\mathbb{F}_{p^6}$, and raising to the power p^2 is computed in $5\mathbf{m}_2$ using the Frobenius computation in a sextic extension of $\mathbb{F}_{p^2}$. We obtain that $\mathbf{c}_{\text{FirstExp}} = (\mathbf{i}_{12} + \mathbf{m}_{12}) + (5\mathbf{m}_2 + \mathbf{m}_{12})$. The second

part of the final exponentiation is to raise to the power $\Phi_k(p)/r$. BN curves construction uses a parameterization for the primes p and r. More precisely,

$$p = p(x) = 36x^4 + 36x^3 + 24x^2 + 6x + 1 \qquad r = r(x) = 36x^4 + 36x^3 + 18x^2 + 6x + 1.$$

Then, we consider $d(x) = \Phi_{12}(p(x))/r(x)$ in base p

$$\Phi_{12}(p)/r = p^3 + (6x^2+1)p^2 + (-36x^3 - 18x^2 - 12x + 1)p + (-36x^3 - 30x^2 - 18x - 2).$$

The most efficient algorithm to compute this last exponentiation is explained in [FKR12, page 4]. Using the fact that a fixed power of a pairing is also a pairing, they choose an exponent so that the final exponentiation is the most efficient possible. More precisely, d' is a multiple of d that is not divisible by r. A lattice-based method lets us obtain that it suffices to raise to the power $d'(x) = 2x(6x^2 + 3x + 1)d(x) = \sum_{i=0}^{3} \lambda_i(x)p^i$ where

$$\lambda_0(x) = 1 + 6x + 12x^2 + 12x^3, \qquad \lambda_1(x) = 4x + 6x^2 + 12x^3,$$
$$\lambda_2(x) = 6x + 6x^2 + 12x^3, \qquad \lambda_3(x) = -1 + 4x + 6x^2 + 12x^3.$$

Using this decomposition in base p, [FKR12, page 6] obtain that $\mathbf{c}_{\text{SecondExp}} = 3\mathbf{c}_x + 3\mathbf{s}_{12}^{\text{cyclo}} + 10\mathbf{m}_{12} + 4\mathbf{f}_{12}$ where $\mathbf{c}_x$ is the cost of an exponentiation to the power x, which costs (according to [GS10, page 7])

$$\mathbf{c}_x = 4(\log_2(x) - 1)\mathbf{m}_2 + (6(\text{HW}(x) - 1) - 3)\mathbf{m}_2 + (\text{HW}(x) - 1)\mathbf{m}_{12} + 3(\text{HW}(x) - 1)\mathbf{s}_2 + \mathbf{i}_2.$$

Estimated cost for a 128-bit security level. For a 128-bit security level, Barbulescu and Duquesne recommend in [BD19] to set $x = 2^{114} + 2^{101} - 2^{14} - 1$. Replacing $\log_2(6x+2)$ and $\text{HW}_{\text{2-NAF}}(6x+2)$ by 117 and 7, we obtain

$$\begin{aligned}\mathbf{c}_{\text{MillerLoop}} &= 11620\mathbf{m}\\ \mathbf{c}_{\text{FinalExp}} &= 5364\mathbf{m}\\ \mathbf{c}_{\text{OptimalAte}} &= 16984\mathbf{m}.\end{aligned}$$

4.4.2 Barreto-Lynn-Scoot curves ($k = 12$).

On a BLS12 curve, the ate pairing is already optimal in the sense that $T = t - 1 = x$ has the minimal size. The computation of this optimal pairing is hence simply $f_{x,Q}(P)^{(p^k-1)/r}$, without any extra line computation.

Miller loop. These curves also benefit from sextic twists and even embedding degree. Hence, the cost of line computations and updates are the same as for BN curves. Finally, we obtain that

$$\begin{aligned}\mathbf{c}_{\text{MillerLoop}} =&(\log_2(x) - 1)(3\mathbf{m}_2 + 6\mathbf{s}_2 + 4\mathbf{m}) + (\log_2(x) - 2)(13\mathbf{m}_2 + \mathbf{s}_{12})\\ &+ (\text{HW}_{\text{2-NAF}}(x) - 1)(11\mathbf{m}_2 + 2\mathbf{s}_2 + 4\mathbf{m} + 13\mathbf{m}_2).\end{aligned}$$

Final exponentiation. As for BN curves, the exponent splits into two parts using the cyclotomic polynomial $\Phi_{12}(p)$. We obtain the same first exponentiation which costs $\mathbf{c}_{\text{FirstExp}} = (\mathbf{i}_{12} + \mathbf{m}_{12}) + (5\mathbf{m}_2 + \mathbf{m}_{12})$. The second part of the exponentiation is similar with the BN case. We

finally raise to the power $3\Phi_{12}(p)/r$ using the parameterization $p(x) = (x^6 - 2x^5 + 2x^3 + x + 1)/3$ and $r(x) = x^4 - x^2 + 1$. Written in base p, we obtain that $3\Phi_{12}(p)/r = \sum_{i=0}^{3} \lambda_i(x)p^i$ with

$$\lambda_0(x) = 3 - x + 2x^2 - 2x^4 + x^5, \qquad \lambda_1(x) = -1 + 2x - 2x^3 + x^4,$$
$$\lambda_2(x) = x - 2x^2 + x^3, \qquad \lambda_3(x) = 1 - 2x + x^2.$$

Finally, [AFK+13, page 10] obtains $\mathbf{c}_{\text{SecondExp}} = 5\mathbf{c}_x + 10\mathbf{m}_{12} + 3\mathbf{f}_{12} + 2\mathbf{s}_{12}^{\text{cyclo}}$, where $\mathbf{c}_x$ is the cost of an exponentiation to the power x, given in Section 4.4.1.

Estimated cost for a 128-bit security level. For a 128-bit security level, Barbulescu and Duquesne set in [BD19] $x = -2^{77} + 2^{50} + 2^{33}$. We obtain with our formulas

$$\mathbf{c}_{\text{MillerLoop}} = 7685\mathbf{m}$$
$$\mathbf{c}_{\text{FinalExp}} = 6288\mathbf{m}$$
$$\mathbf{c}_{\text{OptimalAte}} = 13973\mathbf{m}.$$

4.4.3 Kashisa-Schaefer-Scott curves ($k = 16$).

On a KSS16 curve, [ZL12] obtain that $a_{r,[x,1,0,0,0,-2,0,0]}$ is optimal, and raising to the power p^3, we obtain another optimal pairing defined by

$$\Big(\big(f_{x,Q}(P) \cdot \ell_{[x]Q,[p]Q}(P)\big)^{p^3} \cdot \ell_{Q,Q}(P)\Big)^{(p^{16}-1)/r}.$$

Miller loop. The Miller step is efficiently computed using the optimizations of Section 4.3:

$$\mathbf{c}_{\text{DoubleLine}} = 2\mathbf{m}_4 + 8\mathbf{s}_4 + 8\mathbf{m}, \qquad \mathbf{c}_{\text{AddLine}} = 9\mathbf{m}_4 + 5\mathbf{s}_4 + 8\mathbf{m},$$
$$\mathbf{c}_{\text{Update1}} = 8\mathbf{m}_4 + \mathbf{s}_{16}, \qquad \mathbf{c}_{\text{Update2}} = 8\mathbf{m}_4.$$

The Miller algorithm cost is thus

$$\begin{aligned}\mathbf{c}_{\text{MillerLoop}} =& (\log_2(x) - 1)(2\mathbf{m}_4 + 8\mathbf{s}_4 + 8\mathbf{m}) + (\log_2(x) - 2)(8\mathbf{m}_4 + \mathbf{s}_{16}) \\ &+ (\text{HW}_{\text{2-NAF}}(x) - 1)(9\mathbf{m}_4 + 5\mathbf{s}_4 + 8\mathbf{m} + 8\mathbf{m}_4).\end{aligned}$$

Extra lines. As in the case of BN curves, the extra line computations are simplified: denote $Q' = [x]Q$ and $Q_1 = [p]Q$. The first point is computed from the Miller function, and the second one using two Frobenius exponentiations on x_Q and y_Q, which are sparse because $Q \in \mathbb{G}_2$ ($2\mathbf{m}_4$). According to [ZL12, page 422], the line $\ell_{Q',Q_1}(P)$ is computed in $4\mathbf{m}_4 + 8\mathbf{m}$ and $\ell_{Q,Q}(P)$ in $\mathbf{m}_4 + \mathbf{s}_4 + 8\mathbf{m}$ using *light* line computations (the output points are not computed). The exponentiation to the power p^3 costs only one exponentiation in $\mathbb{F}_{p^{16}}$ and we finally obtain the cost of the extra line computations with two sparse multiplications ($2 \cdot 2\mathbf{m}_4$):

$$\mathbf{c}_{\text{ExtraLines}} = (2\mathbf{m}_4) + (4\mathbf{m}_4 + 8\mathbf{m}) + (\mathbf{m}_4 + \mathbf{s}_4 + 8\mathbf{m}) + \mathbf{f}_{16} + (2 \cdot 2\mathbf{m}4)$$

Again, we warn the reader that this extra lines computation can probably be improved but it would be negligible compared to the total cost of the pairing.

Final exponentiation. The first part of the exponentiation is just a conjugate, an inversion and a multiplication ($\mathbf{i}_{16} + \mathbf{m}_{16}$). The second exponentiation is detailed in [ZL12]. They raise to the power 857500 to get a *nice* decomposition of the exponent $857500\Phi_{16}(p)/r = \sum_{i=0}^{7} \lambda_i(x)p^i$. The λ_i are provided in [ZL12, page 423]. We estimate that this extra computation costs $34\mathbf{s}_{16}^{\text{cyclo}} + 32\mathbf{m}_{16} + 24\mathbf{m}_4 + 8\mathbf{f}_{16} + \mathbf{i}_{16} + 9(34\mathbf{s}_{16}^{\text{cyclo}} + 4\mathbf{m}_{16})$.

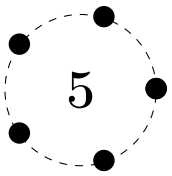
Estimated cost for a 128-bit security level. We reproduce the results from [CLN10] with the parameter $u = 2^{35} - 2^{32} - 2^{18} + 2^8 + 1$ from [BD19]. We obtain:

$$\begin{aligned} \mathbf{c}_{\text{MillerLoop}} &= 7573\mathbf{m} \\ \mathbf{c}_{\text{FinalExp}} &= 25521\mathbf{m} \\ \mathbf{c}_{\text{OptimalAte}} &= 33094\mathbf{m}. \end{aligned}$$

We conclude this chapter with Table 4.1 which compares the three families of curves at the 128-bit security level. All the counts we obtain are reproducible with the code provided at the begining of Section 4.4. All these cost estimations can slightly differ in practice, depending on the implementation.

Curve	**Miller loop**	**Final exponentiation**	**Optimal ate**
BN	12005**m**	5742**m**	17747**m**
BLS12	7685**m**	6288**m**	13973**m**
KSS16	7573**m**	25521**m**	33094**m**

Table 4.1: Cost of the optimal ate pairing for three families of curves at the 128-bit security level.



New pairing-friendly curves

The recent improvements of the NFS algorithms threaten the security of the discrete logarithm problem over particular finite fields. The three families of curves presented in Chapter 4 as well as all families of curves in [FST10] compute p as a polynomial evaluated at a chosen integer. This (often) enables the STNFS algorithm [JP14], so that the DL problem in $\mathbb{F}_{p^k}$ is easier than in other fields of same bit length. While composite extension degrees are appealing for fast pairing computation (see Section 4.3.2), they also offer additional parameterization choices for the TNFS algorithm [KB16]. This also makes DL computations in $\mathbb{F}_{p^k}$ more efficient. In order to resist the recent NFS variants, the sizes of the BN, BLS12 and KSS16 curves parameters need to increased to get a large enough finite field $\mathbb{F}_{p^k}$. We gave for these three curves in Section 4.4 a detailed estimation of the number of $\mathbb{F}_p$ multiplications to compute an optimal ate pairing for a 128-bit security level.

Cryptographers looked for curves immune to these attacks. Chatterjee, Menezes and Rodríguez-Henríquez proposed in [CMR17] to design elliptic curves of embedding degree $k = 1$ in order to avoid the TNFS attack. Thus, the target finite field is a prime field. Moreover, since a non-special prime p is chosen, the *Special* variant of NFS (SNFS) is not efficient and from the considerations of Chapter 1, we obtain a lower bound on the size of the field $\mathbb{F}_p$ ($\log_2(p) \geq 3072$) in order to reach the 128-bit security level. These security advantages are balanced by drawbacks with respect to the efficiency of pairings. On these curves, the ate pairing is not available because the trace is $t = 2$. Hence, the Tate pairing must be used. Its cost is given in [CMR17]: for a 256-bit r, the Miller loop costs $4626\mathbf{m} + \mathbf{i}$ and the final exponentiation costs $4100\mathbf{m}$. The total cost is finally $8726\mathbf{m} + \mathbf{i}$. Note that the cost of $\mathbf{m}$ is much larger than in the case of the curves of Chapter 4: the field $\mathbb{F}_p$ is up to six times larger. Enge and Milan proposed in [EM14] curves of prime embedding degree $k \geq 9$ for the larger security levels of 192 and 256 bits before the TNFS attack. Their choices of curves can be revisited for the security level of 128 bits, considering the TNFS attack.

In this chapter, we aim to generate new curves of embedding degree $k > 1$ that can resist the NFS variants by construction, and estimate the cost of a pairing in these cases. We wish to avoid special primes so that SNFS is not efficient. We present the results of a joint work [GMT20] with Aurore Guillevic and Emmanuel Thomé. More precisely, we introduce a method for generating pairing-friendly curves for which the prime is not parameterized with a polynomial with small

coefficients. We study the case of four embedding degrees: $k = 5$, 6, 7 and 8. Using a prime embedding degree ($k = 5$ and 7), the *Tower* variant of NFS is not efficient, but the optimizations described in Section 4.3 are not available either. In the case of embedding degree $k = 6$ and 8, the Tower NFS is efficient, but we expect to have a prime p which is not special so that the finite field $\mathbb{F}_{p^k}$ is smaller than in the case of BN, BLS12 and KSS16 curves.

The estimation of the pairing cost on these new curves differs slightly from the analysis done for the curve families in Chapter 4, even though curves of embedding degree 6 and 8 benefit from the considerations of Section 4.3. We present in Section 5.3 an estimation of an optimal ate pairing on four curves generated with the modified Cocks-Pinch method of Section 5.1.

We conclude this chapter with a comparison between our new curves (NFS-resistant by construction) and the curves introduced in Chapter 4 with a larger field $\mathbb{F}_{p^k}$.

Summary

5.1 A modified Cocks-Pinch algorithm

In Chapter 4, we presented families of elliptic curves which have the property of being pairing-friendly. These curves are constructed from polynomial methods, which lead to optimizations of the pairing computations. Nevertheless, the recent NFS variants also fit very well with the polynomial families (and in particular the BN, BLS12 and KSS16 curves presented in Chapter 4). Hence, in order to reach the 128-bit security level, we need to increase the size of the finite field $\mathbb{F}_{p^k}$.

In this section, we look for curves defined over $\mathbb{F}_p$ where the prime p is not special in the sense of Definition 1.3. Thus, the targeted curves are not threatened by the recent variants of NFS and we expect to obtain a smaller $\mathbb{F}_{p^k}$ size. Our construction is based on the Cocks-Pinch algorithm, already presented in Section 4.2.2.

5.1.1 Pros and cons of the Cocks-Pinch algorithm

Freely chosen embedding degree. We presented in Algorithm 4.2 the Cocks-Pinch method, that generates pairing-friendly elliptic curves for any freely chosen embedding degree. This is an interesting property because we expect to avoid the tower NFS variant, which is more efficient for composite embedding degrees. Some of the curve families we consider in this chapter have prime embedding degree. This property avoids the efficiency of the TNFS algorithm, but also disables optimizations of the pairing computation.

Non-special primes. In the Cocks-Pinch algorithm, the parameters are chosen using integers mod r, contrary to the polynomial methods that led to the three families of Section 4.2.2. In Algorithm 4.2, t' is a polynomial evaluated at the k-th root of unity T. However, y' is defined using the inverse of $\sqrt{-D}$ mod r.

- When the discriminant is large, $1/\sqrt{-D}$ has no structure and so no sparse polynomial can parameterize the prime $p = (t^2 + Dy^2)/4$.
- Choosing a particular discriminant (typically $D = 3$ or 4), one can obtain a parameterization of t and y. For instance, when $D = 4$ and $k = 4$, $\sqrt{-D} = 2T^2$ where T is a primitive fourth root of unity mod r. Then $y' = (T-1)/\sqrt{-D} = (T-1)T^2/2$ is a polynomial in T. Finally, using a trivial lift from $\mathbb{Z}/r\mathbb{Z}$ to $\mathbb{Z}$, we obtain a parameterization of p that could accelerate the discrete logarithm computations using the special variant of NFS.

Slow pairings. The Cocks-Pinch algorithm is not used in practice for one main reason: the pairing on Cocks-Pinch curves is not efficient. In the original algorithm, the ρ-value is by definition $\rho(E) \approx 2$ because $\log_2(p) \approx 2(\log_2(t) - 1)$. Moreover, the trace t of the curve is an integer mod r which satisfies $\log_2(t) \approx \log_2(r) \geq 256$ for our security level. As we have seen in several examples, the pairing efficiency is closely related to the trace t of the curve: in the ate pairing, the Miller function iterates on a k-th root of unity which is a power of $t - 1$. Using a large trace t, the number of DoubleLine steps increases and the efficiency is affected. In the next section, we modify the Cocks-Pinch algorithm in order to generate curves with a small trace t.

5.1.2 Tweak of the Cocks-Pinch algorithm

To avoid special primes, we revisit the Cocks-Pinch method, which constructs pairing-friendly curves with freely chosen embedding degree k and discriminant $-D$. The classical Cocks-Pinch algorithm first fixes the prime r and deduces a root of unity mod r to compute t and then p satisfying the conditions of pairing-friendly curves. Instead, we first choose T small, and then compute r such that T is a root of the k-th cyclotomic polynomial Φ_k mod r. Our variant is given in Algorithm 5.1. The trace $\bar{t} \in \mathbb{Z}/r\mathbb{Z}$ can be any of $\bar{t} = T^i + 1$ mod r where $\gcd(i, k) = 1$, and $\pm\bar{y} = (\bar{t} - 2)/\pm\sqrt{-D}$ mod r as in the original method. We then lift $\bar{t}, \pm\bar{y}$ to $t_0, y_0 \in \mathbb{Z}$. There is a trade-off between non-speciality of the prime p and optimization of the pairing efficiency (i.e. choosing sparse parameters T, h_t and h_y). Since $\sqrt{-D}$ is only determined up to sign, we fix the sign indetermination problem as follows.

Remark 5.1 (Normalisation of t_0 and y_0). When lifting $\bar{t}, \pm\bar{y}$ from $\mathbb{Z}/r\mathbb{Z}$ to $\mathbb{Z}$, we choose t_0 as the signed representative of $\bar{t}$ with smallest absolute value, and y_0 as the smallest positive representative of $\pm\bar{y}$.

The choice of the cofactors h_t and h_y in Algorithm 5.1 must abide by certain rules so that the Weil number π is an algebraic integer: if $-D \equiv 0 \mod 4$, then $t_0 + h_t$ must be even, and if $-D \equiv 1 \mod 4$, then $t_0 + y_0 + h_t + h_y$ must be even. For p to have the desired bit length, we notice that h_t and h_y must be chosen in an annulus-like region given by the equation $2^{\lambda_p+1} \leq (t_0 + h_t r)^2 + D(y_0 + h_y r)^2 < 2^{\lambda_p+2}$.

Algorithm 5.1: ModifiedCocksPinch$(k, -D, T_0, T_{\max}, \lambda_r, \lambda_p)$

Input. An embedding degree k, a discriminant $-D$, a range $\{T_0, \ldots, T_{\max}\}$, a bitlength λ_r and a bitlength λ_p.

Output. A pairing-friendly curve of embedding degree k and fundamental discriminant $-D$, where $\lceil \log_2(p) \rceil = \lambda_p$ and $\lceil \log_2(r) \rceil = \lambda_r$.

for $T \in \{T_0, \ldots, T_{max}\}$ **do**
  if $r = \Phi_k(T)$ *is not prime* **then continue**
  if $\lceil \log_2(r) \rceil \neq \lambda_r$ *or* $-D$ *is not a square mod* r **then continue**
  for i *in* $\{1, 2, \ldots, k-1\}$ *such that* $\gcd(i, k) = 1$ **do**
    $t_0 = T^i + 1 \mod r$; $y_0 = (t_0 - 2)/\sqrt{-D} \mod r$ ▷ see Section 1.2.2
    Let $\pi_0 = \frac{t_0 + y_0\sqrt{-D}}{2}$.
    Choose h_t and h_y such that $\pi = \pi_0 + \frac{h_t + h_y\sqrt{-D}}{2} r$ is an algebraic integer, and $\lceil \log_2(\pi\bar{\pi}) \rceil = \lambda_p$.
    $t = t_0 + h_t r$; $y = y_0 + h_y r$; $p = \pi\bar{\pi} = (t^2 + Dy^2)/4$
    if $p \equiv 1 \mod k$ **then** ▷ optimization; see Remark 1.4
      if p *is prime* **then return** $[p, r, T, t, y]$

5.1.3 Special form of the obtained prime

The *special* variants of NFS rely on a special form of p: the prime integer should have a polynomial form $p = p(x)$ where $p(x)$ is a polynomial. Roughly put, for the special variants to apply, $p(x)$ should have degree at least 3 and (most importantly) tiny coefficients. In Chapter 1, we have studied several examples of parameterizations where the special variant of NFS apply. In particular, the prime involved in the BN curves fits with the Special setting: the polynomial $p(x)$ has degree 4 and small coefficients. Parameters of BLS12 and KSS16 curves also fit well with SNFS (one can use $3p(x)$ and $980p(x)$ in order to obtain a polynomial with small coefficients). We will obtain families of curves where $p(x)$ has a degree larger than 2 but has large coefficients so that the special setting (STNFS) does not perform better than a generic setting (TNFS).

In this section, we discuss whether we hold to our promise that p, as issued by Algorithm 5.1, is not special. The prime p has the form

$$p = \frac{1}{4}\left((t_0 + h_t r)^2 + D\,(y_0 + h_y r)^2\right)$$

where t_0, y_0 are centered representatives of $T^i + 1 \bmod r$ and $(t_0 - 2)/\sqrt{-D} \bmod r$ resp., and both r and t_0 are low-degree polynomials in T.

If T, h_t, and h_y are chosen by Algorithm 5.1 as random integers of the desired bit length, and that D is arbitrary (then y_0 has no nice sparse polynomial expression in T), then the expression above is considered unlikely to yield any computational advantage to an attacker.

On the other hand, efficiency considerations of Section 4.3 may lead us to choose D specially, so as to allow extra automorphisms on the curve, for example choose $-D$ as the discriminant of the d-th cyclotomic field $\mathbb{Q}(\zeta_d)$ for some $d \mid k$. Then $\sqrt{-D}$ typically has a low-degree polynomial expression in T. Our case of interest is typically $D = 4$ and $D = 3$, when some twists may be defined over a subfield of $\mathbb{F}_{p^k}$. From Section 2.7, this is the case in particular when $D = 4$ and $4 \mid k$ or when $D = 3$ and $6 \mid k$.

- If $D = 4$ and $k = 4k'$, then by construction, T is a primitive k-th root of unity. Hence, $\zeta = T^{k'}$ is a fourth root of unity and so up to sign, $\sqrt{-4} = 2\zeta$.
- If $D = 3$ and $k = 6k'$, then $\zeta = T^{2k'}$ is a third root of unity, and so up to sign, $\sqrt{-3} = 2\zeta+1$.

The study of the pairing cost in Section 4.3 explains that the Miller loop and the final exponentiation are accelerated with sparse parameters. In particular, T, h_t, and h_y can be chosen with low Hamming weight, and in this context, the answer for the parameterization of p is less clear. In comparison to other pairing-based constructions however, we have here a *multivariate* expression for p (it depends on T, h_t, and h_y). First there exists no special-purpose NFS construction that adapts well to a bivariate polynomial. Secondly for fixed h_t and h_y, one can obtain a univariate polynomial in T. This setting will provide a notable advantage to NFS *only if h_t and h_y are tiny enough to produce a sparse polynomial* $p_{h_t,h_y}(T)$. As an illustration, here is the multivariate expression of $4p$ in the case $k = 8$, $D = 4$, and $i = 1$.

$$\begin{aligned} 4p = &(h_t^2 + 4h_y^2)T^8 + 4h_yT^7 - (4h_y - 1)T^6 + 2(h_t - 1)T^5 \\ &+(2h_t^2 + 8h_y^2 + 2h_t + 1)T^4 + 4h_yT^3 - (4h_y - 1)T^2 + 2(h_t + 1)T \\ &+(h_t^2 + 4h_y^2 + 2h_t + 1). \end{aligned}$$

If h_t and h_y are small, one can get a univariate expression for p and exploit the S(T)NFS variant. In [FK19, Remark 3, Table 3], one finds a family (denoted FK-8 in the following to stand for Fotiadis-Konstantinou) with $D = 4, h_t = 1, h_y = 0$ and $p(x) = (x^8 + x^6 + 5x^4 + x^2 + 4x + 4)/4$, $r(x) = x^4+1$, $t(x) = x+1+r(x) = x^4+x+2$, $y(x) = (x^3-x^2)/2$ (where $p(x) = (t(x)^2+4y(x)^2)/4$). In the next section, we provide a curve of embedding degree $k = 8$ where p is of 544 bits, and we have h_y of 16 bits, hence $p_{h_t,h_y}(T)$ has coefficients of more than 32 bits.

5.2 Generation of curves

In this section, we precise how to use Algorithm 5.1 in order to generate STNFS-secure pairing-friendly curves together with an efficient ate pairing. The code we used is accessible at

`https://gitlab.inria.fr/smasson/cocks-pinch-variant`.

Some parameters need to be provided to use the `CocksPinchVariant` function. Note that these parameters can also be specified in the `search.sage` script.

We target small embedding degrees $k \in \{5, \ldots, 8\}$, between the $k = 1$ curves [CMR17], and the BN and BLS12 curves [BD19]. We specify a parameter (k here) in our code as follows:

```
$ sage search.sage -k 5 (...)
$ sage search.sage -k 6 (...)
$ sage search.sage -k 7 (...)
$ sage search.sage -k 8 (...)
```

In order to obtain an efficient ate pairing together with a 128-bit security level, we need to specify several parameters for our modified Cocks-Pinch algorithm. We compensate the NFS variants by choosing appropriate size for the prime p in Section 5.2.1. In order to allow as many twists as possible, we explain the choice of discriminant in Section 5.2.2. Choosing low weight values for T as well as for the cofactors h_t and h_y reduces the cost of the pairing; we specify these parameters in Section 5.2.3. We aim to generate twist-secure and subgroup-secure elliptic curves as defined in Sections 2.8.4 and 2.8.5. It is sometimes hard to fill all the properties for $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}$. We explain our choices in Section 5.2.4. Finally, we provide the curves we found in Section 5.2.5.

5.2.1 Size of r and p for a 128-bit security level

In order to reach the 128-bit security level, we estimate the size of the subgroups we use: $E[r]$ and the multiplicative subgroup of order r of $\mathbb{F}_{p^k}^*$.

Curve subgroups. The situation is quite simple (see Section 2.8.1) on elliptic curves: we choose a subgroups of order r of 256 bits, which is set in Algorithm 5.1 with $\lambda_r = 256$.

Finite field subgroup. We study the case of finite fields as in Chapter 1. By construction, our Modified Cocks-Pinch algorithm provides curves resistant to the special variant of NFS: the prime p is parameterized by a polynomial with large enough coefficients to avoid the special NFS variant.

The situation for embedding degrees $k = 5$ and 7 is different than for $k = 6$ and 8 in the sense that the TNFS algorithm has no latitude for prime embedding degrees. Thus, the size of the finite field $\mathbb{F}_{p^k}$ can be reduced in the case of prime embedding degrees.

- For curves of embedding degree $k = 6$ and 8, the TNFS variant applies and we estimate that the finite field $\mathbb{F}_{p^k}$ needs to be roughly up to 4000 bits. Applying the NFS algorithm and its tower variant when p comes from the Modified Cocks-Pinch method, we reach the 128-bit security level when $\log_2(p^k) = 4032$ for a curve of embedding degree $k = 6$ (resp. $\log_2(p^k) = 4352$ for a curve of embedding degree $k = 8$). These sizes bring us a lower bound for $\log_2(p)$: $\lambda_p = 672$ (resp. $\lambda_p = 544$).
- For curves of embedding degree $k = 5$ and 7, the tower variant is not efficient and we can use a finite field $\mathbb{F}_{p^k}$ of roughly 3300 bits. After studying the discrete logarithm problem in these cases, we estimate that the 128-bit security level is reached for $\lambda_p = 663$ for $k = 5$, and $\lambda_p = 512$ for $k = 7$.

In the same way as for the embedding degree k, these options are specified with the parameters `--lambdar` and `--lambdap` in the `CocksPinchVariant` function.

5.2.2 Choice of discriminant

According to the efficiency considerations of Section 4.3, we target curves with as many twists as possible. Algorithm 5.1 uses a freely chosen discriminant D, but it is preferable to choose $D = 3$ (resp. $D = 4$) when the embedding degree is divisible by 6 (resp. by 4). Hence, for $k = 6$ (resp. $k = 8$), we set $D = 3$ (resp. 4) so that a sextic (resp. quartic) twist is available. In the `CocksPinchVariant` function, we use `-D 3` or `-D 4` to specify this parameter. For $k = 5$ and 7, there is no subfield between $\mathbb{F}_p$ and $\mathbb{F}_{p^k}$. In consequence, no twist optimization is possible

and there is no reason to choose a particular discriminant. For $k = 5$, we choose arbitrarily $D \approx 10^{10}$, which is well within the feasible range for the CM method (see also Section 5.2.5). In our code, we set `-D 10000000147` which is the first possible discriminant above 10^{10}. For $k = 7$, the size of p (512 bits) restricts us to small discriminants, since we must have $4p = t^2 + Dy^2$ with $\log_2(t), \log_2(y) \approx \log_2(r) = 256$. We decided to avoid $D = 3$ and 4 (even though no known attack takes advantage of these particular values), and chose $D = 20$ (`-D 20`).

5.2.3 Low weight parameters

Algorithm 5.1 iterates on different values of T which is a considerably large integer. The lift from $\mathbb{Z}/r\mathbb{Z}$ to $\mathbb{Z}$ also depends on the choice of cofactors h_t and h_y. In terms of efficiency, restricting to low weight integers results in a pairing speed-up. However, we want a large enough range of values so that we obtain a suitable curve. We studied the trade-off between a small hamming weight (or 2-NAF weight) of these integers and the number of possible curves generated.

Miller iteration. The integer T corresponds to the Miller iteration in Algorithm 4.3. The number of AddLine steps is determined by the Hamming weight of T (or the weight of the 2-NAF of T). We restrict to low weight T in order to get a more efficient Miller loop. There is a trade-off between the low weight of Miller iteration and the number of curves which can be very small depending on the properties we aim to satisfy. We found pairing-friendly curves for values of T which satisfy $\mathrm{HW}_{\text{2-NAF}}(T) \leq 5$, except in the case of $k = 7$ where the first curves we found were for $\mathrm{HW}_{\text{2-NAF}}(T) \leq 8$. This option is specified in the `CocksPinchVariant` function or in the `search.sage` file using the synthax `--T_choice "2-naf<=5"`. Note that we obtained a curve of embedding degree $k = 5$ using a T of hamming weight $\mathrm{HW}(T) = 4$. Thus, one can reproduce our search of curve using `--T_choice hamming=4` in the provided code.

Lifting coefficients h_t and h_y. The choice of the cofactors h_t and h_y in Algorithm 5.1 must be chosen in an annulus-like region given by the equation $2^{\lambda_p+1} \leq (t_0 + h_t r)^2 + D(y_0 + h_y r)^2 < 2^{\lambda_p+2}$ for p to have the desired bit length. We can restrict our search of curve for small Hamming weight cofactors so as to accelerate the exponentiation to the power c in the second part of the final exponentiation. Similarly with the cases of Sections 4.4.1, 4.4.2 and 4.4.3, we would like to compute the final exponentiation using a polynomial representation of p. In our modified Cocks-Pinch algorithm, we avoid univariate representation, but a multivariate expression $p(h_t, h_y, T)$ is still available when $k = 6$ or 8 (see Section 5.1.3). Using tiny coefficients h_t and h_y, this multivariate expression becomes a univariate one. Hence, we can restrict to both small cofactors only when $k = 5$ and 7. In practice, to satisfy the annulus-like region equation, we need to increase the size of one of the cofactors in the case $k = 5$.

As the final exponentiation is closely related to exponentiations to the power h_t (see Section 5.3), we restrict on as small T as possible. We found elliptic curves with the desired properties for $|h_t| < 4$ and $\mathrm{HW}_{\text{2-NAF}}(h_y) \leq 7$. This criterion is set using `--hty_choice "2-naf<=7:ht:max=4"`. In the case of $k = 7$, the bounds on p let us restrict on $|h_t|, |h_y| < 4$.

5.2.4 Twist-secure and subgroup-secure parameters.

We checked our curves for twist-security and subgroup-security (see Section 2.8.4, 2.8.5 and [BCM+15]). For each curve (say E), $\mathbb{G}_1$ and $\mathbb{G}_2$ are vulnerable to the twist and subgroup attacks, and $\mathbb{G}$ is also threatened by subgroup attacks. This makes five groups for which we can investigate the small subgroups:

- The curve E defined over $\mathbb{F}_p$ corresponding to the subgroup-security of $\mathbb{G}_1$,
- The quadratic twist of E, also defined over $\mathbb{F}_p$, corresponding to the twist-subgroup-security of $\mathbb{G}_1$,
- The curve corresponding to the subgroup-security of $\mathbb{G}_2$. In the case of $k = 5$ and 7, no twist is available and so we investigate $E(\mathbb{F}_{p^k})$. However, in the case of $k = 6$ (resp. $k = 8$), the twist compression lets us investigate a sextic (resp. quartic) twisted curve to E, defined over $\mathbb{F}_p$ (resp. $\mathbb{F}_{p^2}$).
- The quadratic twist of the curve corresponding to $\mathbb{G}_2$. In the case of $k = 6$ (or 8), it is another sextic (or quartic) twisted curve to E.
- The group $\mathbb{G}$ (with respect to $\Phi_k(p)$).

We aim to generate curves for which the cofactors of the orders of these five groups are not too large. We did not investigate the $\mathbb{G}$ subgroup-security: together with the Cocks-Pinch conditions, it would require finding parameters such that $\Phi_k(p)/r$ is prime or almost prime. With the sizes provided in Section 5.2.1, $1088 \leq \log_2(\Phi_k(p)/r) \leq 2816$ so it is difficult to factor this thousand-bit integer entirely and it will very unlikely be prime.

An option is available in our code using the `--check_small_subgroup_secure` parameter, which is a bitmap, more precisely a sum of bits corresponding to the four first items above. For example, `--check_small_subgroup_secure 11` corresponds to filter curves which are twist-secure and subgroup-secure for $\mathbb{G}_1$, and twist-secure for $\mathbb{G}_2$ (as $11 = \overline{1011}^2$). This parameter is also related to authorized cofactors. We provide in our code:

`--required_cofactor` is a useful option for obtaining an efficient group law on the curve (for instance, $\#E(\mathbb{F}_p) = 4 \pmod r$ is necessary for the Montgomery model).

`--allowed_size_cofactor` is an integer for allowing subgroup of small size (checking the subgroup attack is not expensive for very small subgroups).

`--automatic_cofactor` is an option when a factor automatically divides some of the curve orders we are considering.

5.2.5 Our new curves

Many parameters can be specified in our code. The documentation is available on the repository. Setting the parameters, we are able to generate pairing-friendly elliptic curves of embedding degree five to eight. Recall that from the prime p, the discriminant D, the trace t and the conductor y, we use the complex multiplication (CM) theory of Section 2.5 to recover the curve equation.

We obtain elliptic curves of embedding degree five to eight. We denote by E^t the quadratic twist of E and $\tilde{E}$ the degree d twist of E such that $E(\mathbb{F}_{p^k})[r] \simeq \tilde{E}(\mathbb{F}_{p^{k/d}})[r]$. For the remainder of this section, the notation p_N denotes an arbitrary prime of N bits.

Curve of embedding degree 5

Using the parameters provided in the previous sections, we run the `search.sage` script as follows:

```
sage search.sage -k 5 -D 10000000147\
                 --hty_choice ht:max=4  --restrict_i '[1]'\
                 --save --T_choice hamming=4\
                 --lambdap 663 --lambdar 256\
                 --check_small_subgroup_secure 15 --required_cofactor 4\
                 --spawn 4 --parallel-mode hy 0 4294967296
```

On `houblon.loria.fr`, a fraction $4/2^{39}$ of the search space is processed in 128 seconds (wall-clock time). The following curve was found by job $24165/2^{36}$:

```
C=CocksPinchVariantResult(5,10000000147,0xe000000000008000,\
                          1,ht=3,hy=0x11e36418c7c8b454,\
                          max_B1=600)
```

As we decided to choose the largest possible discriminant, we need to compute efficiently the Hilbert class polynomial in order to obtain the j-invariant of the curve. The `classpol` package from Andrew Sutherland provides the algorithm described in [Sut10, ES10]. A specificity of this software is that it computes the class polynomial modulo p directly. For our discriminant $D \approx 10^{10}$, the computation took less than two hours on one core and we obtain a polynomial H_D of degree 8321. One could have obtained this polynomial faster using the `doubleeta` parameter instead of `j`. The next step is to get a j-invariant by finding a root of this polynomial (it completes in about twenty minutes).

From the j-invariant, we derive the curve equation using Equation (2.5). We obtain a curve $E : y^2 = x^3 - 3x + b_5$ defined over $\mathbb{F}_p$ with

$$\begin{aligned} b_5 = {} & \texttt{0x3dd2d2b0b2e68770bf01b41946ab867390cf9ecc4a858004fc769c} \\ & \texttt{278f079574677c7db3e7201c938b099f85eb6e85f200b95a80b24fdb} \\ & \texttt{df584098d690c6b91b21d00f52cc79473a11123b08ab2a616b4a4fbf} \\ p = {} & \texttt{0x40000138cd26ab94b86e1b2f7482785fa18f877591d2a4476b4760} \\ & \texttt{217f860bfe8674e2a4610d669328bda13044c030e8cc836a5b363f2d} \\ & \texttt{4c8abcab71b12091356bb4695c5626bc319d38bf65768c5695f9ad97}. \end{aligned}$$

In particular, the curve is twist-secure and subgroup-secure for both $\mathbb{G}_1$ and $\mathbb{G}_2$ (given a cofactor 4):

$$\#E(\mathbb{F}_p) = 2^2 \cdot p_{405} \cdot r \quad \#\tilde{E}(\mathbb{F}_{p^5}) = p_{2393} \cdot (2^2 \cdot p_{405} \cdot r) \cdot r$$

$$\#E^t(\mathbb{F}_p) = 2^2 \cdot p_{661} \quad \#E^t(\mathbb{F}_{p^5}) = p_{2649} \cdot (2^2 \cdot p_{661})$$

$$r = \texttt{0x9610000000015700ab80000126012600c4007000a800e000f000200040008001}$$

The additional parameters to obtain the curve from Algorithm 5.1 are :

$$T = 2^{64} - 2^{61} + 2^{15}, D = 10^{10} + 147, i = 1, h_t = 3, h_y = \texttt{0x11e36418c7c8b454}$$

and $\mathbb{F}_{p^5}$ can be defined as $\mathbb{F}_p[x]/(x^5 - 5)$.

Curve of embedding degree 6

In the case of $k = 6$, we choose the parameters so that it allows to get sextic twists and sparse T, h_t and h_y as described above. A search of curve can be done using

```
sage search.sage -k 6 --D 3 \
                 --hty_choice '2-naf<=7,ht:max=4' \
                 --save --T_choice '2-naf<=5' \
                 --lambdap 672 --lambdar 256 \
                 --check_small_subgroup_secure 7 --required_cofactor 4 \
                 --allowed_automatic_cofactor 720 --allowed_cofactor 420 \
                 --allowed_size_cofactor 10
```

We obtain four interesting curves and one of them is given by

```
C=CocksPinchVariantResult(6,3,0xefffffffffffffe00000000000000000, \
                          1,ht=-1,hy=0xffbbfffffffffffffc020, \
                          allowed_cofactor=420,allowed_size_cofactor=10, \
                          max_B1=600)
```

The corresponding elliptic curve is $E : y^2 = x^3 - 1$, defined over $\mathbb{F}_p$ with

$$p = \texttt{0x9401ff90f28bffb0c610fb10bf9e0fefd59211629a7991563c5e468}\\ \texttt{d43ec9cfe1549fd59c20ab5b9a7cda7f27a0067b8303eeb4b31555cf4}\\ \texttt{f24050ed155555cd7fa7a5f8aaaaaaad47ede1a6aaaaaaaab69e6dcb}$$

This curve is twist-secure and subgroup-secure for $\eta = 8$:

$$\#E(\mathbb{F}_p) = 2^2 \cdot p_{414} \cdot r \quad \#\tilde{E}(\mathbb{F}_p) = 3 \cdot p_{414} \cdot r$$

$$\#E^t(\mathbb{F}_p) = 2^2 \cdot 3 \cdot 7 \cdot p_{665} \quad \#E^t(\mathbb{F}_p) = 13 \cdot 19 \cdot p_{664}$$

$$r = \texttt{0xe0fffffffffffc400000000000003ff1000000000000020000000000000001}$$

The additional parameters to obtain the curve from Algorithm 5.1 are :

$$T = 2^{128} - 2^{124} - 2^{69}, D = 3, i = 1, h_t = -1, h_y = 2^{80} - 2^{70} - 2^{66} - \texttt{0x3fe0}$$

and $\mathbb{F}_{p^6}$ can be defined as $\mathbb{F}_p[x]/(x^6 - 2)$.

Curve of embedding degree 7

We decided to generate a curve of embedding degree $k = 7$ with a small discriminant $D = 20$ (see Section 5.2.2). The Hilbert class polynomial H_{20} is easy to compute: $H_{20}(x) = x^2 - 1264000x - 681472000$. We search a curve using the following command. In practice, a parallel search is preferred.

```
sage search.sage -k 7 --D 5 \
                 --hty_choice 'max=3' \
                 --save --T_choice '2-naf<=8'
                 --lambdap 512 --lambdar 256 \
                 --check_small_subgroup_secure 3 \
                 --required_cofactor 4 \
                 --allowed_automatic_cofactor 720 --allowed_cofactor 420 \
                 --allowed_size_cofactor 10
```

We obtain suitable curve parameters:

```
C=CocksPinchVariantResult(7,20,0x5fffb820248, \
                          6,ht=-2,allowed_cofactor=1232, \
                          allowed_size_cofactor=10, \
                          max_B1=600)
```

The corresponding curve is obtain using a root of the Hilbert class polynomial modulo p. We get that a curve $E : y^2 = x^3 - 3u^2x + b_7u^3$ defined over $\mathbb{F}_p$ with

$$\begin{aligned} b_7 = {} & \texttt{0x15d384c76889d377dd63600fbe42628e0c386a3e87} \\ & \texttt{915790188d944845aab2b649964f386dc90b3a9b612} \\ & \texttt{0af5da9a2aaead5e415dd958c5cfa80ea61aac268b0} \\ p = {} & \texttt{0x8f591a9876a6d2344ae66dd7540ea2fd28174755d1} \\ & \texttt{6c4ae5c5cd5c1d208e639271b48c8ba7453c95a2a9b} \\ & \texttt{e6434f2455504d419f13e35062aa5ebbc49ecfd30f9} \\ u = {} & \texttt{11} \end{aligned}$$

satisfies

$$\#E(\mathbb{F}_p) = 2^2 \cdot 3^2 \cdot p_{251} \cdot r \quad \#E^t(\mathbb{F}_{p^7}) = 2^5 \cdot 5 \cdot p_{504}$$

$$r = \texttt{0xb63ccd541c3aa13c7b7098feb312eecf5648fd215c0d2916714b429d14e8f889}$$

The additional parameters to obtain the curve from Algorithm 5.1 are :

$$T = 2^{43} - 2^{41} - \texttt{0x47dfdb8}, D = 20, i = 6, h_t = -2, h_y = 0$$

and $\mathbb{F}_{p^7}$ can be defined as $\mathbb{F}_p[x]/(x^7 - 2)$.

Curve of embedding degree 8

The search of the $k = 8$ curve can be done using the command

```
sage search.sage -k 8 --D 1 \
                 --hty_choice '2-naf<=7,ht:max=4' \
                 --save --T_choice '2-naf<=5' \
                 --lambdap 544 --lambdar 256 \
                 --check_small_subgroup_secure 7 --required_cofactor 4 \
                 --allowed_automatic_cofactor 720 --allowed_cofactor 420 \
                 --allowed_size_cofactor 10
```

We finally obtain parameters using:

```
C8=CocksPinchVariantResult(8,4,0xffc00020ffffffc,1,ht=1,hy=0xdc04, \
                           allowed_cofactor=420,allowed_size_cofactor=10, \
                           max_B1=600)
```

We derive the curve $E : y^2 = x^3 + 2x$ defined over $\mathbb{F}_p$ with

$$\begin{aligned} p = {} & \texttt{0xbb9dfd549299f1c803ddd5d7c05e7cc0373d9b1ac15b} \\ & \texttt{47aa5aa84626f33e58fe66943943049031ae4ca1d2719b} \\ & \texttt{3a84fa363bcd2539a5cd02c6f4b6b645a58c1085e14411} \end{aligned}$$

which satisfies

$$\#E(\mathbb{F}_p) = 2^2 \cdot 3^2 \cdot 5 \cdot 41 \cdot p_{275} \cdot r \qquad \#E^t(\mathbb{F}_p) = 2^4 \cdot p_{540} \quad \#\tilde{E}(\mathbb{F}_{p^2}) = 2 \cdot 89 \cdot p_{824} \cdot r$$

$$r = \texttt{0xff0060739e18d7594a978b0ab6ae4ce3dbfd52a9d00197603fffdf0000000101}$$

The additional parameters to obtain the curve from Algorithm 5.1 are :

$$T = 2^{64} - 2^{54} + 2^{37} + 2^{32} - 4, D = 4, i = 1, h_t = 1, h_y = \texttt{0xdc04}$$

and $\mathbb{F}_{p^8}$ can be defined as $\mathbb{F}_p[x]/(x^8 - 5)$.

5.3 Pairing cost

In this section, we estimate the cost of the ate pairing in the case of our four curves generated by the modified Cocks-Pinch method. By construction, the ate pairing is optimal in the sense of Definition 4.8: we have tweaked the Cocks-Pinch algorithm so that the k-th root of unity T mod r is as small as possible. In the context of the ate pairing in these curves, it corresponds to the Miller iteration: $A_{r,T}(Q,P) = f_{T,Q}(P)^{(p^k-1)/r}$. The size of the Miller iteration defines how far the pairing is from optimal. We generated curves where $\log_2(T) = \log_2(r)/\varphi(k)$, which corresponds to the optimal case.

We continue the cost estimation by studying the Miller loop and the final exponentiation separately. Our curves of embedding degree $k = 6$ (resp. $k = 8$) has been constructed so that it has sextic (resp. quartic) twists. Hence, from the study of Section 4.3, many optimizations are available for these two curves. The story is slightly different for the two curves of prime embedding degree: the curves do not have any twist defined over a subfield of $\mathbb{F}_{p^k}$ ($\mathbb{F}_{p^k}$ does not have any proper subfield except $\mathbb{F}_p$). Thus, denominators are computed during Algorithm 4.3, line computation formulas are different from $j = 0, 1728$ curves, and the final exponentiation computation cannot use any possible parameterization of the primes p and r. As for BN, BLS12 and KSS16 curves, we count the number of multiplications over $\mathbb{F}_p$.

5.3.1 The Miller loop

Curves with twists. We begin with the estimation for the curves of embedding degree $k = 6$ and 8 which is very similar to the BN, BLS12 and KSS16 curves. Our two curves have been constructed so that they have as many twists as possible. In consequence, we can use the optimizations of Section 4.3.2 in order to compute efficiently the lines, omit denominators, etc. The study of Sections 4.4.1 and 4.4.2 is very similar to the case of curves of embedding degree $k = 6$ and discriminant $D = -3$. We roughly manipulate elements of $\mathbb{F}_{p^6}$ and $\mathbb{F}_p$ instead of $\mathbb{F}_{p^{12}}$ and $\mathbb{F}_{p^2}$. We finally obtain:

$$\begin{aligned}\mathbf{c}_{\text{MillerLoop}} =&(\log_2(T) - 1)(3\mathbf{m} + 6\mathbf{s} + 2\mathbf{m}) + (\log_2(T) - 2)(13\mathbf{m} + \mathbf{s}_6)\\ &+ (\text{HW}_{\text{2-NAF}}(T) - 1)(11\mathbf{m} + 2\mathbf{s} + 2\mathbf{m} + 13\mathbf{m}).\end{aligned}$$

The Miller step in Section 4.4.3 is closely related to the case of our $k = 8$ curve. Again, it corresponds to computations on $\mathbb{F}_{p^8}$ and $\mathbb{F}_{p^2}$ instead of $\mathbb{F}_{p^{16}}$ and $\mathbb{F}_{p^4}$. The Miller cost for the $k = 8$ curve is:

$$\begin{aligned}\mathbf{c}_{\text{MillerLoop}} =&(\log_2(T) - 1)(2\mathbf{m}_2 + 8\mathbf{s}_2 + 4\mathbf{m}) + (\log_2(T) - 2)(8\mathbf{m}_2 + \mathbf{s}_8)\\ &+ (\text{HW}_{\text{2-NAF}}(T) - 1)(9\mathbf{m}_2 + 5\mathbf{s}_2 + 4\mathbf{m} + 8\mathbf{m}_2).\end{aligned}$$

Algorithm 5.2: AddLine(S, Q, P) and DoubleLine(S, P)

Input. $S \in E(\mathbb{F}_{p^k})$, $Q \in E(\mathbb{F}_{p^k})$, $P \in E(\mathbb{F}_p)$.	**Input.** $S \in E(\mathbb{F}_{p^k})$,$P \in E(\mathbb{F}_p)$.
Output. The line $\ell_{S,Q}$ at P, The point $S+Q$.	**Output.** The tangent line $\ell_{S,S}$ at P, The point $[2]S$.
$(X, Y, Z, Z_2) \leftarrow S$	$(X, Y, Z, Z_2) \leftarrow S$
$(x_Q, y_Q) \leftarrow Q$	$(x_P, y_P) \leftarrow P$
$(x_P, y_P) \leftarrow P$	$t_1 \leftarrow Y^2$
$t_1 \leftarrow x_Q \cdot Z_2 - X$	$t_2 \leftarrow 4X \cdot t_1$
$t_2 \leftarrow y_Q \cdot Z \cdot Z_2 - Y$	**if** $a = -3u^2$ *for a small* $u \in \mathbb{F}_p$ **then**
$t_3 \leftarrow t_1^2$	$\quad t_3 \leftarrow 3(X - uZ_2) \cdot (X + uZ_2)$
$t_4 \leftarrow t_1 \cdot t_3$	**else**
$t_5 \leftarrow X \cdot t_3$	$\quad t_3 \leftarrow 3X^2 + a \cdot Z_2^2$
$\mathbf{X} \leftarrow t_2^2 - (t_4 + 2t_5)$	$\mathbf{X} \leftarrow t_3^2 - 2t_2$
$\mathbf{Y} \leftarrow t_2 \cdot (t_5 - \mathbf{X}) - Y \cdot t_4$	$\mathbf{Y} \leftarrow t_3 \cdot (t_2 - \mathbf{X}) - 8t_1^2$
$\mathbf{Z} \leftarrow Z \cdot t_1$	$\mathbf{Z} \leftarrow Z \cdot 2Y$
$\lambda_d \leftarrow \mathbf{Z}$	$\lambda_d \leftarrow \mathbf{Z} \cdot Z_2$
$\lambda_n \leftarrow \lambda_d \cdot (y_P - y_Q) - t_2 \cdot (x_P - x_Q)$	$\lambda_n \leftarrow \lambda_d \cdot y_P - 2t_1 - t_3 \cdot (Z_2 \cdot x_P - X)$
return $((\lambda_n, \lambda_d), \mathbf{S} = (\mathbf{X}, \mathbf{Y}, \mathbf{Z}, \mathbf{Z}^2))$	**return** $((\lambda_n, \lambda_d), \mathbf{S} = (\mathbf{X}, \mathbf{Y}, \mathbf{Z}, \mathbf{Z}^2))$

Algorithm 5.3: VerticalLine(S, P)

Input. $S \in E(\mathbb{F}_{p^k})$, $P \in E(\mathbb{F}_p)$.
Output. The vertical line $\ell_{S,-S}$ at P.

$(X, Y, Z, Z_2) \leftarrow S$; $(x_P, y_P) \leftarrow P$
return $(\mu_n = Z_2 \cdot x_P - X, \mu_d = Z_2)$

Curves of prime embedding degree. In the case of curves of prime embedding degree, the story is slightly different. There is no subfield between $\mathbb{F}_p$ and $\mathbb{F}_{p^k}$ and so no compression can be done using twists. Hence, the Miller loop is computed using multiplications over $\mathbb{F}_{p^k}$. Moreover, we need to compute denominators and so the Update 1 and 2 are more expensive. Finally, the Update 3 is necessary and costs $2\mathbf{m}_k$ for updating with the vertical line $v_Q(P)$.

We reuse Algorithm 4.3 and design the line functions when there is no compression for $\mathbb{G}_2$, and when no subfield factor can be omitted. These functions are detailed as Algorithms 5.2 and 5.3. The input point S as well as the output point $\mathbf{S}$ in these algorithms are in modified Jacobian coordinates (see Section 2.8.2). Note that our curves of embedding degree $k = 5$ and 7 have Weierstrass equations E_{-3,b_5} and $E_{-3\cdot 11^2, b_7 \cdot 11^3}$ which leads to an optimization in the Doubling step, as explained in Remark 2.27.

Counting the number of multiplications and squarings in the latter algorithms, we obtain that when the embedding degree is prime,

$$\mathbf{c}_{\text{DoubleLine}} = 6\mathbf{m}_k + 4\mathbf{s}_k + 2k\mathbf{m}, \quad \mathbf{c}_{\text{AddLine}} = 10\mathbf{m}_k + 3\mathbf{s}_k, \quad \mathbf{c}_{\text{VerticalLine}} = k\mathbf{m},$$
$$\mathbf{c}_{\text{Update1}} = 4\mathbf{m}_k + 2\mathbf{s}_k, \quad \mathbf{c}_{\text{Update2}} = 4\mathbf{m}_k, \quad \mathbf{c}_{\text{Update3}} = 2\mathbf{m}_k.$$

We deduce from Equation (4.3) the Miller step cost for our curves of prime embedding degree:

$$\begin{aligned}\mathbf{c}_{\text{MillerLoop}} =&(\log_2(T)-1)(6\mathbf{m}_k+4\mathbf{s}_k+2k\mathbf{m}+k\mathbf{m})+(\log_2(T)-2)(4\mathbf{m}_k+2\mathbf{s}_k)\\&+(\mathrm{HW}_{\text{2-NAF}}(T)-1)(10\mathbf{m}_k+3\mathbf{s}_k+k\mathbf{m}+4\mathbf{m}_k)\\&+k\mathbf{m}+2\mathrm{Nb}_{-1,T}\mathbf{m}_k+\mathbf{i}_k.\end{aligned}$$

5.3.2 Final exponentiation

As in Section 4.3.3, we use the decomposition of the exponent$(p^k-1)/r$ into two factors to compute the final exponentiation:

$$\frac{p^k-1}{r}=\frac{p^k-1}{\Phi_k(p)}\cdot\frac{\Phi_k(p)}{r}$$

First part. The first part of the exponentiation is very similar to the case of BN, BLS12 and KSS16 curves: the factor $(p^k-1)/\Phi_k(p)$ is a polynomial in p whose coefficients are 0, 1 or -1. Recall that we always consider binomial extension fields so that the Frobenius can be computed efficiently (see Section 1.2.2). Thus, we use Frobenius powers to compute the first exponentiation, as in Section 4.3.3:

- If $k=5$, $(p^5-1)/\Phi_5(p)=p-1$ and the exponentiation costs one Frobenius, one inversion and one multiplication in $\mathbb{F}_{p^5}$. Note that we can omit the Frobenius power which appears in the computation of the inversion (see Section 1.2.3). Finally, $\mathbf{c}_{\text{FirstExp}}=(4\mathbf{m})+\mathbf{i}_5+\mathbf{m}_5$.
- If $k=6$, $(p^6-1)/\Phi_6(p)=(p+1)(p^3-1)$. The first exponentiation to the power $p+1$ costs one Frobenius and one multiplication in $\mathbb{F}_{p^6}$. The second part is computed using the fact that $a^{p^3-1}=\bar{a}/a=a^2/N(a)$ (where $\bar{a}$ denotes the conjugate of a in $\mathbb{F}_{p^6}$ seen as a quadratic extension of $\mathbb{F}_{p^3}$). It costs $2\mathbf{s}_3+3\mathbf{m}_3+\mathbf{i}_3$. Finally, $\mathbf{c}_{\text{FirstExp}}=\mathbf{f}_6+\mathbf{m}_6+2\mathbf{s}_3+3\mathbf{m}_3+\mathbf{i}_3$.
- If $k=7$, the situation is similar to the case when $k=5$. The first exponentiation costs $\mathbf{c}_{\text{FirstExp}}=(6\mathbf{m})+\mathbf{i}_7+\mathbf{m}_7$.
- If $k=8$, $(p^8-1)/\Phi_8(p)=p^4-1$ which costs only $\mathbf{c}_{\text{FirstExp}}=\mathbf{i}_8+\mathbf{m}_8$ because a^{p^4} is the conjugate of a in $\mathbb{F}_{p^8}$ seen as a quadratic extension of $\mathbb{F}_{p^4}$.

Second part. The second part of the exponentiation is more expensive and is specific to each curve. The key ingredient is the base-p representation of the exponent, since Frobenius powers p^i are computed efficiently. Notice that in Algorithm 5.1, we have $p\equiv(t-1)\equiv(t_0-1) \mod r$. Let c be such that $p+1-t_0=c\cdot r$. The expression $(\Phi_k(p)-\Phi_k(t_0-1))/r$ simplifies, and we obtain a nice generic formula in p and t_0 for each embedding degree. The actual expression depends on the exponent i in Algorithm 5.1, as well as on congruence conditions on T. We only detail a few examples. Formulas for the other cases can be obtained with the companion software mentioned in Section 4.4.

For $k=8$, we choose $D=4$ so that $\sqrt{-D}=2T^2$. When for instance we choose $i=1$ in Algorithm 5.1, we have $t_0=T+1 \bmod r$. This leads to the following expression, where h_u denotes the integer $(h_t+1)/2$.

$$\begin{aligned}\Phi_8(p)/r&=\Phi_8(t_0-1)/r+(p+t_0-1)(p^2+(t_0-1)^2)c,\\c&=((((h_u^2-h_u+h_y^2+1/4)T+h_y)T-h_y+1/4)T\\&\quad+h_u-1)T+h_u^2+h_y^2\end{aligned}\tag{5.1}$$

where $\Phi_8(t_0-1)/r = \Phi_8(T)/r = 1$ by construction. To raise to the power $\Phi_8(p)/r$, we use the fact that T is even to deal with fractional values in the exponent. The cost estimation of $\mathbf{c}_c$ is specific to each curve, and we give here the details when $i = 1$. Note that we provide the code for all cases in the code repository mentioned in Section 4.4. Since the first part of the final exponentiation has been done, we know that $a^{p^4+1} = 0$ so that $a^{-1} = a^{p^4} = \overline{a}$ where the conjugate is taken over the subfield $\mathbb{F}_{p^4}$. The formula below is specific to $i = 1$, but we let $T = 4U + 2V$ which is the most general form (with $V \in \{0, 1\}$). If we apply this to the parameters in Section 5.2.5, we can do some simplifications using $V = 0$ (in square brackets below). Here we use $\mathbf{c}_T$ to represent the cost of any set of operations whose cost is similar to $b = b^2$ followed by $b = (b^2)^U b^V$, although scheduling above is sometimes different.

$$\begin{array}{ll}
a_y = a^y;\ a_u = a^u;\ a_Q = a_y^y a_u^u;\ b = a_Q\overline{a_u}; & (2\mathbf{c}_u + 2\mathbf{c}_y + 2\mathbf{m}_k) \\
b = b^2;\ b = (b^2 a)^U b^V;\ b = b a_y; & (\mathbf{c}_T + 2\mathbf{m}_k) \\
b = b^2;\ [b = b a^V];\ b = (b^2)^U b^V;\ b = b\overline{a_y}; & (\mathbf{c}_T + \mathbf{m}_k[+\mathbf{m}_k]) \\
b = b^2;\ b = (b^2 a)^U b^V;\ b = b a_u;\ b = b\overline{a}; & (\mathbf{c}_T + 3\mathbf{m}_k) \\
b = b^2;\ [b = b a^V];\ b = (b^2)^U b^V;\ b = b a_Q; & (\mathbf{c}_T + \mathbf{m}_k[+\mathbf{m}_k])
\end{array}$$

The cost of an exponentiation to the power c is $\mathbf{c}_c = 11\mathbf{m}_k + 4\mathbf{c}_T + 2\mathbf{c}_u + 2\mathbf{c}_y$ in general, and $2\mathbf{m}_k$ less if $V = 0$.

For $k = 6$, we choose $D = 3$ so that $\sqrt{-D} = 2T - 1$. We obtain expressions that vary slightly depending on i, and on the congruence class of h_t mod 2 and T mod 3. It also appears that it is more convenient to compute the cube of the pairing. When for instance we choose $i = 1$ in Algorithm 5.1, and that $T \bmod 3 = 1$ and $h_t \bmod 2 = 1$, we have the following expression, where $u = (h_t + 1)/2$, $w = h_y/2$, and $T' = T - (T \bmod 3) = T - 1$:

$$\begin{aligned}
3\Phi_6(p)/r &= 3\Phi_6(t_0 - 1)/r + 3(p + t_0)c, \\
3c &= ((3u^2 + 9w^2 - 3u - 3w + 1)T' + \\
&\qquad 3u^2 + 9w^2 - 6w)T' + 3u^2 + 9w^2 + 3u - 9w. \qquad (5.2)
\end{aligned}$$

Raising to the power $3\Phi_k(p)/r$ thus has the following cost (we give an upper bound on all possible congruence conditions). We use $\mathbf{c}_u$, $\mathbf{c}_w$, $\mathbf{c}_T$ and $\mathbf{c}_{T'}$ to denote the cost of raising to the powers u, w, T and $T' = T - (T \bmod 3)$, respectively.

$$\mathbf{c}_{\textsc{SecondExp},k=6} = (\mathbf{c}_T + \mathbf{f}_k + 2\mathbf{s}_k + 4\mathbf{m}_k) + (12\mathbf{m}_k + 2\mathbf{s}_k + 2\mathbf{c}_u + 2\mathbf{c}_w + 2\mathbf{c}_{T'}).$$

For $k = 5$ and $k = 7$, we use $p = (t_0 - 1) + c\Phi_k(T)$ to reduce $\Phi_k(p)/r = \Phi_k(p)/\Phi_k(T)$ (as a rational function in T) in the form

$$\Phi_k(p)/r = \sum_{0 \le j \le k-2} p^j a_j(c, T).$$

The exact expression of the coefficients $(a_j)_j$ depends on k and i, and so does the cost of raising to these powers. For example, for $k = 5$ and $i = 2$, we have

$$(a_j)_{0 \le j \le 3} = (-cT^3 - T + 1, -cT^3 - (c + 1)T + 1, cT^2 + c + 1, c).$$

By applying this method, we found that for $k = 5$ and $k = 7$, raising to the power $\Phi_k(p)/r$ costs at most

$$\mathbf{c}_{\textsc{SecondExp},k\in\{5,7\}} = 2\mathbf{i}_k + (k - 2)(\mathbf{f}_k + \mathbf{c}_T + 2\mathbf{m}_k) + \mathbf{c}_c + \mathbf{m}_k$$

where the two inversions can be saved in the favorable case $i = 1$, and $\mathbf{c}_T$ and $\mathbf{c}_c$ are the costs of raising to the powers T and c, respectively.

5.4 Comparison of curves

We compare the four curves generated in Section 5.2.5 with the state of the art: BN and BLS12 curves [AFK+13], KSS16 curves [CLN10, §4], and $k = 1$ curves [CMR17] presented at the beginning of this chapter.

5.4.1 Elliptic curve scalar multiplication in $\mathbb{G}_1$ and $\mathbb{G}_2$.

Our generation of curve leads to large prime values (up to eleven 64-bit words instead of eight for BN and BLS12 curves). The scalar multiplication cost on $\mathbb{G}_1$ is not affected by slow finite field multiplications because our curves benefit of other improvements: BN (resp. BLS) curves parameters for 128 bits of security lead to scalar multiplications $[k]P$ on $\mathbb{G}_1$ and $\mathbb{G}_2$ with $\log_2(k) \approx 448$ (resp. 300). For our curves of embedding degree five to eight, we choose r of minimal size (256 bits to withstand the Pollard rho attack). Some curves get benefits of efficient group law arithmetic: curves of embedding degree 5, 7, and 8 can use the Montgomery coordinates. The $k = 6$ curve uses the efficient formulas available for $a = 0$ curves, widely used in practice. The Gallant–Lambert–Vanstone (GLV) method can be performed on $k = 6$ and $k = 8$ curves in order to reduce the number of doubling and addition steps. Over $\mathbb{G}_2$, the scalar multiplication is often accelerated by using a twist of the curve. The trick is available for curves of degree 6 and 8, but not for $k = 5$ and 7. Even if the main topic of this paper is about pairing computations, various protocols also compute scalar multiplications. Curves of embedding degree 5 and 7 do not benefit of twists, so the cost over $\mathbb{G}_2$ is too expensive for practical applications. Finally, no GLV optimization is possible in the case of the $k = 5$ curve which has a large discriminant $D \approx 10^{10}$.

5.4.2 Pairing timing estimation

In Sections 4.4 and 5.3, we estimated a pairing cost with a number of finite field operations. This number is not relevant for a comparison in the sense that the field $\mathbb{F}_p$ is of different size depending on the curves we study. For instance, a multiplication in the 3072-bit field of the $k = 1$ curves of [CMR17] is much more expensive than in the 512-bit prime field of our $k = 7$ Cocks-Pinch curve. Hence, we compare the pairings using an estimation of how long it would cost in practice. To do so, we measure the arithmetic operations over $\mathbb{F}_p$ for various sizes of prime p on the same machine (a Intel Core i7-8700 CPU, 3.20GHz with TurboBoost disabled). Thus, we estimate the pairing cost multiplying the estimated time for $\mathbf{m}$ by the number of $\mathbb{F}_p$ multiplications for a pairing. We warn the reader that we did not implement the pairing entirely.

From the considerations of Section 1.2.1, we obtain in Table 5.1 the measurements of the base field arithmetic with the different primes involved.

From this estimation, we derive an expected timing for the different pairings in Table 5.2. Again, we warn the reader that timings of Table 5.2 are not real pairing computations, as we simply used as a base the arithmetic operations of RELIC [AG], and the multiplication costs that we detailed in Sections 4.4, 5.3 and [CMR17]. This being said, for the curves where an actual implementation of the optimal ate pairing is available with RELIC (BN and BLS12 curves), the estimation that we obtain is within 10% of the actual computation time. This gives reasonable confidence for the validity of the other projected timings.

Miller loop. We obtain a faster Miller loop for $k = 6$ and $k = 8$ curves compared to BN and BLS12 curves. The $k = 8$ curve has a shorter Miller loop (64-bit) compared to the BN and BLS12 ones (117-bit). The $k = 6$ curve has a sextic twist that allows to compute $f_{T,Q}(P)$ on $\mathbb{F}_p$ of 672

Prime size	Building block for	$\mathbb{F}_p$ multiplication
$192 < \log_2(p) \leq 256$		32ns
$320 < \log_2(p) \leq 384$	KSS16	65ns
$384 < \log_2(p) \leq 448$	BN, BLS12	85ns
$448 < \log_2(p) \leq 512$	BN, BLS12, $k = 7$	106ns
$512 < \log_2(p) \leq 576$	$k = 8$	129ns
$576 < \log_2(p) \leq 640$		154ns
$640 < \log_2(p) \leq 704$	$k = 5$, $k = 6$	181ns*
$3008 < \log_2(p) \leq 3072$	[CMR17]	3800ns**

Table 5.1: $\mathbb{F}_p$ multiplication timing for RELIC on a Intel Core i7-8700 CPU, 3.20GHz with TurboBoost disabled; *Estimation because no measurement is available for 11 machine-word primes; **Measured with GNU MP

bits, compared to a quadratic field of 922 bits for BN and BLS12 curves. As for the cases $k = 5$ and $k = 7$, the Miller loop is not as efficient because no twist is available, and the computation is done over $\mathbb{F}_{p^k}$. Comparisons between $k = 6$ and $k = 7$ curves show that using a curve with twists is a better option than having a short Miller loop. The best option is obviously to have a short Miller loop *and* a curve with twists, as for $k = 8$ curves.

Final exponentiation. The rewriting tricks used in Section 5.3.2 for the final exponentiation apply for any curve obtained with Algorithm 5.1 with the optimization $r = \Phi_k(T)$. For $k = 6$ and $k = 8$ the cofactor is smaller, and the discriminant $D = 3$, resp. $D = 4$, gives formulas that are as good as for BN and BLS12 curves. For $k = 5$ and $k = 7$ curves, the exponentiation is less efficient because fast cyclotomic squaring formulas are not available.

Total cost. Table 5.2 shows that our new pairing is almost as efficient as the optimal ate pairing on the BLS12 and KSS16 curves. Given the nature of Table 5.2 which gives *estimated* timings, it is however more appropriate to say that the performance difference is within the error margin. Additionally, we estimate that the optimal ate pairing on our $k = 8$ curve is up to 22 times more efficient than the Tate pairing on $k = 1$ curves [CMR17].

Remark 5.2. We also estimated the cost of a Tate pairing on Cocks-Pinch curves without twists for $k = 5, 7$ where the first input point P has coordinates in $\mathbb{F}_p$ and the second input point Q has coordinates in $\mathbb{F}_{p^k}$. The doublings and additions of points now take place in $\mathbb{F}_p$, but the accumulations still require $\mathbb{F}_{p^k}$ arithmetic. The costs are $\mathbf{c}_{\text{AddLine}} = 8\mathbf{m} + 3\mathbf{s} + 2k\mathbf{m}$, $\mathbf{c}_{\text{DoubleLine}} = 7\mathbf{m} + 4\mathbf{s} + 2k\mathbf{m}$, $\mathbf{c}_{\text{VerticalLine}} = k\mathbf{m}$, $\mathbf{c}_{\text{Update1}} = 2\mathbf{m}_k + \mathbf{s}_k$, and $\mathbf{c}_{\text{Update2}} = 2\mathbf{m}_k$ ($m_d \in \mathbb{F}_p$ is not computed since it would be 1 after the final exponentiation). The Miller length is $(k-1)$ times longer, of length $\log_2 r$ instead of $\log_2 T$. The accumulation steps $\mathbf{c}_{\text{Update1}}$ cost $(k-1)$ times more. The Miller loop of a Tate pairing would cost roughly $(\log_2(r)-1)(\mathbf{c}_{\text{DoubleLine}} + \mathbf{c}_{\text{VerticalLine}}) + (\log_2(r)-2)\mathbf{c}_{\text{Update1}} + (\text{HW}_{\text{2-NAF}}(r)-1)(\mathbf{c}_{\text{AddLine}} + \mathbf{c}_{\text{VerticalLine}} + \mathbf{c}_{\text{Update2}}) + \mathbf{i}_k$. For $k = 5$ we have $\log_2(r) = 256$ and $\text{HW}_{\text{2-NAF}}(r) = 39$, which gives $18585\mathbf{m}$. For $k = 7$ we have $\log_2(r) = 256$ and $\text{HW}_{\text{2-NAF}}(r) = 90$, which gives $31817\mathbf{m}$. The gain of swapping P and Q in the Tate pairing is counterbalanced by the much longer Miller loop and finally, the Miller loop of a Tate pairing would require more multiplications in $\mathbb{F}_p$ compared to an optimal ate Miller loop costing $14496\mathbf{m}$ for $k = 5$ and $18830\mathbf{m}$ for $k = 7$ (see Table 5.2).

Curve	**Prime**	**Miller loop** time estimation	**Exponentiation** time estimation	**Total**	time estimation
$k = 5$	663-bit	14527**m** 2.6ms	9809**m** 1.8ms	24336**m**	4.4ms
$k = 6$	672-bit	4601**m** 0.8ms	3871**m** 0.7ms	8472**m**	1.5ms
$k = 7$	512-bit	18381**m** 1.9ms	13439**m** 1.4ms	31820**m**	3.4ms
$k = 8$	544-bit	4502**m** 0.6ms	7056**m** 0.9ms	11558**m**	1.5ms
BN	446-bit	11620**m** 1.0ms	5349**m** 0.5ms	16969**m**	1.4ms
BLS12	446-bit	7805**m** 0.7ms	7723**m** 0.7ms	15528**m**	1.3ms
BN	462-bit	12180**m** 1.3ms	5727**m** 0.6ms	17907**m**	1.9ms
BLS12	461-bit	7685**m** 0.8ms	6283**m** 0.7ms	13968**m**	1.5ms
KSS16	339-bit	7691**m** 0.5ms	18235**m** 1.2ms	25926**m**	1.7ms
$k = 1$	3072-bit	4651**m** 17.7ms	4100**m** 15.6ms	8751**m**	33.3ms

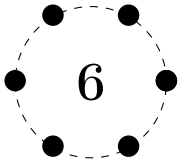
Table 5.2: Pairing cost and timing extrapolation from Table 5.1

Comparison with concrete measurements. We did not provide a concrete implementation of the optimal pairing on our new curves of embedding degree $k = 5$, 6, 7 and 8, but we presented an estimation of the pairing time in the differente cases. The RELIC library provides the computation of the optimal ate pairing on different curves. Using the same machine as for the measurements of Table 5.1, we obtained optimal ate measurements in order to validate our estimations.

- On a BLS12 curve with $\log_2(p) = 446$, the optimal pairing costs 1.4ms, corresponding to our estimation in Table 5.2 within 10%.
- The RELIC library also provides the implementation for a BLS12 curve for a larger prime p. We remark that they do not use a 461-bit prime p as in Table 5.2, but a slightly smaller one (a 455-bit integer). It does not affect the practical cost because both primes are written using 8 machine words, leading to the same integer arithmetic cost. Moreover, the Miller loop has a NAF representation with only one less non-zero bit. Then measurement on this curve also validates our estimation within 10% as we measured 1.6ms.
- An implementation for our $k = 8$ curve is also provided in the RELIC library. We measured that the optimal ate pairing costs 2.1ms, which is more expensive than the expected cost of Table 5.2. The implementation for the BLS12 curves is older and has been optimized for longer than the $k = 8$ construction, explaining this difference. The measurements show that the difference comes from the final exponentiation: the Miller step costs 0.69ms which is close to our estimation, while the final exponentiation costs 1.4ms. The major optimization

in the $k = 8$ pairing implementation is the decomposition of the final exponent $\Phi_k(p)/r$, which is different from the BLS12 case.

All these concrete measurements correspond to our estimations and it gives reasonable confidence for the validity of the timings expected for our new curves. In particular, from our estimations on the $k = 5$ and $k = 7$ curves, an implementation of these curves would not lead to a competitive pairing. Finally, optimizations on the final exponentiation of the $k = 8$ curve, and a concrete implementation on the $k = 6$ curve could be interesting in order to compare with the current computations on BLS12 curves.



Representation of endomorphism rings of supersingular curves

In this chapter, we implement algorithms related to specific quaternion algebras. The goal of this chapter is to bring an explicit Deuring correspondence, i.e. a concrete representation of endomorphism rings of supersingular curves. In the first section, we recall basic facts about lattices useful in our context. Then, we explain how to represent particular elements of quaternion algebras together with a way of computing the arithmetic explicitly. In a third section, we define orders of these algebras and represent them as lattices of rank 4. We also introduce left and right ideals of orders, represented in the same way as orders. Sections 6.4, 6.5 and 6.6 investigate the Deuring correspondence, and introduce the necessary algorithms for computing the bijection through isogenies. Finally, we go into detail on the computation of equivalent ideals using an algorithm provided in [GPS20], and give a brief conclusion on the efficiency of the algorithms we provide. We warn the reader that the algorithms presented in this chapter come from recent works [KLPT14, GPS20]. The implementation aspect was not explicitly provided and the work of this chapter is to provide practical algorithms. Recently, Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit and Benjamin Wesolowski presented a post-quantum signature called SQISign [FKL$^+$20] which is also related to the computation of ideals in a quaternion algebra. This construction will be presented at Asiacrypt 2020. The implementation of our algorithms is publicly accessible at

`https://gitlab.inria.fr/smasson/endomorphismsthroughisogenies`.

Summary

6.1 Lattices

We consider in this section discrete subgroups of finite rank, also called lattices. In our context, a lattice $\mathcal{L}$ can be represented as $\mathcal{L} = \mathbb{Z}\langle v_1, \ldots, v_d \rangle$ where $v_1, \ldots, v_d$ are vectors of $\mathbb{Q}^d$. It means that we consider integer linear combinations of $v_1, \ldots, v_d$. One can also represent the lattice $\mathcal{L}$ using a matrix whose rows are the v_i represented with d rational coefficients, i.e. a matrix $L \in \mathbf{M}_d(\mathbb{Q})$. From now on, lattices are denoted with calligraphic letters and their matrices with usual letters. From the matrix L, the lattice elements correspond to the $(n_1, \ldots, n_d)L$, where $n_1, \ldots, n_d \in \mathbb{Z}$. Hence, writing a vector $x = (x_1, \ldots, x_d) \in \mathbb{Q}^d$, the vector x belongs to $\mathcal{L}$ if, and only if, all coordinates of xL^{-1} are integers. Indeed, the vector xL^{-1} corresponds to the coordinates of x in the basis of $\mathcal{L}$ given by the matrix L.

We recall facts about matrix reduction in the context of integer matrices. $\mathbf{SL}_d(\mathbb{Z})$ denotes the special linear group of matrices of $\mathbf{M}_d(\mathbb{Z})$ with determinant ± 1.

Definition 6.1 (Equivalence classes in $\mathbf{M}_d(\mathbb{Z})$)**.** *Two matrices L and L' of $\mathbf{M}_d(\mathbb{Z})$ are equivalent if there exists a matrix $U \in \mathbf{SL}_d(\mathbb{Z})$ such that $L = UL'$.*

Two equivalent matrices represent the same lattice $\mathcal{L}$ and the change of variables between the two bases is given in the matrix $U \in \mathbf{SL}_d(\mathbb{Z})$. It is common to reduce matrices in order to get a coset representative that satisfy particular conditions. Reduction algorithms cost depends on the dimension d of the matrices we look for. A survey is presented in [NV10]. The Hermite–Korkine–Zolotarev (HKZ) algorithm provides a lattice basis that reaches the shortest vector norms, but is not very efficient in large dimension. The Lenstra–Lenstra–Lovász (LLL) algorithm is more efficient but the new basis vectors only bounded by a constant (and do not reach the shortest one). In the following sections, we consider $d = 4$ so that these reduction algorithms are very efficient. From now on, we consider the Hermite Normal Form of a matrix.

Definition 6.2 (Hermite Normal Form from [Coh10, page 67])**.** *A matrix $M = (m_{i,j}) \in \mathbf{M}_d(\mathbb{Z})$ of determinant $\det(M) \neq 0$ is in Hermite Normal Form (HNF) if it satisfies the following conditions:*

- *M is an upper triangular matrix, i.e. $m_{i,j} = 0$ if $i > j$.*
- *For every i, we have $m_{i,i} > 0$.*
- *For every $j > i$, we have $0 \leq m_{i,j} < m_{i,i}$.*

Theorem 6.3 (from [Coh10, page 67])**.** *Let $M \in \mathbf{M}_d(\mathbb{Z})$. Then, there exists a unique matrix $H \in \mathbf{M}_d(\mathbb{Z})$ in HNF of the form $H = UM$ with $U \in \mathbf{SL}_d(\mathbb{Z})$.*

Remark 6.4. Definition 6.2 and Theorem 6.3 also apply for rectangular matrices of $\mathbf{M}_{d,d'}(\mathbb{Z})$ with $d \geq d'$. A matrix in HNF has the following shape

$$\begin{pmatrix} * & * & \ldots & * \\ 0 & * & \ldots & * \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \ldots & 0 & * \\ 0 & \ldots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \ldots & 0 & 0 \\ 0 & \ldots & 0 & 0 \end{pmatrix}$$

where the first d' rows form a matrix in HNF.

Computing the HNF of a matrix M corresponds to applying linear algebra using only integer arithmetic. Cohen provides an algorithm for computing the Hermite Normal Form in [Coh10, page 68] and its complexity is $O(d^3)$ integer operations. In our context, we consider matrices with rational coefficients, but the lattices are still defined as $\mathbb{Z}$-linear combination of the basis vector. Hence, it also makes sense to consider the HNF of these matrices of $\mathbf{M}_d(\mathbb{Q})$. Algorithms naturally extend by multiplying by the lcm of denominators of the coefficients, as in the case of number fields (see [Coh10, page 73]).

Definition 6.5 (Extended Hermite Normal Form)**.** *Let $M = (m_{i,j}) \in \mathbf{M}_d(\mathbb{Q})$ and c be the lowest common multiple of the denominators of the $m_{i,j}$ (for $1 \leq i, j \leq d$). The matrix M is in extended Hermite Normal Form if the integer-valued matrix $\tilde{M} = cM$ satisfies the conditions of Definition 6.2.*

In particular, the extended HNF of a matrix has (numerators of) coefficients bounded by the (numerators of the) pivots of their columns, and is upper triangular, similarly to the integer case. In other words, for a matrix $M = (m_{i,j}) \in \mathbf{M}_d(\mathbb{Q})$, there exists a unique matrix $H \in \mathbf{M}_d(\mathbb{Z})$ in HNF of the form $H = UcM$ where c is the common denominator of the $m_{i,j}$, and $U \in \mathbf{SL}_d(\mathbb{Z})$.

We now recall generalities on lattices. More precisely, we give the definitions and the computations of the index of a sublattice and the interesection of two lattices (see [Coh10, Exercise 4.18]).

Index of a sublattice. Given a lattice $\mathcal{L}' \subset \mathcal{L}$, the index of $\mathcal{L}'$ in $\mathcal{L}$ is the integer $[\mathcal{L} : \mathcal{L}'] := \#(\mathcal{L}/\mathcal{L}')$. If $(v'_1, \ldots, v'_d)$ is a basis of $\mathcal{L}'$, we obtain the index by writing $v'_1, \ldots, v'_d$ in the basis of $\mathcal{L}$: it corresponds to the determinant of the transformation matrix. As we have seen, the change of basis is computed using $L'L^{-1}$ (its rows are the $v'_1, \ldots, v'_d$ in the basis of $\mathcal{L}$) so that $[\mathcal{L} : \mathcal{L}'] = \det(L'L^{-1})$.

Intersection of lattices. Given two lattices $\mathcal{L} = \mathbb{Z}\langle v_1, \ldots, v_d\rangle$ and $\mathcal{L}' = \mathbb{Z}\langle v'_1, \ldots, v'_d\rangle$, the intersection $\mathcal{L} \cap \mathcal{L}'$ is composed of vectors that can be written in the form $\sum_i m_i v_i$ as well as $\sum_i m'_i v'_i$, for integers m_i, m'_i. Using the Hermite Normal Form of the matrix composed of L and L' vertically joint, one obtains a unimodular matrix $U \in \mathbf{SL}_{2d}(\mathbb{Z})$ (we denote here U with four dimension-d matrices) such that

$$\begin{pmatrix} U_1 & U'_1 \\ U_2 & U'_2 \end{pmatrix} \begin{pmatrix} L \\ L' \end{pmatrix} = \begin{pmatrix} \triangledown \\ 0 \end{pmatrix}.$$

The matrix composed of L and L' has $2d$ rows and d columns so that the Hermite Normal Form is upper triangular and at least its d last rows are zeros. The upper part represents the lattice $\mathcal{L} \cup \mathcal{L}'$. Moreover, $U_2 L + U'_2 L' = 0$. The matrix $U_2 L$ represents the lattice $\tilde{\mathcal{L}} = \left\{ mU_2 L, m \in \mathbb{Z}^d \right\}$. It is exactly the intersection lattice $\mathcal{L} \cap \mathcal{L}'$. Indeed,

- As $(m_1, \ldots, m_d)U_2 \in \mathbb{Z}^d$, we obtain that $\tilde{\mathcal{L}} \subset \mathcal{L}$. Given the relation $U_2 L = -U'_2 L'$, the inclusion $\tilde{\mathcal{L}} \subset \mathcal{L}'$ is very similar.
- For $x \in \mathcal{L} \cap \mathcal{L}'$, there exists $m \in \mathbb{Z}^d$ and $m' \in \mathbb{Z}^d$ such that $mL = m'L'$. Then, $x = -m'U_2'^{-1}U_2 L$ and $-m'U_2'^{-1} \in \mathbb{Z}^d$. Hence, x is in the lattice $\tilde{\mathcal{L}}$ and so $\tilde{\mathcal{L}} = \mathcal{L} \cap \mathcal{L}'$.

In the next sections, we will represent particular lattices using the matrix notations. More precisely, we will consider vectors of $\mathbb{Q}^4$, corresponding to elements of an algebra of dimension 4, called a quaternion algebra.

6.2 Quaternion algebras

From now on, we study non-commutative $\mathbb{Q}$-algebras of dimension 4 defined with four generators, as in Section 2.5.2. Let p be a prime. We are interested in quaternion algebras H ramified only at p and ∞: it means that $H \otimes \mathbb{Q}_p$ and $H \otimes \mathbb{R}$ are division algebras (i.e. algebras where division by non-zero elements is possible). We refer to [Voi20, Section 14.1] for details on general quaternion algebras. In this particular case of ramification, these quaternion algebras can always be written $H_{-a,-b} = \mathbb{Q}\mathbf{1} + \mathbb{Q}\mathbf{i} + \mathbb{Q}\mathbf{j} + \mathbb{Q}\mathbf{k}$ where $\mathbf{i}^2 + a = 0$, $\mathbf{j}^2 + b = 0$ and $\mathbf{k} = \mathbf{ij}$. For a given prime p, Pizer gives in [Piz80, page 368] a couple of integers (a, b) such that the algebra $H_{-a,-b}$ satisfies these properties of ramification. In particular,

- If $p = 3 \bmod 4$, $(a, b) = (1, p)$.
- If $p = 5 \bmod 8$, $(a, b) = (2, p)$.
- If $p = 1 \bmod 8$, $(a, b) = (r, p)$ where $r = 3 \bmod 4$ and $(p/r) = -1$. Assuming that the generalized Riemann hypothesis holds, there exists $r = O(\log_2^2(p))$ satisfying these conditions.

For our computations, we will represent a quaternion x by a quadruplet of rationals (x_1, x_2, x_3, x_4). The addition of two quaternions is simply the vector addition, and the multiplication of two quaternions is computed in Algorithm 6.1, using the matrix

$$\mathbf{M} = \begin{pmatrix} \mathbf{1} & \mathbf{i} & \mathbf{j} & \mathbf{k} \\ \mathbf{i} & -a & \mathbf{k} & a\mathbf{j} \\ \mathbf{j} & -\mathbf{k} & -b & -b\mathbf{i} \\ \mathbf{k} & a\mathbf{j} & -b\mathbf{i} & -ab \end{pmatrix} = M_1\mathbf{1} + M_2\mathbf{i} + M_3\mathbf{j} + M_4\mathbf{k}.$$

Algorithm 6.1: Multiply(x, y)

Input. $x = x_1\mathbf{1} + x_2\mathbf{i} + x_3\mathbf{j} + x_4\mathbf{k}$,
$y = y_1\mathbf{1} + y_2\mathbf{i} + y_3\mathbf{j} + y_4\mathbf{k}$.
Output. $x \times y = r_1\mathbf{1} + r_2\mathbf{i} + r_3\mathbf{j} + r_4\mathbf{k}$.

for i between 1 and 4 **do**
  $r_i \leftarrow x \cdot M_i \cdot y^t$
end for
return $(r_1, \ldots, r_4)$

Quaternion algebras have similarities with number fields: an element of $H_{-a,-b}$ has a degree 2 minimal polynomial by construction. Hence, each element defines a quadratic field. Similarly to the case of number fields, the conjugate of a quaternion x is defined to be $\bar{x} := x_1 - \mathbf{i}x_2 - \mathbf{j}x_3 - \mathbf{k}x_4$. This involution gives rise to the trace and the norm:

$$\mathrm{Tr}(x) = x + \bar{x} = 2x_1 = 2xM_1\mathbf{1} \qquad N(x) = xM_1\bar{x} = x_1^2 + ax_2^2 + bx_3^2 + abx_4^2$$

and the inverse of a non-zero quaternion x can be computed as $x^{-1} = \bar{x}/N(x)$.

Definition 6.6 (Integral quaternion)**.** *A quaternion x is integral if its minimal polynomial has integer coefficients.*

An element is integral if, and only if, $\mathrm{Tr}(x)$ and $N(x)$ are in $\mathbb{Z}$. The set of integral elements is defined as in the case of number fields, but the situation is slightly different: this set is not always a ring, as illustrated in Example 6.7. We denote this set to be the integers of $H_{-a,-b}$. In the next section, we consider particular subsets of the integers, having the property of being a ring.

Example 6.7. Consider the quaternion algebra $H_{-1,-1}$ defined with $\mathbf{i}^2 = -1$, $\mathbf{j}^2 = -1$ and $\mathbf{k} = \mathbf{ij}$. Then, $\alpha = \mathbf{i}$ and $\beta = (3\mathbf{i} + 4\mathbf{j})/5$ are integral but neither $\alpha\beta$ nor $\alpha + \beta$ is integral.

6.3 Orders and ideals

We now consider particular subrings of the set of integers that are called orders.

6.3.1 Orders in quaternion algebras

Definition 6.8 (Order of a quaternion algebra)**.** *An order of $H_{-a,-b}$ is a full rank lattice that is also a subring of $H_{-a,-b}$. An order is maximal if there is no other order that contains this order.*

As a lattice, an order $\mathcal{O}$ can also be represented using a matrix. We consider orders of $H_{-a,-b}$ so a basis of an order can also be represented with four quaternions. In the following, we denote $O \in \mathbf{M}_4(\mathbb{Q})$ to be a matrix representing an order $\mathcal{O}$, and we write $\mathcal{O} = \mathbb{Z}\langle b_1, \dots, b_4\rangle$ when seen as a subset of $H_{-a,-b}$.

Definition 6.9 (Discriminant of an order)**.** *The discriminant of an order is the integer* $\mathrm{Disc}(\mathcal{O}) = \det(\mathrm{Tr}(b_i b_j)_{i,j})$.

The discriminant can be computed using matrices: $\mathrm{Disc}(O) = \det(2OM_1O^t)$. For a suborder $\mathcal{O}' \subset \mathcal{O}$, $\mathrm{Disc}(\mathcal{O}') = f^2\,\mathrm{Disc}(\mathcal{O})$ where f is the index $[\mathcal{O} : \mathcal{O}']$. Indeed, if $\mathcal{O}' \subset \mathcal{O}$, then there is a change of basis matrix T such that $O' = TO$ and then, $\mathrm{Disc}(O') = \det(2O'M_1O'^t) = \det(2TOM_1O^tT^t) = \mathrm{Disc}(O) \cdot \det(T)^2$. A maximal order has discriminant $-p^2$ so that it is easy to check the maximality of an order when the ramification is known.

6.3.2 Ideals in quaternion orders

We now consider ideals of orders. In non-commutative rings, we talk about *left* and *right* ideals.

Definition 6.10 (Left and right ideals)**.** *Let $\mathcal{O}$ be an order. Then,*

- *a $\mathcal{O}$-left ideal $\mathcal{I}$ is a subgroup of $\mathcal{O}$ such that $\mathcal{O}\mathcal{I} \subset \mathcal{I}$,*
- *a $\mathcal{O}$-right ideal $\mathcal{I}$ is a subgroup of $\mathcal{O}$ such that $\mathcal{I}\mathcal{O} \subset \mathcal{I}$.*

In particular, an ideal $\mathcal{I}$ is included in $\mathcal{O}$ (as it is a subgroup) and it makes sense to consider the index $[\mathcal{O} : \mathcal{I}]$ of the lattice $\mathcal{I}$ inside $\mathcal{O}$.

Definition 6.11 (Norm of an ideal)**.** *The norm $N(\mathcal{I})$ of an ideal $\mathcal{I}$ is the integer* $N(\mathcal{I}) := \gcd(\{N(x)$ *for* $x \in \mathcal{I}\})$.

Remark 6.12. Voight proves in [Voi20, Theorem 16.1.3] that the norm of a $\mathcal{O}$-(left or right) ideal satisfies $N(\mathcal{I}) = \sqrt{[\mathcal{O} : \mathcal{I}]}$.

Two-generator representation. An ideal $\mathcal{I}$ is a lattice of dimension 4, but also a $\mathcal{O}$-module that can be expressed with two generators: for any $\alpha \in \mathcal{I}$,

- a $\mathcal{O}$-left ideal $\mathcal{I}$ can be expressed as $\mathcal{O} \cdot \alpha + \mathcal{O} \cdot \beta$ for some $\beta \in \mathcal{I}$,
- a $\mathcal{O}$-right ideal $\mathcal{I}$ can be expressed as $\alpha \cdot \mathcal{O} + \beta \cdot \mathcal{O}$ for some $\beta \in \mathcal{I}$.

Algorithm 6.2 provides a two-generator representation for an $\mathcal{O}$-left ideal $\mathcal{I}$ as $\mathcal{O} \cdot N(\mathcal{I}) + \mathcal{O} \cdot \alpha$.

Algorithm 6.2: SecondGenerator(I, m) from [GPS20]

Input. I a matrix representing an ideal $\mathcal{I}$ of an order $\mathcal{O}$,
m an integer.
Output. An element $\alpha \in \mathcal{I}$ such that $\mathcal{I} = \mathcal{O} \cdot N(\mathcal{I}) + \mathcal{O} \cdot \alpha$.

$N \leftarrow N(\mathcal{I})$.
repeat
  (increase m if needed)
  $a \leftarrow \text{Random}(1, m)$
  $b \leftarrow \text{Random}(1, m)$
  $c \leftarrow \text{Random}(1, m)$
  $d \leftarrow \text{Random}(1, m)$
  $\alpha \leftarrow (a, b, c, d) \times I$
until $N(\alpha) = NM$ for an integer M such that $\gcd(N, M) = 1$.
return α

Definition 6.13 (Left and right orders)**.** *Let $\mathcal{I}$ be a (left or right) ideal $\mathcal{I}$ of an order $\mathcal{O}$. Then,*

$$O_L(\mathcal{I}) := \{x \in H_{-q,-p}, x\mathcal{I} \subset \mathcal{I}\} \qquad O_R(\mathcal{I}) := \{x \in H_{-q,-p}, \mathcal{I}x \subset \mathcal{I}\}.$$

Given a (left or right) ideal $\mathcal{I} = \mathbb{Z}\langle u_1, u_2, u_3, u_4\rangle$ of an order $\mathcal{O}$, its left and right orders are

$$O_L(\mathcal{I}) = \bigcap_{i=1}^{4} \{x \in H_{-q,-p}, xu_i \subset \mathcal{I}\}, \qquad O_R(\mathcal{I}) = \bigcap_{i=1}^{4} \{x \in H_{-q,-p}, u_i x \subset \mathcal{I}\}$$

and their lattices correspond to the intersections $\bigcap_{i=1}^{4} \mathcal{I}u_i^{-1}$ and $\bigcap_{i=1}^{4} u_i^{-1}\mathcal{I}$ respectively. Thus, left and right orders can be computed using algorithms of Section 6.1. Recall that $u_1^{-1}\mathcal{I}$ corresponds to the lattice $\mathbb{Z}\langle \mathbf{1}, \frac{u_2\bar{u}_1}{N(u_1)}, \frac{u_3\bar{u}_1}{N(u_1)}, \frac{u_4\bar{u}_1}{N(u_1)}\rangle$.

Definition 6.14 (Connecting ideal)**.** *An ideal $\mathcal{I}$ is a connecting ideal between two orders $\mathcal{O}$ and $\mathcal{O}'$ if $O_L(\mathcal{I}) = \mathcal{O}$ and $O_R(\mathcal{I}) = \mathcal{O}'$.*

Equivalence classes of orders and ideals An order $\mathcal{O}$ is isomorphic to $\mathcal{O}'$ if there exists an element $b \in H_{-a,-b}$ such that $b\mathcal{O} = \mathcal{O}'b$. This leads to an equivalence relation: $\mathcal{O} \sim \mathcal{O}'$ if, and only if, $\mathcal{O}$ is isomorphic to $\mathcal{O}'$. We now consider fractional ideals, i.e. ideals of the form $q\mathcal{I}$ with $q \in \mathbb{Q}^*$ and $\mathcal{I}$ as above. Considering a (fractional) connecting ideal $\mathcal{I}$ between two orders $\mathcal{O}$ and $\mathcal{O}'$, we define the following equivalence relation: $\mathcal{I} \sim \mathcal{J}$ if, and only if, there exists $b \in H_{-a,-b}$ such that $\mathcal{J} = \mathcal{I}b$ or $b\mathcal{I}$. If $\mathcal{I} \sim \mathcal{J}$, then $O_L(\mathcal{I}) \sim O_L(\mathcal{J})$ and $O_R(\mathcal{I}) \sim O_R(\mathcal{J})$.

6.3.3 Quotient of orders

We consider an order $\mathcal{O}$ of $H_{-a,-b}$ as above, and its quotient by the ideal $N\mathcal{O}$, for a given prime N. The quotient $\mathcal{O}/N\mathcal{O}$ is a $\mathbb{Z}/N\mathbb{Z}$-quaternion algebra, and Voight obtains in [Voi20, Proposition 2.2.8] that it is isomorphic to $\mathbf{M}_2((\mathbb{Z}/N\mathbb{Z})(\sqrt{-a}))$. From now on, we assume that $-a$ is a square modulo N so that the endomorphism matrices are defined over $\mathbb{Z}/N\mathbb{Z}$. Algorithm 6.3 explicitly computes the matrix corresponding to an endomorphism of $\mathcal{O}/N\mathcal{O}$.

Algorithm 6.3: MatrixModulo(f, N, r)

Input. f an endomorphism of $\mathcal{O}$,
N a prime integer,
r a root of $-a$ in $\mathbb{Z}/N\mathbb{Z}$.
Output. A matrix of $\mathbf{M}_2(\mathbb{Z}/N\mathbb{Z})$ representing f in $\mathcal{O}/N\mathcal{O}$.

$(a, b, c, d) \leftarrow f$.
return $\begin{pmatrix} a + br & -b(c + dr) \\ c - dr & a - br \end{pmatrix}$

In [KLPT14], Kohel et al. obtain particular endomorphisms of an ideal of norm N by looking at the quotient $\mathcal{O}/N\mathcal{O}$. They prove that there is a bijection between $\mathbf{M}_2(\mathbb{Z}/N\mathbb{Z})$ and $\mathbb{P}^1(\mathbb{Z}/N\mathbb{Z}) \times \mathbb{P}^1(\mathbb{Z}/N\mathbb{Z})$, and describe the left ideals of $\mathcal{O}/N\mathcal{O}$ in this representation.

Proposition 6.15 (from [KLPT14])**.** *The map*

$$\begin{array}{rcl} \mathbb{P}^1(\mathbb{Z}/N\mathbb{Z}) \times \mathbb{P}^1(\mathbb{Z}/N\mathbb{Z}) & \longrightarrow & \mathbf{M}_2(\mathbb{Z}/N\mathbb{Z}) \\ ((x : y), (u : v)) & \longmapsto & \begin{pmatrix} ux & uy \\ vx & vy \end{pmatrix} \end{array}$$

is a bijection, and a left ideal of $\mathcal{O}/N\mathcal{O}$ corresponds to $\mathbb{P}^1(\mathbb{Z}/N\mathbb{Z}) \times \{P_0\}$ for a fixed point of $P_0 \in \mathbb{P}^1(\mathbb{Z}/N\mathbb{Z})$.

Given $\alpha = x + y\mathbf{i} + z\mathbf{j} + t\mathbf{k}$, the elements of the ideal $\mathcal{O}\alpha/N\mathcal{O}$ are represented as matrices of the form

$$\frac{1}{x - yr} \begin{pmatrix} -bu(z + tr) & u(z - tr) \\ -bv(z + tr) & v(z - tr) \end{pmatrix} \text{ for } (u : v) \in \mathbb{P}^1(\mathbb{Z}/N\mathbb{Z})$$

through the bijection $\mathbf{M}_2(\mathbb{Z}/N\mathbb{Z}) \simeq \mathcal{O}/N\mathcal{O}$.

Given an ideal $\mathcal{I} = \mathcal{O}N + \mathcal{O}\alpha$ of norm N, an element γ is in $\mathcal{I}$ if, and only if, $\gamma \in \mathcal{O}\alpha \bmod N\mathcal{O}$. By Proposition 6.15, $\alpha \bmod N\mathcal{O}$ corresponds to a matrix of the form

$$\begin{pmatrix} ux & uy \\ vx & vy \end{pmatrix}$$

and an element γ of the ideal $\mathcal{O}\alpha/N\mathcal{O}$ is of the form

$$\begin{pmatrix} u_0x & u_0y \\ v_0x & v_0y \end{pmatrix}$$

for $(u_0 : v_0) \in \mathbb{P}^1(\mathbb{Z}/N\mathbb{Z})$. In Section 6.6.3, we consider elements of ideals and compute them by looking at $\mathcal{O}/N\mathcal{O}$.

6.4 Deuring correspondence

We now go into detail on the relation between maximal orders of quaternion algebras of Section 6.3.2 and the endomorphism rings of supersingular elliptic curves. From now on, we assume that p and q are two primes such that $H_{-q,-p}$ is a quaternion algebra ramified at p and ∞ as in [Piz80].

Proposition 6.16 (Deuring correspondence (from [Voi20, Proposition 42.4.7]))**.** *There is a one-to-one correspondence between the set of j-invariants of supersingular elliptic curves defined over $\mathbb{F}_{p^2}$ up to Galois conjugate and the set of isomorphism classes of maximal orders in the quaternion algebra $H_{-q,-p}$.*

Remark 6.17. The latter correspondence says that if E is a supersingular elliptic curve, it can be seen as a coset representative of its isomorphism class up to Galois conjugate. Then, $\mathrm{End}(E)$ is represented by a maximal order of $H_{-q,-p}$, representing the coset up to isomorphism. In this context, the quotient $\mathcal{O}/N\mathcal{O}$ is more intuitive: the subgroup $E[N]$ is isomorphic to $\mathbb{Z}/N\mathbb{Z}\times\mathbb{Z}/N\mathbb{Z}$, and so the coset representatives of $\mathcal{O}/N\mathcal{O}$ can be chosen as endomorphisms of a 2-dimensional $\mathbb{Z}/N\mathbb{Z}$-vector space, i.e. matrices of $\mathbf{M}_2(\mathbb{Z}/N\mathbb{Z})$.

Example 6.18. Let $p = 3 \bmod 4$ be a prime, and $E : y^2 = x^3 - x$ the elliptic curve defined over $\mathbb{F}_{p^2} = \mathbb{F}_p(i)$ with $i^2 = -1$. E is also defined over $\mathbb{F}_p$ and has discriminant $-D = -4$. These two properties lead to two endomorphisms $(x, y) \mapsto (-x, iy)$ and the Frobenius $\pi : (x, y) \mapsto (x^p, y^p)$. The curve E is supersingular and $\mathrm{End}(E)$ is a maximal order $\mathcal{O}$ of $H_{-1,-p}$. Looking at the minimal polynomial of the two latter endomorphisms, we obtain that they correspond to the quaternions $\mathbf{i}$ and $\mathbf{j}$. $\mathbf{j}(P) = -P$ for $P \in E[2]$ so that $(\mathbf{1} + \mathbf{j})(E[2]) = 0$ and it makes sense to consider $\frac{\mathbf{1}+\mathbf{j}}{2}$ as an endomorphism. A basis of the maximal order is given by $\mathcal{O} = \mathbb{Z}\langle \mathbf{1}, \mathbf{i}, \frac{\mathbf{1}+\mathbf{j}}{2}, \frac{\mathbf{i}+\mathbf{k}}{2}\rangle$. Note that it is similar to the case of quadratic fields, where the ring of integer sometimes contains fractional elements (the ring of integers of $\mathbb{Q}[\sqrt{-3}]$ is $\mathbb{Z}[\frac{1+\sqrt{-3}}{2}]$). Using the lattice representation of Section 6.1, $\mathcal{O}$ is represented with the matrix O, and its Hermite Normal Form is given by H:

$$O = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 \\ 0 & 1/2 & 0 & 1/2 \end{pmatrix} \qquad H = \begin{pmatrix} 1/2 & 0 & 1/2 & 0 \\ 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

In order to evaluate $\frac{1+\pi}{2}$ (corresponding to $\frac{\mathbf{1}+\mathbf{j}}{2}$) at a point P, one first computes Q such that $[2]Q = P$, and then computes $Q + \pi(Q)$.

Similarly, the endomorphism ring of a supersingular curve of j-invariant 0 is also well-known, and more generally, the computation of the endomorphism ring of small discriminant curves is possible by computing particular endomorphisms of small degrees. Except these particular curves, computing the maximal order corresponding to a supersingular elliptic curve (with no additional information) is a hard problem. Its complexity is exponential in $\log_2(p)$, even on a quantum computer. The security of some cryptographic protocols introduced in Chapter 3 relies on the difficulty of this problem.

In the latter example, we obtained that some endomorphisms require a division of point by 2 in order to be evaluated. More generally, given a representation of the endomorphism ring as a maximal order in a quaternion algebra, the evaluation of the endomorphisms may require a division by a large integer D. If $\phi : E \to E'$ is an isogeny of degree D, then, $D\,\mathrm{End}(E') \simeq \phi \circ \mathrm{End}(E) \circ \hat{\phi}$. Endomorphisms of E' can be computed through the isogeny ϕ and its dual, together with an additional division by D of the point considered.

Cost of endomorphism evaluation. We consider the elliptic curve of Example 6.18 so that evaluation $\mathbf{1}, \mathbf{i}, \mathbf{j}$ and $\mathbf{k}$ is almost free. We also consider an isogeny $\phi : E \to E'$ of degree D as above so that

$$\mathrm{End}(E') \simeq \frac{1}{D}\phi \circ \mathrm{End}(E) \circ \hat{\phi}.$$

Suppose that we know a quaternion description of $\mathrm{End}(E') \approx \mathcal{O}'$. In order to evaluate $\alpha' \in \mathrm{End}(E')$ at a point P, one first compute Q such that $[2D]Q = P$ and then apply ϕ, $2D\alpha' \in \mathbb{Z}\langle \mathbf{1}, \mathbf{i}, \mathbf{j}, \mathbf{k}\rangle$ seen as an element of $\mathcal{O}$, and finally $\hat{\phi}$. The cost of evaluating endomorphisms of E' is closely related to the cost of dividing a point by D, which is not easy to estimate because the point involved may be defined over an extension of the field where P is defined.

Example 6.19. Let $E : y^2 = x^3 + x$ defined over $\mathbb{F}_{p^2}$ where $p = 7$. Then, the full 2^3-torsion is defined over $\mathbb{F}_{p^2} = \mathbb{F}_p(u) = \mathbb{F}_p[x]/(x^2+1)$. The point $P = (3, 4)$ has order 2^3. We look for a point Q such that $[2]Q = P$, which means that $Q \in E[2^4]$. Defining $\mathbb{F}_{p^4} = \mathbb{F}_{p^2}(v) = \mathbb{F}_{p^2}[y]/(y^2 - (i+2))$, we obtain that $E[2^4]$ is fully rational over $\mathbb{F}_{p^4}$. The point $Q = ((2u+1)v+2u+3, (3u+3)v+2u+1)$ satisfies the condition $[2]Q = P$. Note that $Q + (u, 0)$, $Q + (0, 0)$ and $Q + (-u, 0)$ also satisfy this latter condition. More generally, if $[n]Q = P$, then $[n](Q + Q') = P$ for $Q' \in E[n]$.

We consider a point P of order $\prod_i p_i^{e_i}$ and a division by $D = \prod_j d_j^{f_j}$. Using the Chinese Remainder Theorem (CRT), P can be decomposed as $(P_i)_i \in \prod_i E'[p_i^{e_i}]$. We recall here the constructive group isomorphism of the CRT in the case of an elliptic curve.

Proposition 6.20 (Elliptic curve version of the Chinese Remainder Theorem)**.** *Let E be an elliptic curve defined over $\mathbb{F}_{p^2}$ such that the full $\prod_i p_i^{e_i}$-torsion is defined over $\mathbb{F}_{p^2}$. Then, there exists a group isomorphism between $E[\prod_i p_i^{e_i}]$ and $\prod_i E[p_i^{e_i}]$:*

$$\begin{array}{rcl} E[\prod_i p_i^{e_i}] & \longrightarrow & \prod_i E[p_i^{e_i}] \\ P & \longmapsto & ([\prod_{j\neq i} p_j^{e_j}]P)_i \end{array} \qquad \begin{array}{rcl} \prod_i E[p_i^{e_i}] & \longrightarrow & E[\prod_i p_i^{e_i}] \\ (P_i)_i & \longmapsto & \sum_j \left(\sum_{i\neq j} u_i p_i^{e_i}\right) P_j \end{array}$$

where $(u_i)_i$ are the Bézout coefficients of the prime power divisors of $\#E(\mathbb{F}_{p^2})$: $\sum_i u_i p_i^{e_i} = 1$.

Remark 6.21. The cost of the computation of the isomorphisms is an extended Euclidean algorithm which is $O(\log_2(\#E(\mathbb{F}_{p^2})))$ in the worst case, which is linear in $\log_2(p)$ by the Hasse bound (see Section 2.4).

Then, the division by D is done successively using all the prime power factors $d_j^{f_j}$ on the points $(P_i)_i$ of $p_i^{e_i}$-torsion:

- If $d_j^{f_j}$ is coprime with $p_i^{e_i}$, then the point $Q_{i,j}$ such that $[d_j^{f_j}]Q_{i,j} = P_i$ is simply $Q_{i,j} = [\mathrm{inv}_{\bmod p_i^{e_i}}(d_j^{f_j})]P_i$.
- Otherwise, it means that $d_j = p_i$ and the point $Q_{i,j}$ is a point of order $p_i^{e_i+f_j}$. Depending on the structure of the curve, the $E'[p_i^{e_i+f_j}]$ is defined over a field which can be very large. Writing $A_{i,j}, A'_{i,j}$ a basis of $E'[p_i^{e_i+f_j}]$, we write $P_i = n_{i,j}A_{i,j} + n_{i,j}A'_{i,j}$ and then find $Q_{i,j}$ by solving the equation $(2x - n_{i,j})A_{i,j} + (2y - n'_{i,j})A'_{i,j} = 0$. We investigate such equations in Section 6.5.

From the points $Q_{i,j}$, we obtain Q such that $[D]Q = P$ using the CRT isomorphism given in Proposition 6.20.

Algorithm 6.4 presents how to evaluate an endomorphism of a curve isogenous to the curve of Example 6.18.

Algorithm 6.4: Evaluate(β, P, ϕ)

Input. $\beta \in \mathcal{O}_2$, an endomorphism of E_2,
$P \in E_2(\mathbb{F}_{p^2})$,
$\phi : E_1 \to E_2$ an isogeny, where E_1 is the curve of Example 6.18.
Output. The point $\beta(P)$.

$\tilde{\phi} \leftarrow 2\deg(\phi)\beta \in \mathbb{Z}\langle \mathbf{1}, \mathbf{i}, \mathbf{j}, \mathbf{k}\rangle$.
Write $\tilde{\phi} = w_1\mathbf{1} + w_2\mathbf{i} + w_3\mathbf{j} + w_4\mathbf{k}$, $w_i \in \mathbb{Z}$.
$Q \leftarrow \frac{1}{2\deg(\phi)}P$.
$R \leftarrow \hat{\phi}(Q)$.
$S \leftarrow [w_1]R + [w_2]\mathbf{i}(R) + [w_3]\mathbf{j}(R) + [w_4]\mathbf{i}(\mathbf{j}(R))$.
$T \leftarrow \phi(S)$.
return T

In Section 6.6, we will look for the computation of maximal orders corresponding to supersingular curves, given additional information (for instance, the knowledge of an isogeny). Before going into detail on the endomorphism ring computation (closely related to evaluating endomorphisms), we deal with solving equations with curve points, as it will be necessary for Section 6.6.

6.5 Solving equations with curve points

We consider the following equation with unknowns $n_1, \ldots, n_k$

$$\sum_{i=1}^{d} n_i P_i = Q \tag{6.1}$$

for some fixed points P_i, Q of an elliptic curve E defined over $\mathbb{F}_{p^2}$. We assume that $\#E(\mathbb{F}_{p^2})$ and its factorization into primes $\prod_j \ell_j^{e_j}$ is known. Using the elliptic curve version of the Chinese Remainder Theorem above, it is sufficient to solve the equation on the prime power order subgroups of $E(\bar{\mathbb{F}}_p)$.

From now on, we assume that ℓ is a prime, and the P_i are of order ℓ^e. $E[\ell^e]$ is isomorphic to $\mathbb{Z}/\ell^e\mathbb{Z} \times \mathbb{Z}/\ell^e\mathbb{Z}$ and so we solve a linear system with a matrix of $\mathbf{M}_{d,2}(\mathbb{Z}/\ell^e\mathbb{Z})$. Section 6.5.1 explains how to decompose the points P_i in a given basis of $E[\ell^e]$, and Section 6.5.2 details the linear algebra for solving equations with points such as Equation (6.1).

6.5.1 Torsion decomposition

For any integer $e > 0$, the subgroup $E[\ell^e]$ is isomorphic to $\mathbb{Z}/\ell^e\mathbb{Z} \times \mathbb{Z}/\ell^e\mathbb{Z}$. A basis of $E[\ell^e]$ can be obtained using Algorithm 6.5. We recall that Φ_{ℓ^e} is the ℓ^e-th cyclotomic polynomial, and e_{ℓ^e} is the Weil pairing on $E[\ell^e]$. From now on, we denote $E[\ell^e] = \langle A, A' \rangle$ for two points A, A'.

Algorithm 6.5: TorsionBasis(ℓ, e, E)

Input. ℓ a prime,
e an integer,
E a supersingular elliptic curve defined over $\mathbb{F}_q$, $\gcd(\ell, q) = 1$.
Output. A basis A, A' of $E[\ell^e]$.

Find κ such that $E[\ell^e] \subset E(\mathbb{F}_{q^\kappa})$ and κ is minimal.
Find f the largest integer such that there is some ℓ^f-torsion over $\mathbb{F}_{q^\kappa}$.
$c \leftarrow \#E(\mathbb{F}_{q^\kappa})/\ell^f$
repeat
  $A \leftarrow [c]\text{Random}(E(\mathbb{F}_{q^\kappa}))$
  $A' \leftarrow [c]\text{Random}(E(\mathbb{F}_{q^\kappa}))$
  while $[\ell]A \neq 0$ **do**
    $A \leftarrow [\ell]A$
  end while
  while $[\ell]A' \neq 0$ **do**
    $A' \leftarrow [\ell]A'$
  end while
until $[\ell^{e-1}]A \neq 0$, $[\ell^{e-1}]A' \neq 0$ and $e_\ell([\ell^{e-1}]A, [\ell^{e-1}]A') \neq 1$
return (A, A')

Complexity. Algorithm 6.5 is probabilistic as it computes the basis by taking random points of order dividing ℓ^e. The point A has order exactly ℓ^e if $A \in E[\ell^e]$ and $A \notin E[\ell^{e-1}]$. It occurs with probability $\frac{\ell^{2e}-\ell^{2(e-1)}}{\ell^{2e}} = 1 - \ell^{-2}$. Hence, we expect to find A in $O(1)$ scalar multiplications. Similarly, we obtain A' in a distinct subgroup of $\langle A \rangle$ with probability $\frac{\ell^2-\ell}{\ell^2} = 1 - 1/\ell$ and so the basis is computed in $O(1)$ scalar multiplications on $E(\mathbb{F}_{q^\kappa})$. Determining κ requires studing the factorization of the ℓ^e-th division polynomial $\psi_{\ell^e}(x)$. One subgroup of the ℓ^e torsion is defined over a quadratic extension of $\mathbb{F}_{q^{\deg(F(x))}}$, where $F(x)$ is an irreducible factor of $\psi_{\ell^e}(x)/\psi_{\ell^{e-1}}(x)$. In the worst case, the subgroup is defined over an extension of degree $O(\ell^e)$ as $\deg(\psi_{\ell^e}) = (\ell^e - 1)/2$ for ℓ coprime with q. In order to get the full ℓ^e-torsion, we extend the curve to $\mathbb{F}_{q^{2\deg(F(x))k}}$, where k is the embedding degree of the curve with respect to ℓ^e, also in $O(\ell^e)$. Finally, $\kappa = O(\ell^{2e})$. The computation of $\#E(\mathbb{F}_{q^\kappa})$ does not affect the asymptotic complexity as we only consider supersingular curves. We obtain an overall complexity by estimating the cost of scalar multiplication over $E(\mathbb{F}_{q^\kappa})$. The order of $E(\mathbb{F}_{q^\kappa})$ is roughly $O(q^\kappa)$ and so the final complexity is $O(\kappa \log_2(q))$ multiplication over $\mathbb{F}_{q^\kappa}$, i.e. $O(\ell^{2e} \log_2(q))$ multiplications over $\mathbb{F}_{q^{O(\ell^{2e})}}$.

Remark 6.22. In practice, we will often consider elliptic curves defined over $\mathbb{F}_{p^2}$ with the ℓ^e-torsion rational, as in the case of SIDH curves (see Section 3.3.3). In this context, Algorithm 6.5 is very efficient. For instance, consider the 5-torsion and the 11^3-torsion of the elliptic curve defined by $y^2 = x^3 - x$ over $\mathbb{F}_{5323}$. The 5-torsion is fully rational over an extension of degree 8, but not over an extension of degree 4. However, the curve is defined over $\mathbb{F}_p$ and $p = 4 \cdot 11^3 - 1$ so the 11^3-torsion is fully rational over $\mathbb{F}_{p^2}$.

From A and A', a point $P \in E[\ell^e]$ decomposes as $P = nA + n'A'$ using Algorithm 6.6. The algorithm computes discrete logarithms in groups of powersmooth order: if $P = nA + n'A'$ with $n, n' < \ell^e$, then the bilinearity of the Weil pairing (see Theorem 4.4) shows that

$$e_{\ell^e}(P, A) = e_{\ell^e}(A', A)^n \text{ and } e_{\ell^e}(P, A') = e_{\ell^e}(A, A')^{n'}.$$

Once the pairings are computed (in $O(e\log_2(\ell))$ multiplications in $\mathbb{F}_{q^\kappa}$), the two discrete logarithms are computed using e discrete logarithms in a group of order ℓ, i.e. in $O(e\sqrt{\ell})$ multiplications using an adaptation of the algorithm of Section 1.1.1. Remark that if ℓ is very small, an exhaustive search in $O(\ell^2)$ is often sufficient for solving discrete logarithms in prime order ℓ subgroups. Finally, the overall complexity of Algorithm 6.6 is $O(e\sqrt{\ell})$ multiplications over $\mathbb{F}_{q^\kappa}$ if A and A' are defined over $\mathbb{F}_{q^\kappa}$, with $\kappa = O(\ell^{2e})$.

Algorithm 6.6: InBase(ℓ, e, A, A', P)

Input. ℓ a prime,
e an integer,
A, A' a basis of $E[\ell^e]$,
$P \in E[\ell^e]$.
Output. Two integers n_1 and n_2 such that $P = [n_1]A + [n_2]A'$.

$L \leftarrow$ the list of the $e_{\ell^e}(A, A')^{\ell^i}$ $(1 \le i < e)$.
$B \leftarrow$ the list of the $\sqrt{\ell}$ baby-steps of the group $\langle e_{\ell^e}(A, A')$.
$w_1 \leftarrow e_{\ell^e}(P, A)$
$w_2 \leftarrow e_{\ell^e}(P, A')$
$(n_1, n_2) \leftarrow (0, 0)$
for $s \in \{0, \ldots, e-1\}$ **do**
  $t_1, t_2 \leftarrow$ the DL of $w_1^{\ell^{e-1-s}}$ and $w_2^{\ell^{e-1-s}}$ using the baby-steps list B. ▷ See Section 1.1.1
  $(w_1, w_2) \leftarrow (w_1/L[1+s]^{t_1}, w_2/L[1+s]^{t_2})$.
  $(n_1, n_2) \leftarrow (n_1 + t\ell^s, n_2 + t\ell^s)$.
end for
return (n_1, n_2).

6.5.2 Solving the equation using linear algebra

Homogeneous equation. We begin with the case where $Q = 0_E$. Using Algorithm 6.6, the points P_i can be decomposed as $a_iA + a'_iA'$. Hence, Equation (6.1) can be transformed into

$$\sum_{i=1}^{d} n_i a_i A = 0 \text{ and } \sum_{i=1}^{d} n_i a'_i A' = 0.$$

Using a matrix representation, solutions of these equations correspond to the kernel of the matrix of the a_i, a'_i, i.e. a matrix of $\mathbf{M}_{d,2}(\mathbb{Z}/\ell^e\mathbb{Z})$:

$$(n_1, \ldots, n_d) \begin{pmatrix} a_1 & a'_1 \\ \vdots & \vdots \\ a_d & a'_d \end{pmatrix} = (0, 0).$$

In order to find a solution of the equations, we compute the Hermite Normal Form of the matrix of the a_i and the a'_i (seen in $\mathbb{Z}$) so that we get a unimodular transformation matrix $T \in \mathbf{SL}_d(\mathbb{Z})$ such that

$$T \begin{pmatrix} a_1 & a'_1 \\ \vdots & \vdots \\ a_4 & a'_4 \end{pmatrix} = \begin{pmatrix} \diagdown \\ 0 \end{pmatrix}.$$

There exists $i_0 \in \{1, 2\}$ such that for all $i > i_0$, $\sum_j T_{ij} a_j = T_{ij} a'_j = 0$ so that the $d-2$ last rows of T form a basis of the solutions of the equations. Any i-th row ($i > i_0$) provides a solution of the equation.

Non-homogeneous equation. Solving an equation of the form $\sum_{i=1}^{d} n_i P_i = Q$ for a non-zero $Q \in E[\ell^e]$ is very similar. Using the torsion decomposition $Q = bA + b'A'$ and $P_i = a_i A + a'_i A'$ for $1 \leq i \leq d$, the latter equation is equivalent to

$$\sum_{i=1}^{d} n_i a_i A - bA = 0 \text{ and } \sum_{i=1}^{d} n_i a'_i A' - b'A' = 0.$$

Similarly to the homogeneous case, we compute the HNF H of the matrix

$$\begin{pmatrix} a_1 & a'_1 \\ \vdots & \vdots \\ a_d & a'_d \\ b & b' \end{pmatrix}.$$

The transformation matrix T leads to the relations

$$\sum_{j=1}^{d} [T_{i,j}] P_j + [T_{i,d+1}] Q = 0 \text{ for } d + 1 - \mathrm{rk}(H) \leq i \leq d + 1$$

A solution of the equation is a vector of this lattice generated by the $\mathrm{rk}(H) + 1$ last rows of T, whose last coefficient is -1. Such a vector is obtained using the HNF of the column matrix C composed of the $\{T_{i,d+1}, d+1-\mathrm{rk}(H) \leq i \leq d+1\}$: we obtain a matrix $T' \in \mathbf{M}_{d+1-\mathrm{rk}(H)}(\mathbb{Z}/\ell^e\mathbb{Z})$ such that $\sum_i T'_{1,i} C_{i,1} = 1$. Thus, $-\sum_i T'_{1,i} T_{i+\mathrm{rk}(H)}$ is of the form $(n_1, \ldots, n_d, -1)$ and leads to a solution of the non-homogeneous equation.

Algorithm 6.7: SolveEquation$(P_1, \ldots, P_d, Q)$

Input. d points $P_1, \ldots, P_d$ of $E[\ell^e]$,
Q a point of $E[\ell^e]$.
Output. $n_1, \ldots, n_d$ integers such that $\sum_{i=1}^{d} n_i P_i = Q$.

$A, A' \leftarrow$ a basis of $E[\ell^e]$.
$M \leftarrow$ the matrix of $\mathbf{M}_{d,2}(\mathbb{Z})$ whose i-th row is InBase(ℓ, e, A, A', P_i).
if $Q = 0_E$ **then**
  $H, T \leftarrow$ the HNF of M together with the transformation matrix.
  if H has a zero row **then**
    return the corresponding row in T.
  else
    return $\varnothing$
  end if
else
  $N \leftarrow$ the matrix of $\mathbf{M}_{1,2}(\mathbb{Z})$ whose row is InBase$(\ell, e, A, A', -Q)$.
  $H, T \leftarrow$ the HNF of the matrix of M and N vertically joint, together with the transformation matrix.
  if H does not have a zero row **then**
    return $\varnothing$
  else
    $H', T' \leftarrow$ the HNF of the column matrix $C = (T_{i,d+1})_{\mathrm{rk}(H)+1 \le i \le d+1}$.
    return $-\sum_i T'_{1,i} T_{i+\mathrm{rk}(H)}$
  end if
end if

Algorithm 6.7 first computes d times Algorithm 6.6, and then computes simply linear algebra with matrices of $\mathbf{M}_{d,2}(\mathbb{Z}/\ell^e\mathbb{Z})$, costing $O(d)$ multiplications modulo ℓ^e. Thus, the overall complexity is $O(de\sqrt{\ell})$ multiplications in $\mathbb{F}_{q^{2(\ell-1)\ell^{e-1}}}$.

Solving equations with curve points will be useful for the computations of Section 6.6 where we compute endomorphism rings of supersingular curves through isogenies.

6.6 Maximal orders through isogenies

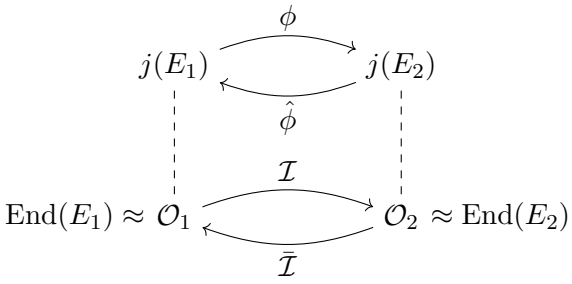
We consider supersingular curves E_1 and E_2 defined over $\mathbb{F}_{p^2}$. We also assume that the full ℓ^e-torsion is rational over $\mathbb{F}_{p^2}$. Finally, we consider additional information:

- End(E_1) is known: the corresponding maximal order $\mathcal{O}_1 = \mathbb{Z}\langle b_1, \ldots, b_4\rangle$ of $H_{-q,-p}$ is given, and the endomorphisms can be evaluated at points of E_1.
- There is a separable cyclic isogeny $\phi : E_1 \to E_2$ of degree ℓ^e. Note that we treat in the chapter only the case of cyclic isogenies, which is the case of interest of most of the current cryptographic applications. We assume that a generator of its kernel is given, i.e. $\ker(\phi) = \langle P \rangle$ with $\mathrm{ord}(P) = \ell^e$.

Endomorphism rings of E_1 and E_2 are closely related with the following formula (which is consequence of [Sil86, Theorem 6.2]):

$$\mathrm{End}(E_2) = \frac{1}{\deg(\phi)} \phi \circ \mathrm{End}(E_1) \circ \hat{\phi} \tag{6.2}$$

In this section, we aim to compute the maximal order $\mathcal{O}_2$ corresponding to $\mathrm{End}(E_2)$ through the isogeny ϕ. As isogenies connect $\bar{\mathbb{F}}_p$-isomorphism classes of supersingular curves, connecting ideals join the corresponding maximal orders. More precisely, there exists an integral $\mathcal{O}_1$-left ideal $\mathcal{I}$ (i.e. $\mathcal{I} \subset \mathcal{O}_1$) connecting $\mathcal{O}_1$ and $\mathcal{O}_2 = O_R(\mathcal{I})$.



In order to understand the relation between ϕ and $\mathcal{I}$, we introduce a subgroup of $E_1(\bar{\mathbb{F}}_p)$ related to $\mathcal{I}$ and ϕ.

Definition 6.23 (Ideal torsion subgroup)**.** *Let E be a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$ and $\mathcal{O}$ the maximal order corresponding to* $\mathrm{End}(E)$*. Let $\mathcal{I}$ be an integral $\mathcal{O}$-left ideal $\mathcal{I}$. Elements of $\mathcal{I}$ represent endomorphisms, and the $\mathcal{I}$ torsion subgroup $E[\mathcal{I}]$ is the set of points of $E(\bar{\mathbb{F}}_p)$ that cancel all endomorphisms of $\mathcal{I}$:*

$$E[\mathcal{I}] := \{P \in E(\bar{\mathbb{F}}_p), \alpha(P) = 0 \text{ for all } \alpha \in \mathcal{I}\}.$$

$E[\mathcal{I}]$ is a finite subgroup of $E(\bar{\mathbb{F}}_p)$. Using this subgroup, Voight brings in [Voi20] the explicit correspondence as follows:

- If $\mathcal{O}_1$ is a maximal order corresponding to $\mathrm{End}(E_1)$, and $\mathcal{I}$ is an integral $\mathcal{O}_1$-left ideal, the subgroup $E_1[\mathcal{I}]$ defines an ($\bar{\mathbb{F}}_p$-isomorphism class of) isogeny $\phi : E_1 \to E_2$ of kernel $E_1[\mathcal{I}]$. The codomain curve satisfies $\mathrm{End}(E_2) \sim O_R(\mathcal{I})$ so that $\mathcal{I}$ is a connecting ideal between $\mathrm{End}(E_1)$ and $\mathrm{End}(E_2)$. The isogeny computation is developed in Section 6.6.1.
- If ϕ is an isogeny between two supersingular curves E_1 and E_2 with $\mathrm{End}(E_1) \sim \mathcal{O}_1$, then there exists an integral $\mathcal{O}_1$-left ideal $\mathcal{I}$ such that $E_1[\mathcal{I}] = \ker(\phi)$, and $O_R(\mathcal{I}) \sim \mathrm{End}(E_2)$. The ideal computation is developed in Section 6.6.2.

We now investigate the explicit computation of the subgroup $E_1[\mathcal{I}]$ (from the knowledge of $\mathcal{I}$) and the ideal $\mathcal{I}$ (from the knowledge of the isogeny).

6.6.1 From ideal to isogeny

In this section, we consider that $\mathcal{O}_1 = \mathbb{Z}\langle b_1, \ldots, b_4\rangle$ is a maximal order of $H_{-q,-p}$ corresponding to the endomorphism ring of a supersingular curve E_1. We also assume that the endomorphisms that are represented by the quaternions b_i can be evaluated at points of the curve E_1 (see Example 6.18 for a possible choice for E_1). Given an integral $\mathcal{O}_1$-left ideal $\mathcal{I} = \mathbb{Z}\langle v_1, \ldots, v_4\rangle$ of norm $N(\mathcal{I}) = \ell^e$, we use Algorithm 6.2 in order to write $\mathcal{I} = \mathcal{O}_1 \cdot \ell^e + \mathcal{O}_1 \cdot \alpha$. The $\mathcal{I}$ torsion subgroup $E_1[\mathcal{I}]$ is generated by a point $P \in E_1[N(\mathcal{I})]$ so that we look for a point of order ℓ^e generating the subgroup. Recall that the ℓ^e-torsion subgroup of a curve is isomorphic to $\mathbb{Z}/\ell^e\mathbb{Z} \times \mathbb{Z}/\ell^e\mathbb{Z}$. From now on, we assume that A and A' form a basis of the ℓ^e-torsion of E_1: $E_1[\ell^e] = \langle A, A'\rangle$. In order to recover the point P, we need to solve the following equation where n and n' are the unknowns:

$$\alpha(nA + n'A') = n\alpha(A) + n'\alpha(A') = 0_{E_1}.$$

Note that the order of $\alpha(A)$ divides the order of A. Again, one can use the CRT before solving an equation as in Section 6.5. Finally, we use one of the algorithms of Section 3.2 in order to compute the isogeny of kernel $E_1[\mathcal{I}]$.

Algorithm 6.8: Isogeny(I)

Input. I a matrix representing an $\mathcal{O}$-left ideal $\mathcal{I}$ of norm ℓ^e.
Output. An isogeny from a curve E (where $\mathrm{End}(E) = \mathcal{O}$) with kernel $E[\mathcal{I}]$.

$A, A' \leftarrow$ a basis of $E[\ell^e]$	▷ Algorithm 6.5
Write $\mathcal{I} = \mathcal{O} \cdot \ell^e + \mathcal{O} \cdot \alpha$	▷ Algorithm 6.2
Compute $\alpha(A), \alpha(A')$.	
Find n and n' such that $\alpha(nA + n'A') = 0$	▷ Algorithm 6.7
return the isogeny of kernel $\langle [n]A + [n']A' \rangle$	▷ Algorithm 3.2

Complexity. Once a basis of $E[2\ell^e]$ is computed, Algorithm 6.8 is essentially linear algebra with matrices of $\mathbf{M}_4(\mathbb{Z}/\ell^e\mathbb{Z})$. Hence, the cost is dominated by the computation of the basis of $E[2\ell^e]$ which costs $O(\kappa \log_2(p))\mathbf{m}_{2\kappa}$ if we consider elliptic curves defined over $\mathbb{F}_{p^2}$ (see Algorithm 6.5).

Remark 6.24. In practice, we will be able to compute equivalent ideals (see Section 6.6.3) so that we will consider a small extension degree κ, and hence obtain an efficient algorithm with $O(\log_2(p))$ multiplications.

Example 6.25. We consider a curve E_1 as in Example 6.18 with $p = 2^{22}3^{15} - 1$, and $\mathcal{I} = \mathbb{Z}\langle \frac{\mathbf{1}+\mathbf{j}}{2}, \frac{\mathbf{i}+\mathbf{k}}{2}, 3\mathbf{j}, 3\mathbf{k} \rangle$. $\mathcal{I}$ is an ideal of norm 3 and can be written $\mathcal{I} = \mathcal{O}_1 \cdot 3 + \mathcal{O}_1 \cdot \alpha$ with $\alpha = \frac{29+42\mathbf{i}+293\mathbf{j}+282\mathbf{k}}{2}$. We look for the subgroup $E_1[\mathcal{I}]$ of $E_1(\bar{\mathbb{F}}_p)$, generated by a point of order 3. Writing $\mathbb{F}_{p^2} = \mathbb{F}_p(i) = \mathbb{F}_p[x]/(x^2+1)$ and choosing

$$x = 60172068522527, \qquad y_0 = 48359098373596, \qquad y_1 = 11824579652131,$$

the points $A = (xi, y_0 + iy_1)$ and $A' = \pi(A)$ form a basis of $E_1[3]$. We find the kernel point $nA + n'A'$ by solving the equation $\alpha(nA + n'A') = 0_{E_1}$ using Algorithm 6.7. More precisely, we evaluate α at A and A' and solve a linear system in $\mathbb{Z}/3\mathbb{Z}$:

$$\alpha(A) = A + A' = \alpha(A') \Longrightarrow (n, n') \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} = (0, 0).$$

$(2, 1)$ is a solution of the system and provides a generator of the subgroup $E_1[\mathcal{I}] = \langle 2A + A' \rangle$. The corresponding isogeny is defined with the rational maps

$$(x, y) \mapsto \left(\frac{x^3+23219016768x^2+62705669x+92876060160}{x^2+23219016768x+8957953}, y\frac{x^3+34828525152x^2+60183651151870x+34828514784}{x^3+34828525152x^2+26873859x+34828521696} \right)$$

and the codomain curve is $E_2 : y^2 = x^3 + 60183409287146x + 1950397166592$.

The latter example is reproducible with the file `example/ideal-to-isogeny.m` provided in the repository.

6.6.2 From isogeny to ideal

We consider as in Section 6.6.1 that $\mathcal{O}_1 = \mathbb{Z}\langle b_1, \ldots, b_4 \rangle$ is a maximal order of $H_{-q,-p}$ corresponding to the endomorphism ring of a supersingular curve E_1. We also assume that the b_i can be

evaluated at points of E_1. Suppose that $\phi : E_1 \to E_2$ is a separable isogeny of degree ℓ^e, and that $\ker(\phi) = \langle P \rangle$ with $\text{ord}(P) = \ell^e$. We want to find a basis of an ideal $\mathcal{I}$ such that $E[\mathcal{I}] = \ker(\phi)$ as described in Section 6.6. Remark that $\ell^e \in \mathcal{I}$ because $[\ell^e]P = 0$. By the *two-generator* decomposition of Section 6.3.2, we look for an ideal of the form $\mathcal{I} = \mathcal{O}_1\ell^e + \mathcal{O}_1\alpha$, where $\alpha \in \mathcal{O}$ and $\ker(\alpha) \supset \langle P \rangle$. Hence, the ideal norm is exactly the degree of the isogeny: $N(\mathcal{I}) = \ell^e$. The only unknown is the endomorphism $\alpha = \sum_{i=1}^{4} n_i b_i$ written in the basis of $\mathcal{O}_1$. We need to solve the following equation, where the n_i are the unknowns:

$$\sum_{i=1}^{4} n_i b_i(P) = 0.$$

Again, we solve this equation with Algorithm 6.7 together with the CRT and obtain $\alpha = \sum_{i=1}^{4} n_i b_i$. Finally, $\mathcal{I} = \mathcal{O}_1\ell^e + \mathcal{O}_1\alpha$. One obtains the matrix I of $\mathcal{I}$ by computing a 8×4 matrix and reducing to a Hermite Normal Form:

$$U \begin{pmatrix} b_i\ell^e \text{ for } 1 \le i \le 4 \\ \hline b_i\alpha \text{ for } 1 \le i \le 4 \end{pmatrix} = \begin{pmatrix} \diagdown \\ 0 \end{pmatrix}.$$

The four non-zero vectors of the upper triangular part lead to $\mathcal{I} = \mathbb{Z}\langle u_1, \ldots, u_4 \rangle$. We obtain the E_2 endomorphism ring by computing its right order $O_R(\mathcal{I})$ using intersection of lattices as described in Section 6.3.2.

Algorithm 6.9: Ideal(ϕ, $\mathcal{O}$)

Input. $\phi : E \to E'$ an isogeny of degree ℓ^e given by its kernel $\langle P \rangle$,
$\mathcal{O} = \text{End}(E)$ given with a basis $b_1, \ldots, b_4$ efficiently computable.
Output. An ideal $\mathcal{I}$ of norm ℓ^e connecting $\text{End}(E)$ and $\text{End}(E')$.

Compute $b_1(P), \ldots, b_4(P)$.
Find $(n_1, \ldots, n_4)$ such that $\sum_{i=1}^{4} n_i b_i(P) = 0_E$ ▷ Algorithm 6.7
$\alpha \leftarrow n_1 b_1 + \ldots + n_4 b_4$.
return $\mathcal{O}\ell^e + \mathcal{O}\alpha$.

As in the case of Algorithm 6.8, the cost of Algorithm 6.9 depends on the parameters, as it evaluates endomorphisms at points of the curve. Again, once a basis of $E[2\ell^e]$ is computed, linear algebra leads to an endomorphism α and the ideal $\mathcal{O} \cdot \ell^e + \mathcal{O} \cdot \alpha$. Finding a basis of $E[2\ell^e]$ is done in $O(\kappa \log_2(p))\mathbf{m}_\kappa$ with Algorithm 6.5. In the worst case, $kappa = O(\ell^{2e})$ but we will see in Section 6.6.3 that we can only deal with a small κ, bringing an efficient algorithm in practice.

We provide here an example where evaluating endomorphisms is efficient.

Example 6.26. We consider the same setting as in Example 6.25. Let $p = 2^{22}3^{15} - 1$ and $E_1 : y^2 = x^3 - x$. We consider the isogeny of kernel generated by $2A + A'$ obtained at the end of Example 6.25, and we aim to recover the ideal $\mathcal{I}$. The equation to solve is given by $\sum_{i=1}^{4} n_i b_i(2A + A') = 0$ which leads to

$$n_1(2A + A') + n_2(2A + 2A') = 0 \iff \begin{cases} 2n_1 + 2n_2 &= 0 \\ n_1 + 2n_2 &= 0 \end{cases}.$$

Modulo 3, it leads to $n_2 = n_1 = 0$ so that we choose $\alpha = b_3$. In order to get a basis of $\mathcal{I} = \mathcal{O}_1 \cdot 3 + \mathcal{O}_1 \cdot \alpha$, we compute the matrix of the $3b_i$ vertically joint with the $b_i\alpha$:

$$\begin{pmatrix} 3 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 3/2 & 0 & 3/2 & 0 \\ 0 & 3/2 & 0 & 3/2 \\ 1/2 & 0 & 1/2 & 0 \\ 0 & 1/2 & 0 & 1/2 \\ (1-p)/4 & 0 & 1/2 & 0 \\ 0 & (1-p)/4 & 0 & 1/2 \end{pmatrix}$$

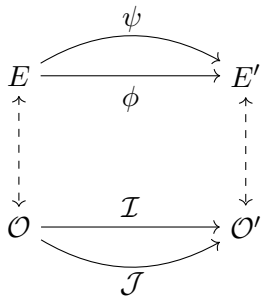
Reducing this matrix to a HNF, the four first non-zero rows lead to the ideal $\mathcal{I} = \mathbb{Z}\langle(\mathbf{1} + \mathbf{j})/2, (\mathbf{i} + \mathbf{k})/2, 3\mathbf{j}, 3\mathbf{k}\rangle$. It is indeed the ideal of the setting of Example 6.25, and its right order is $O_R(\mathcal{I}) = \langle(\mathbf{1} + \mathbf{j})/2, (\mathbf{i} + \mathbf{k})/6, \mathbf{j}, 3\mathbf{k}\rangle$.
The latter example is reproducible with the file `example/isogny-to-ideal.m` provided in the repository.

In practice, evaluating endomorphisms of $O_R(\mathcal{I})$ require division by $\deg(\phi) = N(\mathcal{I})$ which can be large in cryptographic applications. Instead, [KLPT14, GPS20] look for an equivalent ideal of norm coprime with the order of the point considered so that evaluating endomorphisms becomes efficient.

6.6.3 Computing equivalent ideals

Recall that the ideal $\mathcal{I}$ corresponding to an isogeny ϕ satisfies $N(\mathcal{I}) = \deg(\phi)$. Then, the right order $O_R(\mathcal{I})$ corresponds to $\mathrm{End}(E_2)$ and evaluating endomorphisms of E_2 requires computing a division by $N(\mathcal{I})$ which is often expensive. Example 6.19 requires looking at a degree 2 extension of $\mathbb{F}_{p^2}$ but in practice, we need to compute scalar divisions of points by large integers, therefore large extension degrees are needed.

Instead, we look for an ideal $\mathcal{J} \sim \mathcal{I}$ of norm $N(\mathcal{J})$ coprime to the factors of $p+1$ (recall that points of $E_2(\mathbb{F}_{p^2})$ have order a divisor of $p+1$). This way, the division of a point of $P \in E_2(\mathbb{F}_{p^2})$ by $N(\mathcal{J})$ is simply $Q = [\mathrm{inv}_{\bmod \mathrm{ord}(P)}(N(\mathcal{J}))]P$.



We provide in this section an algorithm described in [GPS20] for computing an equivalent ideal of powersmooth norm, given an ideal $\mathcal{I}$ for which the division point is expensive (i.e. an ideal of norm powersmooth, but not coprime to $p+1$). Finding an equivalent ideal $\mathcal{J} \sim \mathcal{I}$ is related to Proposition 6.27.

Proposition 6.27 (from [GPS20]). *Given an ideal $\mathcal{I}$ of norm $N(\mathcal{I})$ and an element $b \in H_{-q,-p}$ of norm $N(b)$, the equivalent ideal $\mathcal{J} = \mathcal{I}\bar{b}/N(\mathcal{I})$ has norm $N(\mathcal{J}) = N(b)/N(\mathcal{I})$.*

Galbraith, Petit, Silva and Ti proposed in [GPS20] an algorithm for finding an adequate element $b \in \mathcal{I}$ such that an ideal of the form $\mathcal{I}\bar{b}/N(\mathcal{I})$ has the desired properties. Finding a fractional ideal (i.e. $b \notin \mathcal{I}$) is easy but it does not make sense for the elliptic curve point of view: a fractional ideal is not included in $\mathcal{O}_2$ (Algorithm of Section 6.6.1 does not apply for fractional ideals). The algorithm of [GPS20] computes an integral ideal $\mathcal{J}$ and splits in three steps:
Step 1. Find $\delta \in \mathcal{I} = \mathbb{Z}\langle u_1, \dots, u_4\rangle$ such that the ideal $\mathcal{I}' := \mathcal{I}\bar{\delta}/N(\mathcal{I})$ has prime norm $N(\mathcal{I}') = N$.
Step 2a. Find $\beta_1 \in \mathcal{O}$ such that $N(\beta_1) = NS_1$ for a smooth integer S_1.
Step 2b. Find $\beta_2 \in \mathbb{Z}\mathbf{j} + \mathbb{Z}\mathbf{k}$ such that $\beta_1\beta_2 \in \mathcal{I}'$.
Step 2c. Find β_2' such that $\beta_2' = \lambda\beta_2 \bmod N\mathcal{O}$ and $N(\beta_2') = S_2$ for a smooth integer S_2.
Step 3. Set $\beta = \beta_1\beta_2'$ and output $\mathcal{J} := \mathcal{I}'\bar{\beta}/N$.
We precise how to algorithmically obtain these three steps.

Step 1. In order to compute an ideal of prime norm, Galbraith et al. fix a bound m and choose random integers $n_1, \dots, n_4$ between 1 and m such that $\delta = \sum_{i=1}^{4} n_i u_i$ has norm $N(\mathcal{I})$ times a prime integer N. In our notations, one computes the vector-matrix multiplication $(n_1, \dots, n_4) \times I$ until the norm of the corresponding quaternion is of the form $N \cdot N(\mathcal{I})$. Recall that I is the matrix whose rows are the u_i in the base $\mathbf{1}, \mathbf{i}, \mathbf{j}, \mathbf{k}$. Choosing $m = \lceil \log_2(p) \rceil$ and assuming heuristically that the numbers N generated behave like random numbers, Algorithm 6.10 produces a prime number N in $\tilde{O}(\sqrt{p})$.

Algorithm 6.10: PrimeNormEquivalentIdeal(I, m) from [GPS20]

Input. I a matrix representing an ideal $\mathcal{I}$ of an order $\mathcal{O}$,
m an integer.
Output. An ideal of prime norm N, equivalent to $\mathcal{I}$.

repeat
(increase m if needed)
$a \leftarrow \text{Random}(1, m)$
$b \leftarrow \text{Random}(1, m)$
$c \leftarrow \text{Random}(1, m)$
$d \leftarrow \text{Random}(1, m)$
$\delta \leftarrow (a, b, c, d) \times I$
until $N(\delta)/N(\mathcal{I})$ is prime and is a square mod N.
return $\mathcal{I} \cdot \bar{\delta}/N(\mathcal{I})$

The ideal $\mathcal{I}'$ can be represented with two generators. By Proposition 6.27, it has norm N and so we can write $\mathcal{I}' = \mathcal{O} \cdot N + \mathcal{O} \cdot \alpha$ for a given α, using Algorithm 6.2. In fact, any α satisfying $\gcd(N(\alpha), N^2) = N$ provides a two-generator representation, as N is prime. See [GPS20] for details.

The following step studies the quotient $\mathcal{O}/N\mathcal{O}$. As stated at the end of Section 6.3.2, we also require for simplicity that $-q$ is a square modulo N. In practice, we often choose $H_{-1,-p}$ as it is common that $p = 3 \bmod 4$.

Step 2a. The search of a quaternion β_1 corresponds to solving a norm equation of the form $a^2 + qb^2 + p(c^2 + qd^2) = NS_1$. The powersmooth number S_1 is chosen so that the latter equation is solvable. The authors of [GPS20] consider that S_1 needs to be larger than $p \log_2(p)$. Algorithm 6.11

finds a quaternion of $\mathbb{Z}\langle \mathbf{1}, \mathbf{i}, \mathbf{j}, \mathbf{k}\rangle$ by solving a norm equation: it uniformly chooses two coordinates of the quaternion, and then use a Cornacchia algorithm in order to get the two others (see [GPS20, page 163]). We obtain β_1 using this algorithm with the input $n = N(\mathcal{I})S_1$, where $S_1 > p\log_2(p)$ is a powersmooth integer.

Algorithm 6.11: FindQuaternionOfNorm(n) from [GPS20]

Input. n an integer.
Output. A quaternion $a\mathbf{1} + b\mathbf{i} + c\mathbf{j} + d\mathbf{k} \in H_{-1,-p}$ of norm n.

$m \leftarrow \lfloor \sqrt{n/2p} \rfloor$
repeat
  (increase m if needed)
  $c \leftarrow \text{Random}(1, m)$
  $d \leftarrow \text{Random}(1, m)$
until the equation $x^2 + y^2 = n - p(c^2 + d^2)$ has a solution.
$a, b \leftarrow$ a solution of $x^2 + y^2 = n - p(c^2 + d^2)$.
return (a, b, c, d)

Step 2b. The next step finds $\beta_2 = C\mathbf{j} + D\mathbf{k}$ satsfying the condition $\beta_1\beta_2 \in \mathcal{I}'$, which is equivalent to $\beta_1\beta_2 \in \mathcal{O}\alpha \bmod N\mathcal{O}$. An endomorphism of $\mathcal{O}$ seen in the quotient $\mathcal{O}/N\mathcal{O}$ is described by a matrix of $\mathbf{M}_2(\mathbb{Z}/N\mathbb{Z})$ (see Algorithm 6.3). From this representation of $\mathcal{O}/N\mathcal{O}$, Algorithm 6.12 computes linear algebra modulo N in order to compute the element β_2 modulo $N\mathcal{O}$. More precisely, it first writes $\alpha \bmod N\mathcal{O}$ using Algorithm 6.3:

$$\alpha \bmod N\mathcal{O} = \begin{pmatrix} ux & uy \\ vx & vy \end{pmatrix}.$$

Then, we compute the matrix $(m_{i,j})$ (resp. $(m'_{i,j})$) corresponding to $\beta_1\mathbf{j}$ (resp. $\beta_1\mathbf{k}$) in $\mathcal{O}/N\mathcal{O}$ in the same way. From Proposition 6.15, there exists$(u' : v') \in \mathbb{P}^1(\mathbb{Z}/N\mathbb{Z})$ such that $(x : y)$ and $(u' : v')$ map to $\beta_1\beta_2 \bmod N\mathcal{O}$ by the bijection given on page 105. Thus, β_2 is obtained by solving the linear system modulo N:

$$(C, D, u', v') \begin{pmatrix} m_{11} & m_{12} & m_{21} & m_{22} \\ m'_{11} & m'_{12} & m'_{21} & m'_{22} \\ -x & -y & 0 & 0 \\ 0 & 0 & -x & -y \end{pmatrix} = (0, 0, 0, 0). \tag{6.3}$$

Algorithm 6.12: ApproximationInIdeal(β_1, $\mathcal{I}$)

Input. $\beta_1 \in \mathcal{O}$,
  $\mathcal{I}$ an $\mathcal{O}$-left ideal of prime norm N.
Output. An element $\beta_2 \in \mathbb{Z}\mathbf{j} + \mathbb{Z}\mathbf{k}$ such that $\beta_1\beta_2 \in \mathcal{I}$.

Write $\mathcal{I} = \mathcal{O}N + \mathcal{O}\alpha$ ▷ Algorithm 6.2
$M_{\alpha \bmod N} \leftarrow \alpha \bmod N\mathcal{O}$ ▷ Algorithm 6.3
$M_{\beta_1\mathbf{j} \bmod N} \leftarrow \beta_1\mathbf{j} \bmod N\mathcal{O}$ ▷ Algorithm 6.3
$M_{\beta_1\mathbf{k} \bmod N} \leftarrow \beta_1\mathbf{k} \bmod N\mathcal{O}$ ▷ Algorithm 6.3
Find $((x : y), (u : v)) \in \mathbb{P}^1(\mathbb{Z}/N\mathbb{Z}) \times \mathbb{P}^1(\mathbb{Z}/N\mathbb{Z})$ corresponding to $M_{\alpha \bmod N}$ ▷ Propositon 6.15
$(C, D, (u' : v')) \leftarrow$ a solution of Equation (6.3).
return $(0, 0, C, D)$

Step 2c. From β_2, the next step computes an element $\beta_2' \in \mathcal{I}'$ of powersmooth norm S_2 such that $\beta_2' = \lambda\beta_2 \bmod N\mathcal{O}$ for a scalar $\lambda \in \mathbb{Z}$. Galbraith et al. estimate that S_2 needs to be larger than $p^3 \log_2(p)$ in order to find a solution. We provide Algorithm 6.13 for details.

Algorithm 6.13: SecondQuaternionApproximation(β_2, I, S_2)

Input. A quaternion $\beta_2 \in \mathcal{O}$ of the form $C\mathbf{j} + D\mathbf{k}$,
I a matrix representing an ideal $\mathcal{I}$ of prime norm (output from Algorithm 6.10),
S_2 a powersmooth integer.
Output. An element $\beta_2' \in \mathcal{I}'$ with norm S_2, such that $\beta_2' = \lambda\beta_2 \bmod N\mathcal{O}$ for a scalar $\lambda \in \mathbb{Z}$.

$(0, 0, C, D) \leftarrow \beta_2$.
$N \leftarrow N(\mathcal{I})$.
$r \leftarrow$ the first odd prime such that $S_2 r/(p(C^2 + D^2))$ is a square mod N
$S_2 \leftarrow S_2 r$.
$\lambda \leftarrow$ a square root of $S_2/(p(C^2 + D^2)) \bmod N$.
repeat
  $c \leftarrow \text{Random}(1, N)$
  $d \leftarrow ((S_2 - p\lambda^2(C^2 + D^2) - 2\lambda NCcp)/(2\lambda Dp))/N$
until the equation $x^2 + y^2 = (S_2 - p(\lambda C + cN)^2 + (\lambda D + dN)^2)/N^2$ has a solution.
$(a, b) \leftarrow$ a solution of $x^2 + y^2 = (S_2 - p(\lambda C + cN)^2 + (\lambda D + dN)^2)/N^2$.
return $(Na, Nb, \lambda C + cN, \lambda D + dN)$

Step 3. The element $\beta := \beta_1\beta_2'$ satisfies $\beta = \beta_1\lambda\beta_2 = \lambda\alpha \bmod N\mathcal{O}$. It means that $\beta = \lambda\alpha + Nx$ for an element $x \in \mathcal{O}$, and so $\beta \in \mathcal{I}' = \mathcal{O} \cdot N + \mathcal{O} \cdot \alpha$. Moreover, $N(\beta) = N(\beta_1)N(\beta_2) = NS_1S_2$ and so the ideal $\mathcal{J} := \mathcal{I}'\bar{\beta}/N$ has powersmooth norm: $N(\mathcal{J}) = S_1S_2$.

Summarized algorithm. Algorithm 6.14 summarizes the different steps of the previous algorithms.

Algorithm 6.14: EquivalentIdeal(I)

Input. I a matrix representing an ideal $\mathcal{I}$ of an order $\mathcal{O}$.
Output. An equivalent ideal $\mathcal{J} \sim \mathcal{I}$.

$\mathcal{I}' \leftarrow$ an ideal equivalent to $\mathcal{I}$ of prime norm N using $m = \lceil \log_2(p) \rceil$ ▷ Algorithm 6.10
Write $\mathcal{I}' = \mathcal{O} \cdot N + \mathcal{O} \cdot \alpha$ ▷ Algorithm 6.2
$S_1 \leftarrow$ a powersmooth integer larger than $\lceil p \log_2(p) \rceil$.
$S_2 \leftarrow$ a powersmooth integer larger than $\lceil p^3 \log_2(p) \rceil$.
Compute β_1 of norm NS_1. ▷ Algorithm 6.11
Compute β_2 from β_1 and $\mathcal{I}'$ ▷ Algorithm 6.12
Compute β_2' from β_2, $\mathcal{I}'$ and S_2 ▷ Algorithm 6.13
$\beta \leftarrow \beta_1\beta_2'$.
return $\mathcal{I}' \cdot \bar{\beta}/N$

Practical complexity. As stated in [GPS20, page 163], the asymptotic complexity of Algorithm 6.14 is $\tilde{O}(\log_2(p)^3)$ bit operations, corresponding to the $\log_2(p)$ tests of (pseudo-)primality for checking that the Diophantine equation has a solution (see [GPS20, page 163]). Indeed, most

of the time is spent in these primality tests while running our implementation. Galbraith et al. obtain that the target ideal $\mathcal{J}$ has expected norm $\tilde{O}(3.5\log_2(p))$. We provide Example 6.28 in order to show that the constant in the $\tilde{O}$ can be significant.

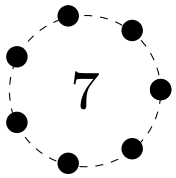
Example 6.28. We end this section with an example of equivalent ideal computation, leading to two isogenies of different degrees between two curves. Let $p = 2^{89} - 1$ and $E : y^2 = x^3 - x$ be the supersingular elliptic curve defined over $\mathbb{F}_{p^2} = \mathbb{F}_p(i) = \mathbb{F}_p[x]/(x^2+1)$. The full 2^{89}-torsion is rational over $\mathbb{F}_{p^2}$. As we have seen in Example 6.18, $\mathrm{End}(E)$ is isomorphic to the maximal order $\mathcal{O} = \mathbb{Z}\langle \mathbf{1}, \mathbf{i}, \frac{\mathbf{1}+\mathbf{j}}{2}, \frac{\mathbf{i}+\mathbf{k}}{2}\rangle \subset H_{-1,-p}$. From an isogeny of degree 2^{87}, Algorithm 6.9 computes an ideal of norm 2^{87}. Then, Algorithm 6.10 computes $\mathcal{I}' = \mathcal{I}\delta$ of 54-bit prime norm (28% larger than $\log_2(p)/2$). Algorithm 6.11 outputs a quaternion β_1 of 150-bit norm (72% larger than $\log_2(p)$), and Algorithm 6.13 produces a quaternion β_2' of 326-bit norm (22% larger than $3\log_2(p)$). Finally, the ideal $\mathcal{J}$ has norm a 423-bit powersmooth integer. This is 36% larger than the expected $3.5\log_2(p)$ size, probably due the asymptotic constant factor in the $\tilde{O}$. The right order of $\mathcal{J}$ has a basis with elements of $\mathbb{Z}[\frac{1}{2\cdot 3^{267}}]\langle \mathbf{1}, \mathbf{i}, \mathbf{j}, \mathbf{k}\rangle$. Thus, we obtain another representation for endomorphisms of $\mathcal{O}'$ using the fact that $O_R(J) = \omega^{-1}O_R(I)\omega$. Finally, evaluating an endomorphism at a point P can be computed efficiently with $O_R(I)$ if $\gcd(\mathrm{ord}(P), 2^{87}) = 1$, and $O_R(J)$ otherwise.

The latter example is reproducible with the file `example/equivalent-ideal.m` in the repository.

The equivalent isogeny. The target ideal output in Algorithm 6.14 is also integral so that it makes sense to consider the corresponding isogeny, of degree $N(\mathcal{J})$. The isogeny is often expensive to compute directly: computing the isogeny kernel requires looking at $E[N(\mathcal{J})]$ which is defined over a large extension field. Instead, one can split the ideal J into prime norm ideals so that it corresponds to prime degree isogenies. Writing $J = \mathcal{O}N(\mathcal{J}) + \mathcal{O}\alpha$ using Algorithm 6.2, we set $\tilde{J} = \mathcal{O}\ell + \mathcal{O}(\alpha \bmod \ell\mathcal{O})$, which corresponds to the first degree ℓ isogeny, and then iterate on $J\tilde{J}^{-1}$. Thus, we can compute an equivalent isogeny given the knowledge of the s-torsion, for the primes s dividing the powersmooth integer S. We notice that the computation of the equivalent isogeny is not provided in our implementation.

6.7 Conclusion

In this chapter, we present a concrete algorithmic presentation of the work of [GPS20, KLPT14]. The complexity claimed in the latter works is heuristic, but polynomial in $\log_2(p)$. It involves manipulating elements of extension fields that may not be practically reachable in the context of cryptographic applications. We provided here a concrete implementation in order to estimate a practical complexity. In the next chapter, we will reuse these algorithms in order to compute a shortcut in the graph of supersingular curves. This implementation has not been optimized in the sense that it is used as an illustration of the attack we describe in the following chapter. A work would be need in order to use the code for constructive cryptography such as the signature of [GPS20]. Moreover, we did not investigate computations with other initial maximal orders (we always assume that the initial curve is the curve of j-invariant 1728). Finally, we note that another work from Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit and Benjamin Wesolowski will be presented at Asiacrypt 2020 and provides an efficient implementation of equivalent ideal computation together with a generalization of the [GPS20] signature.



Verifiable delay functions from isogenies and pairings

A Verifiable Delay Function (VDF), first formalized in 2018 by Boneh, Bonneau, Bünz and Fisch [BBBF18], is a function $f : X \to Y$ that takes a prescribed wall-clock time to evaluate, independently of the parallelism of the architecture employed, and such that its output can be verified efficiently. In a nutshell, it is required that anyone can evaluate f in T sequential steps, but no less, even with a large number of processors; on top of that, given an input x and an output y, anyone must be able to verify that $y = f(x)$ in a short amount of time, say in polylog(T). In this chapter, we recall what is a VDF and its applications. Then, we go through some of the proposed solutions and describe a new framework for VDFs. This leads to two instantiations of verifiable delay functions using isogenies of supersingular elliptic curves (studied in Chapter 3) and bilinear pairings presented in Chapter 4. Section 7.5 gives security proofs and reviews the available attacks against our proposals. Finally Section 7.6 provides an implementation of our VDFs, and related measurements. This chapter corresponds to a joint work with Luca De Feo, Christophe Petit and Antonio Sanso [DMPS19], presented at Asiacrypt 2019.

Summary

7.1 Definition and applications

7.1.1 Definition

We recall here the formal definition of a Verifiable Delay Function, following [BBBF18]. A VDF consists of three algorithms:

1. $\mathsf{Setup}(\lambda, T) \to (\mathsf{ek}, \mathsf{vk})$: is a procedure that takes a security parameter λ, a delay parameter T, and outputs public parameters consisting of an evaluation key ek and a verification key vk.

2. $\mathsf{Eval}(\mathsf{ek}, s) \to (a, \pi)$: is a procedure to evaluate the function on input s. It produces the output a from s, and a (possibly empty) proof π. This procedure is meant to be infeasible in time less than T.

3. $\mathsf{Verify}(\mathsf{vk}, s, a, \pi) \to \{\mathsf{true}, \mathsf{false}\}$: is a procedure to verify that a is indeed the correct output for s, with the help of the proof π.

A VDF shall satisfy three security properties: *Correcteness*, stating that a honest evaluator always passes verification, *Soundness*, stating that a lying evaluator never passes verification, and *Sequentiality*, stating that it is impossible to correctly evaluate the VDF in time less than $T - o(T)$, even when using $\text{poly}(T)$ parallel processors. We will give formal security definitions in Section 7.5, but see also [BBBF18].

According to [BBBF18], the Setup routine should run in time $\text{poly}(\lambda)$; here we slightly relax this constraint and allow it to run in $\text{poly}(T, \lambda)$. Eval must be doable in time T; Verify in time $\text{poly}(\lambda)$. A VDF is said to be optimal when T is allowed to be in $o(2^\lambda)$ without harming security; note that it does not make sense to have $T \in O(2^\lambda)$, since in that case it is cheaper to break soundness than to run Eval.

7.1.2 Applications

We highlight a few applications of VDFs:

- Constructing a trustworthy *randomness beacon*, like the one introduced by Rabin in [Rab83], where a public service produces a continuous stream of guaranteed unbiased randomness. The classic approach consisting in extracting randomness from entropy pool sources, such as stock prices or proof-of-work blockchains *à la* Bitcoin, has been shown to be manipulable by active attackers [PW18]. For example, while the price of a particular stock may seem unpredictable to a passive observer, a powerful trader can influence the market trend, making the random output biased. Here is where VDFs are useful: if the beacon is calculated by applying a VDF with a long enough delay to the entropy source, the malicious trader would not have the time to try to "adjust" the market at his own advantage.

  The other common solution based on the "commit-and-reveal" paradigm with *multiparty randomness* has also been shown to have flaws. Indeed a malicious party with the intention of manipulating the output might refuse to reveal his commitment after seeing the other opened commitments, forcing to restart the protocol. This can be mitigated by threshold techniques, as shown in [SJK+17], or by replacing commitments with VDFs, as shown by Lenstra and Wesolowski [LW15].

- Reducing energy consumption. VDFs may be used to reduce the energy consumption of blockchains based on proofs-of-work. An elegant idea by Cohen [Coh17] combines proofs

of space with VDFs in order to achieve a proof of Space and time. In a nutshell, in order to write a new block of the blockchain, a miner needs to provide a proof of space and the output of a verifiable delay function. More precisely, each block B_i is composed of a proof of space σ_i and a proof of time τ_i, which is actually the output of a VDF. In order to *win* the block B_i, one needs to compute the VDF first, and the delay parameter depends on the storage of the miner:

1. A challenge c_i for the block B_i is provided in order to determine the miner delay (VDF) parameter. This challenge is the hash of the previous proof of time τ_{i-1}.
2. The miner finds the *closest* proof of space to c_i. We denote Δ to be the difference between his closest proof and c_i.
3. The miner computes a VDF whose delay parameter is proportional to Δ.

For a description of other applications of VDFs, such as *proof of replication* or *computational timestamping*, we refer to [BBBF18].

7.2 Existing constructions

We present constructions of functions that satisfy some of the criteria to be a verifiable delay function.

7.2.1 Chaining hash functions

An example of a delay function lacking efficient verification is a chained one-way function:

$$s \to H(s) \to H(H(s)) \to \cdots \to H^{(T)}(s) = a.$$

This clearly takes T steps to evaluate, even on a parallel computer, however the only feasible way to verify the output is to re-evaluate the function. Two related known cryptographic primitives are the time-lock puzzles defined by Rivest, Shamir, and Wagner in [RSW96] and proofs of sequential work [MMV13, CP18]. The problem with the former is that it is not publicly verifiable while the latter is not a function (i.e., it does not have a unique output).

So far, few constructions meet the requirements of a VDF; we summarize them below.

7.2.2 Modular square roots

One of the earlier examples of a VDF can be found in the 1992 paper by Dwork and Naor [DN93]. The underlying idea is rather simple: given a prime number p such that $p = 3 \bmod 4$, a (canonical) square root $a = \sqrt{s} \bmod p$ can be computed using the formula $a = s^{\frac{p+1}{4}}$. This requires about $\log(p)$ sequential squaring operations, i.e $\tilde{O}(\log(p)^2)$ binary operations. On the other hand, verifying correctness only requires to check that $a^2 = s$, costing $\tilde{O}(\log(p))$ binary operations. Using a delay parameter $T = \log(p)^2$, there is a gap between the evaluation and the verification, but this simple approach has two issues: first, the gap is only polynomial in the delay parameter (the verification costs $\tilde{O}(\sqrt{T})$ binary operations), secondly, due to the possibility of parallelizing field multiplications, this gap vanishes asymptotically if the evaluator is provided with large amounts of parallelism (see also Table 7.1). We refer to [BS07] for details on the asymptotic complexity using $n^{O(1)}$ processors, leading to $\tilde{O}(\sqrt{T})$ binary operations for a parallel evaluation. Lenstra and Wesolowski introduced with Sloth [LW15] the possibility of chaining square root operations. The problem with this construction, though, is that it does not achieve asymptotically efficient verification.

7.2.3 Time-lock puzzles

Time-lock puzzles were introduced by Rivest, Shamir, and Wagner [RSW96] to provide encryption that can only be decrypted after a given time T. They use a classical RSA modulus $N = pq$; the encryption key is then $a = s^{2^T} \bmod N$ for some starting value s. Now, it is clear that any party knowing $\varphi(N)$ can compute the value of a quickly (they can reduce the exponent $e = 2^T \bmod \varphi(N)$). But for everyone else the value of a is obtained by computing T sequential squaring operations.

The main reason why this construction cannot be classified as a VDF is that there is not an efficient way to perform public verification without giving away the factorization of N. This issue has recently been solved, independently, by Pietrzak and Wesolowski. We briefly present their constructions next; for a more in-depth survey, see [BBF18].

7.2.4 Wesolowski's VDF

In 2018, Wesolowski presented a VDF based on groups of unknown order [Wes19]. His work leverages the time lock puzzle described above, introducing a way to publicly verify the output $a = s^{2^T}$. He defines an interactive protocol where, after seeing the output a, the verifier sends to the prover a random prime $\ell < B$, where B is some small bound. The prover replies with the value $b = s^{\lfloor 2^T/\ell \rfloor}$; the verifier then checks that $a = b^\ell s^r$, where $r = 2^T \bmod \ell$. Because the verifier only uses public randomness, this protocol can be made non-interactive using the Fiat–Shamir heuristic. Wesolowski's proposal shines for the shortness of the proof (only one group element) and the speed of the verification (only two group exponentiations).

Wesolowski suggests two ways of instantiating groups of unknown order. The first one is using RSA groups $(\mathbb{Z}/N\mathbb{Z})^*$, like in Section 7.2.3. In order to get a secret RSA integer N (so that no one knows the factorization of N), a trusted third party is needed to produce the modulus N. The second is using class groups of imaginary quadratic number fields [Cox97, page 100]. While the former instantiation is better studied in public key cryptography, the second has the advantage of not requiring a trusted setup.

7.2.5 Pietrzak's VDF

Concurrently with Wesolowski, Pietrzak [Pie18] introduced another protocol to verify Rivest–Shamir–Wagner time-lock puzzles. Pietrzak's verification procedure is an interactive recursive protocol, where the prover outputs a proof π consisting of $O(\log(T))$ group elements, and the verifier needs about $O(\log(T))$ time to do the verification. The main advantage of his construction is that the prover only needs about $O(\sqrt{T})$ group multiplications to build π. Pietrzak presents his protocol using RSA groups, but class groups like in Wesolowski's VDF can also be used (although this affects slightly the computational assumptions needed for soundness).

7.2.6 Univariate permutation polynomials

Boneh, Bonneau, Bünz and Fisch explored in their seminal paper [BBBF18] an approach based on permutation polynomials over finite fields $\mathbb{F}_p$. In full generality, their proposal is a weaker form of VDF, where a certain amount of parallelism is needed to give an advantage to the evaluator (see [BBBF18, Definition 5]). The gist of their approach is that, given a permutation polynomial of degree T, inverting such polynomial implies computing polynomial GCDs. This operation takes $O(\log(p))$ multiplications of dense polynomials of degree $O(T)$, and it is conjectured that it cannot be done in less than T steps on at most $O(T^2)$ processors (see [BBBF18, Assumption 2]).

VDF	Sequential Eval	Parallel Eval	Verify	Setup	Proof size
Modular square root	T	$T^{1/2}$	$T^{1/2}$	T^3	—
Univariate permutation polynomials[3]	T^2	$> T - o(T)$	$\log(T)$	$\log(T)$	—
Wesolowski's VDF	$(1 + \frac{2}{\log(T)})T$	$(1 + \frac{2}{s\log(T)})T$	λ^4	λ^3	λ^3
Pietrzak's VDF	$(1 + \frac{2}{\sqrt{T}})T$	$(1 + \frac{2}{s\sqrt{T}})T$	$\log(T)$	λ^3	$\log(T)$
Isogeny and pairing VDF	T	T	λ^4	$T\lambda^3$	—
Isogeny and pairing VDF (optimized)	T	T	λ^4	$T\log(\lambda)$	—

Table 7.1: Asymptotic VDF comparison: T represents the delay factor, λ the security parameter, s the number of processors. For simplicity, we assume that T is super-polynomial in λ. All times are to be understood up to a (global across a line) constant factor.

On the other hand, any such polynomial can be evaluated, and thus verified, using $O(\log(T))$ operations on $O(T)$ processors, which is exponentially smaller. Moreover, there exists a family of permutation polynomials, due to Guralnick and Müller [GM97], that can be evaluated in $O(\log(T))$ operations without parallelism, and it is conjectured in [BBBF18] that the derived VDF is secure.

The drawbacks of this construction are that the parallelism of the evaluator needs to be polynomially bounded in T, and that it is based on assumptions that have been seldom studied in a cryptographic setting.

For completeness we need to mention that a theoretical, albeit impractical, VDF can be constructed using Incrementally Verifiable SNARKs. Again, we refer to [BBBF18] for a deeper analysis of the topic.

We compare the asymptotic performance of the VDFs above and of our proposal in Table 7.1. Outside of modular square roots, all VDFs constructions meet the requirements of an optimal VDF, however each has its qualitative strengths and weaknesses: permutation polynomials require to bound the parallelism of the evaluator, and are based on little studied assumptions; VDFs derived from time-lock puzzles are interactive, have no perfect soundness, and may or may not require a trusted setup; ours need a trusted setup, and require an effort to validate public parameters comparable to evaluating the VDF.

7.3 A new VDF construction framework

We start by describing a framework for defining VDFs inspired by the BLS signature scheme based on pairing groups [BLS04]. We briefly presented the BLS signature in Chapter 4 and recall it here with few more details:

- The BLS signature uses a pairing friendly elliptic curve E defined over $\mathbb{F}_p$, with a non-degenerate bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{F}_{p^k}$, where $\mathbb{G}_1, \mathbb{G}_2$ are distinct subgroups of prime order r of the curve E, and k is the embedding degree of the curve with respect to r (see Definition 2.13).
- The secret key is a scalar $s < r$, and the public key is a pair of points $P, [s]P \in \mathbb{G}_1$.

[3] According to [BBBF18, § 5.1], one must limit the evaluator to $O(T^2)$ parallel processors for the bound on parallel Eval to hold. VDFs based on permutation polynomials can be evaluated in time $O\big(\log^2(T)\big)$ using $O(T^{3.8})$ parallel processors.

- To sign a message m, the signer computes a hash $Q = H(m) \in \mathbb{G}_2$, and gives back the signature $[s]Q$.
- The verifier then checks that $e(P, [s]Q) = e([s]P, Q)$.

The BLS signature is also by design a Verifiable Random Function $f_s : \mathbb{G}_2 \to \mathbb{G}_2$, where only the owner of the trapdoor s can evaluate f_s, while anyone can verify the result [MRV99]; however, it is not a VDF because both evaluation and verification are in polylog(r). Our generalization, instead, has efficient instantiations based on isogeny graphs of supersingular elliptic curves, where the evaluation can be made exponentially slower than the verification. If the trapdoor is kept secret, one obtains a signature/identification protocol based on walks in isogeny graphs; if the trapdoor is made public, one obtains a VDF. Note that this is different from a trapdoor VDF, as defined by Wesolowski [Wes19], where the trapdoor is used to efficiently compute the evaluation. We will present our instantiations in Section 7.4.

We use notations of Chapter 4 and denote $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}'_1, \mathbb{G}'_2$ and $\mathbb{G}$ to be groups of exponent r as in Section 4.1.1. Let also $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}$ and $e' : \mathbb{G}'_1 \times \mathbb{G}'_2 \to \mathbb{G}$ be non degenerate bilinear pairings. Furthermore, we assume that there is a pair of bijections $\phi : \mathbb{G}_1 \to \mathbb{G}'_1$ and $\hat{\phi} : \mathbb{G}'_2 \to \mathbb{G}_2$ that satisfy the following diagram,

$$\begin{array}{ccc} \mathbb{G}_1 \times \mathbb{G}'_2 & \xrightarrow{\phi \times 1} & \mathbb{G}'_1 \times \mathbb{G}'_2 \\ {\scriptstyle 1 \times \hat{\phi}} \downarrow & & \downarrow {\scriptstyle e'} \\ \mathbb{G}_1 \times \mathbb{G}_2 & \xrightarrow[e]{} & \mathbb{G} \end{array}$$

Note that the diagram implies ϕ and $\hat{\phi}$ are group isomorphisms.

We shall assume that the pairings e, e' can be evaluated in time polylog(N), whereas both ϕ and $\hat{\phi}$ can be evaluated in sequential time T, where T is some parameter independent from r (but still in $o(r)$).

Let P be any generator of $\mathbb{G}_1$, the public parameters of our system are going to be $(r, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}'_1, \mathbb{G}'_2, \mathbb{G}, e, e', P, \phi(P))$. In the following, we consider subgroups $\mathbb{G}_i$ (resp. $\mathbb{G}'_i$) of a pairing friendly elliptic curve E (resp. E') as in Section 2.4. Thus, P is a point of the curve E generating the cyclic group $\mathbb{G}_1$, and e and e' are pairings such as in Sections 4.1.3, 4.1.4 and 4.1.5. For efficiency considerations, we consider here the Tate pairing in our implementation (see Section 7.6.2). Finally, the group $\mathbb{G}$ is a multiplicative subgroup of $\mathbb{F}^*_{p^k}$ of order r as in Chapters 4 and 5. From this setup, we derive two primitives:

An identification protocol. The maps ϕ and $\hat{\phi}$ are the trapdoor. The verifier gives an element $Q \in \mathbb{G}'_2$ to the prover, the proof is the element $\hat{\phi}(Q)$. Then, the verifier checks that

$$e(P, \hat{\phi}(Q)) = e'(\phi(P), Q).$$

It should be apparent that BLS signatures [BLS04] correspond to the special case where $\mathbb{G}_1 = \mathbb{G}'_1$ and $\mathbb{G}_2 = \mathbb{G}'_2$ are orthogonal groups with respect to the pairing $e = e'$, and $\phi = \hat{\phi} = [s]$ is the multiplication endomorphism by a secret scalar s. This generalization is closely related to several patents [JMVB09, JV09, BCL12]. More precisely, the latter abstract scheme already appears in a patent by Broker, Charles and Lauter [BCL12]. We will construct our VDFs using a similar structure in Section 7.3, but the design and the implementation are different, and likely

more efficient. We shall see in Section 7.5 that our instantiation presents the minor advantage over BLS signatures of being partially resistant to quantum attacks.

More recently, Koshiba and Takashima [KT16, KT19] have provided a framework and security definitions for some cryptographic protocols involving pairings and isogenies, called isogenous pairing groups. They also present key-policy attribute-based encryption schemes based on their framework.

Our new VDF construction does not fit within any of the previous frameworks: while the isogeny is secret there, here it is public. Moreover the isogeny involved in our construction has very large degrees to achieve the delay property; using isogenies of such degree would make any of the previous protocols unnecessarily slow. Security properties required for VDFs differ significantly in nature from traditional cryptographic protocols, and none of the computational assumptions previously used in isogeny-based cryptography, including those in [KT16], is relevant to our construction.

A verifiable delay function. The maps ϕ and $\hat{\phi}$ are also part of the public parameters. The VDF is the map $\hat{\phi}$, Eval simply amounts to evaluating it at points $Q \in \mathbb{G}_2'$. To verify the output, one checks that

$$e(P, \hat{\phi}(Q)) = e'(\phi(P), Q).$$

It should be clear that, because the map $R \mapsto e(P, R)$ is an isomorphism, verification succeeds if and only if the output is correct; this will be used to prove correctness and soundness in Section 7.5. By hypothesis, Eval takes T sequential steps, while the pairings can be evaluated in time $\mathrm{polylog}(r)$.

7.4 Two instantiations with supersingular elliptic curves

We now give two instantiations of the VDF described in Section 7.3, using supersingular elliptic curves for the pairing groups, and isogenies of prime power degree for the maps $\phi, \hat{\phi}$. We will see in Section 7.5 that the choice of the curves severely affects the security of the protocol, however we ignore this issue for the moment.

7.4.1 VDF from supersingular curves over $\mathbb{F}_p$

Our first construction uses supersingular curves defined over a prime field $\mathbb{F}_p$. It shares similarities with the key exchange protocol CSIDH [CLM+18] of Section 3.3.2 and with the VDF based on class groups of imaginary quadratic fields by Wesolowski [Wes19].

Let p be a prime such that $p+1$ contains a large prime factor r. Let ℓ be one of:

- $\ell = 2$, only if $p = 7 \bmod 8$, or
- a small prime such that $\left(\frac{-p}{\ell}\right) = 1$.

Let E be a supersingular elliptic curve defined over $\mathbb{F}_p$, and denote by e the Tate pairing on $E[r]$. When $\ell = 2$ we shall add the requirement that $E[2] \subset E(\mathbb{F}_p)$, implying that E is on the surface. By construction $\#E(\mathbb{F}_p) = p+1$, and $E(\mathbb{F}_p)$ contains exactly one cyclic subgroup of order r, that we shall use as $\mathbb{G}_1 = E(\mathbb{F}_p)[r]$. Recall that in large characteristic, supersingular curves defined over $\mathbb{F}_p$ have embedding degree $k = 2$. Hence, the full r-torsion of E is defined over $\mathbb{F}_{p^2}$.

$\mathsf{Setup}(\lambda, T)$

1. Choose primes r, p with the properties above, according to the security parameter λ;
2. Select a supersingular curve E defined over $\mathbb{F}_p$;
3. Choose a direction on the horizontal ℓ-isogeny graph, and compute a cyclic isogeny $\phi : E \to E'$ of degree ℓ^T, and its dual $\hat{\phi}$;
4. Choose a generator P of $\mathbb{G}_1 = \tau_2^{-1}(E^t(\mathbb{F}_p)[r])$, and compute $\phi(P)$;
5. Output $(\mathsf{ek}, \mathsf{vk}) = \big(\hat{\phi}, (E, E', P, \phi(P))\big)$.

$\mathsf{Eval}(\hat{\phi}, Q \in \mathbb{G}_2')$

1. Compute and output $\hat{\phi}(Q)$.

$\mathsf{Verify}(E, E', P, Q, \phi(P), \hat{\phi}(Q))$

1. Verify that $\hat{\phi}(Q) \in \mathbb{G}_2 = E(\mathbb{F}_p)[r]$;
2. Verify that $e(P, \hat{\phi}(Q)) = e'(\phi(P), Q)$.

Figure 7.1: Instantiation of the Verifiable Delay Function over $\mathbb{F}_p$.

We denote E^t to be a quadratic twist of E. Using the notations of Section 2.7, $E^t = \tau_2(E)$ (see Page 30 for the definition of τ_2). The rational maps defining τ_2 are defined over $\mathbb{F}_{p^2}$, but E^t is defined over $\mathbb{F}_p$. Moreover, $\#E^t(\mathbb{F}_p) = p + 1$, and it contains exactly one cyclic subgroup $\mathbb{G}_1^t = E^t(\mathbb{F}_p)[r]$. We shall then set $\mathbb{G}_2 := \tau_2^{-1}(\mathbb{G}_1^t)$. Finally, the Tate pairing is non-degenerate on $\mathbb{G}_1 \times \mathbb{G}_2$.

The map ϕ will be instantiated with an isogeny of degree ℓ^T, and the map $\hat{\phi}$ with its dual. In practice, we assume that these isogenies are stored as a sequence of T isogenies of degree ℓ (e.g., specified by their kernels), so that evaluating ϕ and $\hat{\phi}$ can be done in time polynomial in ℓ and linear in T. For a representation that is more compact by a (large) constant factor, see Section 7.6.

Because of the way we have chosen ℓ, the graph of (horizontal) ℓ-isogenies containing E is a cycle of length dividing the class number $\#\operatorname{Cl}(\operatorname{End}_p(E))$, thus an isogeny of degree ℓ^T is obtained by choosing a direction on the cycle and composing T isogeny steps each of degree ℓ. The isogeny $\phi : E \to E'$ defines an image curve $E'/\mathbb{F}_p$ having the same group structure as E; in particular we define the cyclic groups $\mathbb{G}_1' = E'(\mathbb{F}_p)[r]$ and $\mathbb{G}_2' = \tau_2^{-1}(E'^t(\mathbb{F}_p)[r])$, where $E'^t = \tau_2(E')$ is a quadratic twist of E'.

Note that it is easy to sample uniformly from any of the groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_1', \mathbb{G}_2'$, in a way that does not reveal discrete logarithms[4]: one simply takes random points on the curves or on their twists (see [BHKL13] for example) and multiplies by the cofactor $(p + 1)/r$. The algorithms defining the VDF are described in Figure 7.1.

The similarity with Wesolowski's VDF is evident here: all ℓ-isogenies with the same direction correspond to an ideal $\mathfrak{a}$ of norm ℓ inside the quadratic imaginary order $\mathcal{O} \simeq \operatorname{End}_p(E)$, which is also a representative of an ideal class in $\operatorname{Cl}(\mathcal{O})$. Composing isogenies corresponds to multiplying ideals, thus ϕ corresponds to $\mathfrak{a}^T$ and $\hat{\phi}$ corresponds to $\mathfrak{a}^{-T}$ in $\operatorname{Cl}(\mathcal{O})$. While Wesolowski raises elements to the power 2^T, we only do the equivalent of raising to the power T, because no

[4] In the elliptic curve cryptography literature, this is typically called *hashing* into the groups.

analogue of the square-and-multiply algorithm is known for composing isogenies. Of course, the fundamental difference is in the way we verify the computation.

7.4.2 VDF from supersingular curves over $\mathbb{F}_{p^2}$

Our second VDF is very similar to the previous one, but uses supersingular curves defined over $\mathbb{F}_{p^2}$, thus sharing some similarities with the Charles–Goren–Lauter hash function [CGL09], and with SIDH [JD11, DFJP14]. It deviates slightly from the paradigm presented in Section 7.3 in that the inputs are not taken in a cyclic group, and evaluation is slower than the previous one by a factor of about 2 (see Section 7.6), but has some advantages over it that will be discussed in Section 7.5.

Like before, we choose a prime p such that $p+1$ contains a large prime factor r, and a small prime ℓ, e.g., $\ell = 2$.[5] We again choose a supersingular elliptic curve E defined over $\mathbb{F}_p$ (this will be necessary to define the orthogonal groups $\mathbb{G}_1, \mathbb{G}_2$), however we see it as a curve over $\mathbb{F}_{p^2}$, so that $t = -2p$ and $\#E(\mathbb{F}_{p^2}) = (p+1)^2$.

Like before, we define $\mathbb{G}_1 = \tau_2^{-1}(E^t(\mathbb{F}_p)[r])$ and $\mathbb{G}_2 = E(\mathbb{F}_p)[r]$, where $E^t = \tau_2(E)$ is a quadratic twist of E over $\mathbb{F}_p$. The maps $\phi, \hat{\phi}$ are again an isogeny of degree ℓ^T and its dual. We look at cyclic isogenies (i.e. isogenies of cyclic kernel). It corresponds to a non-backtracking walk on this isogeny graph.

Definition 7.1 (Non-backtracking walk)**.** *An isogeny walk is called* non-backtracking *if no isogeny step (i.e. a prime degree isogeny of the walk) is followed by its dual.*

Hence, we select a cyclic isogeny by doing a non-backtracking walk on this ℓ-isogeny graph. Over $\mathbb{F}_{p^2}$, there are $(\ell+1)\ell^{T-1}$ possible choices for them, instead of just two in the $\mathbb{F}_p$ case.

On the image curve E', we define $\mathbb{G}'_1 = \phi(\mathbb{G}_1)$ and $\mathbb{G}'_2 = \phi(\mathbb{G}_2)$. However, we are now faced with a difficulty: there is no known efficient way to sample from $\mathbb{G}'_2$ or $\mathbb{G}'_1$, indeed E' is generally defined over $\mathbb{F}_{p^2}$ and it has therefore no $\mathbb{F}_p$-twists. To bypass this problem, we deviate from the abstract description of Section 7.3, obtaining an r-to-1 map instead of a bijection. Let π be the Frobenius endomorphism of E over $\mathbb{F}_p$, the trace map on E over $\mathbb{F}_{p^2}$ is the map defined in [Gal05, page 194] as follows:

$$\begin{aligned} \mathrm{Tr} : E &\to E, \\ P &\mapsto P + \pi(P). \end{aligned}$$

In particular, the trace map sends a point $E[r]$ into $\mathbb{G}_2$: if $P \in E[r]$, P is defined over $\mathbb{F}_{p^2}$ and $\pi(\mathrm{Tr}(P)) = \mathrm{Tr}(P)$ which means that $\mathrm{Tr}(P) \in E(\mathbb{F}_p)$. Moreover, $\pi(P) = -P$ for $P \in \mathbb{G}_1$ and the dual of $1+\pi$ is $1-\pi$. Finally, we obtain that for all $P \in \mathbb{G}_1$ and $R \in E[r]$,

$$e(P, \mathrm{Tr}(R)) = e(P, (1+\pi)(R)) = e((1-\pi)(P), R) = e([2]P, R) = e(P, R)^2.$$

We thus define our VDF as

$$\begin{aligned} f : E'[r] &\to \mathbb{G}_2, \\ Q &\mapsto (\mathrm{Tr} \circ \hat{\phi})(Q), \end{aligned}$$

and the verification is done by checking a pairing equation as before. The algorithms are described in Figure 7.2.

[5] For this VDF, there is no practical reason to choose any other prime than $\ell = 2$.

$\mathsf{Setup}(\lambda, T)$

1. Choose primes r, p with the properties above, according to the security parameter λ;
2. Select a supersingular curve E defined over $\mathbb{F}_p$;
3. Perform a random non-backtracking walk of length T in the ℓ-isogeny $\mathbb{F}_{p^2}$-graph, defining a cyclic ℓ^T-isogeny $\phi : E \to E'$ and its dual $\hat{\phi}$;
4. Choose a generator P of $\mathbb{G}_1 = \tau_2^{-1}(E^t(\mathbb{F}_p)[r])$, and compute $\phi(P)$;
5. Output $(\mathsf{ek}, \mathsf{vk}) = \big(\hat{\phi}, (E, E', P, \phi(P))\big)$.

$\mathsf{Eval}(\hat{\phi}, Q \in E'[r])$

1. Compute and output $(\mathrm{Tr} \circ \hat{\phi})(Q)$.

$\mathsf{Verify}(E, E', P, Q, \phi(P), (\mathrm{Tr} \circ \hat{\phi})(Q))$

1. Verify $(\mathrm{Tr} \circ \hat{\phi})(Q) \in \mathbb{G}_2 = E(\mathbb{F}_p)[r]$;
2. Verify that $e(P, (\mathrm{Tr} \circ \hat{\phi})(Q)) = e(\phi(P), Q)^2$.

Figure 7.2: Instantiation of the Verifiable Delay Function over $\mathbb{F}_{p^2}$.

A bijective VDF over $\mathbb{F}_{p^2}$. If a bijection is wanted, an alternative VDF using the $\mathbb{F}_{p^2}$-graph would swap roles by having E' defined over $\mathbb{F}_p$, and E over $\mathbb{F}_{p^2}$. During the Setup phase, a basis (P, R) of $\mathbb{G}_1 \times \mathbb{G}_2$ is sampled by evaluating $\hat{\phi}/2^T$ on a basis of $\mathbb{G}_1' \times \mathbb{G}_2'$, and it is added to the verification key vk. Then, sampling Q in $\mathbb{G}_2'$ is easy, and verifying that $\hat{\phi}(Q) \in \mathbb{G}_2$ can be done by checking that $e(R, \hat{\phi}(Q)) = 1$. However this protocol is less efficient, because verification requires two pairing computations instead of one.

7.4.3 Properties of the VDFs.

In slight disagreement with the definitions of [BBBF18], the Setup routines presented here take $O(T)$ time to compute the isogenies $\phi, \hat{\phi}$, and produce evaluation keys of size $O(T)$. While the size of the evaluation key can be reduced by redoing parts of the computation in Eval (see Section 7.6), the only known way to verify the public parameters is to, essentially, rerun the Setup.

We also note that, although T can be arbitrary (we discuss bounds on T in the next section), neither of our VDFs is incremental in the sense of [BBBF18], meaning that a single parameter set produced by Setup shall support more than one delay T. A possible workaround is to have Setup include some intermediate curves in the verification key, so that a single Setup can be used for many delay parameters up to T, at the cost of increasing the size of the verification key.

Finally, the VDF over $\mathbb{F}_p$ is decodable in the sense of [BBBF18], meaning that given the output $\hat{\phi}(Q)$ one can compute the input Q (although not more efficiently than evaluating $\hat{\phi}$); the VDF over $\mathbb{F}_{p^2}$, on the other hand, is obviously not decodable because it is non-injective.

7.5 Security and parameter sizes

We now give formal security definitions and proofs, following [BBBF18]. A VDF must satisfy three security properties: *correctness*, *soundess*, and *sequentiality*, as defined below. In [BBBF18],

soundness is a weaker property where the evaluator is allowed a negligible cheating probability; we introduce here the stronger notion of perfect soundness, which is achieved by our VDFs.

Definition 7.2 (Correctness, soundness)**.** *The VDFs of Section 7.4 are correct if, for any λ, T, public parameters $(\mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{Setup}(\lambda, T)$, and all input Q, if $R \leftarrow \mathsf{Eval}(\mathsf{ek}, Q)$ then $\mathsf{Verify}(\mathsf{vk}, Q, R)$ outputs* true*.*

They are perfectly sound if for all λ, T, public parameters $(\mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{Setup}(\lambda, T)$, and all input Q, if $R \neq \mathsf{Eval}(\mathsf{ek}, Q)$ then $\mathsf{Verify}(\mathsf{vk}, Q, R)$ outputs false*.*

Theorem 7.3. *The VDFs of Section 7.4 are correct and perfectly sound.*

Proof. The map $R \mapsto e(P, R)$ is a group isomorphism between the output space $\mathbb{G}_2 \subset E[r]$ and the multiplicative subgroup $\mu_r \subset \mathbb{F}_{p^2}$. Hence, verification succeeds if and only if the output is correct. □

Sequentiality is the defining property of VDFs, and is much subtler to define. Intuitively, we want it to be impossible to evaluate the VDF faster than running Eval, even given an unbounded amount of parallel resources, and even if the adversary is allowed a large amount of precomputation after the public parameters are generated. We must of course exclude trivial cases where, for example, the adversary precomputes a list of input-output pairs, hence we model security as a game where the adversary is allowed a polynomial amount of precomputation, after which he receives a random input point Q and must produce the output $\hat{\phi}(Q)$ (or $\mathrm{Tr} \circ \hat{\phi}(Q)$) faster than Eval with non-negligible probability. We also introduce here a new definition: if the adversary cannot break sequentiality, even when he is allowed a quantum precomputation before seeing the point Q, we say that the VDF is quantum annoying.

Definition 7.4 (Sequentiality, quantum annoyance)**.** *The VDFs of Section 7.4 are sequential if no pair of randomized algorithms $\mathcal{A}_0$, which runs in total time $\mathrm{poly}(T, \lambda)$, and $\mathcal{A}_1$, which runs in parallel time less than T, can win with non-negligible probability the following sequentiality game*

1. $(\mathsf{ek}, \mathsf{vk}) \xleftarrow{\$} \mathsf{Setup}(\lambda, T)$*, where the random input tape to* Setup *is filled with uniformly distributed bits,*
2. $A \leftarrow \mathcal{A}_0(\lambda, \mathsf{ek}, \mathsf{vk}, T)$*,*
3. $Q \xleftarrow{\$} \mathbb{G}_2'$*, uniformly sampled,*
4. $Q' \leftarrow \mathcal{A}_1(A, \mathsf{vk}, Q)$*,*

where winning is defined as outputting $Q' = \hat{\phi}(Q)$ (or $Q' = \mathrm{Tr} \circ \hat{\phi}(Q)$).

Moreover, if $\mathcal{A}_0$ is allowed a quantum computation in $\mathrm{poly}(T, \lambda)$, we say that the VDFs are quantum annoying*.*

We leave aside the question of formally defining a computational model where "running in parallel time less than T" has a definite meaning; see [BBBF18, Wes19] for details.

We shall see soon that Setup must use a secret randomness to select the starting curve $E/\mathbb{F}_p$; after that, Setup is only left with choosing the isogeny $\phi : E \to E'$ and the generator $P \in E[r]$, and both choices can be done using public randomness. Furthermore $\mathcal{A}_0$ is allowed $\mathrm{poly}(T)$ computation, so it can compute $\hat{\phi}$ and evaluate ϕ on P (and also evaluate $\hat{\phi}$ on polynomially many points of $\mathbb{G}_2'$). Hence, choice of $E/\mathbb{F}_p$ aside, Setup can be absorbed into $\mathcal{A}_0$; this justifies defining the following problem, which is a simple rewording of the sequentiality hypothesis:

Definition 7.5 (Isogeny shortcut problem (over k))**.** *Let E be a curve uniformly sampled in the set of all supersingular curves defined over $\mathbb{F}_p$. Given an isogeny $\phi : E \to E'$ of degree ℓ^T to a curve E'/k, with $k = \mathbb{F}_p$ or $k = \mathbb{F}_{p^2}$; being allowed a precomputation taking total time* $\mathrm{poly}(T, \lambda)$*, evaluate $\hat{\phi}(Q)$ on a random point $Q \in E'(k)[r]$ in parallel time less than T.*

7.5.1 Attacks

We now discuss three natural attack strategies on the isogeny shortcut problem, and we use them to set parameter sizes. We summarize their complexities in Table 7.2.

Pairing inversion. The simplest attack exploits the same properties as the verification. It works both against the VDFs and the generalization of BLS signatures sketched in Section 7.3. Given $P, \phi(P), Q$, to compute $\hat{\phi}(Q)$ (or $(\mathrm{Tr} \circ \hat{\phi})(Q)$) it is enough to solve the pairing inversion problem $e(P, \cdot\,) = e'(\phi(P), Q)$, studied in [GHV08]. Note that this attack must be repeated for each new input Q.

The hardness of the pairing inversion problem impacts the size of r and p. Given that our curves have embedding degrees 2 or 1, the best algorithm at our disposal is the Number Field Sieve for $\mathbb{F}_{p^2}$, described in Chapter 1. To reach the 128-bit security level, we take p around 1500 bits, and r of 256 bits.

Computing shortcuts. A different path to breaking our VDFs consists in finding a "simpler" isogeny from E to E', agreeing with ϕ on $E[r]$, but taking less parallel time to compute. This kind of attacks can be decomposed in two steps: first find a "simpler" isogeny $\psi : E \to E'$ (e.g., of lower degree), then find an endomorphism $\omega \in \mathrm{End}(E)$ such that $\omega \circ \hat{\psi}$ agrees with $\hat{\phi}$ on $E'[r]$.

Concerning the first step, when $\deg \phi$ is super-polynomial in p, a lower degree isogeny $\psi : E \to E'$ always exists; indeed Pizer [Piz90, Piz98] has shown that ℓ-isogeny graphs of supersingular curves over $\overline{\mathbb{F}}_p$ are optimal expanders for any prime ℓ, and thus have diameter in $O(\log(p))$, implying that there is an ℓ-isogeny walk connecting E to E' of degree polynomial in p. However, it may be difficult to compute such an isogeny in general: the best generic algorithm in the case of $\mathbb{F}_{p^2}$-graphs is a birthday paradox method [GHS02, DG16], that finds a collision in $O(\sqrt{p})$ isogeny steps on average. Note that the only quantum speedup known for this problem is a generic Grover search, giving a square-root acceleration at best [BJS14].

For curves over $\mathbb{F}_p$, computing the structure of the class group $\mathrm{Cl}(\mathrm{End}_p(E))$ allows an attacker to find an equivalent isogeny ψ, of (smooth) lower degree. A similar computation is at the hearth of the signature scheme CSI-FiSh [BKV19], and has recently been demonstrated to be doable for primes of around 500 bits. Nevertheless, the asymptotically best algorithm, due to Jao and Soukharev [JS10], computes an equivalent isogeny of smooth subexponential degree using $L_p(1/2)$ operations, and is thus not better than the pairing inversion attack mentioned above. We will sketch later how a similar attack can be performed in polynomial time on a quantum computer.

After computing a "simpler" isogeny $\psi : E \to E'$, we are left with the problem of finding ω such that $\omega \circ \hat{\psi} = \hat{\phi}$ on $E'[r]$. This problem can be solved by computing discrete logarithms in $E[r]$, which is again not easier than the pairing inversion problem; however, this attack needs only to be performed once on the public parameters, and can then be used to speed up any evaluation.

So far, we have only discussed the computation of shortcuts in the generic case; however, when E or E' are special curves, there are much better ways to solve this problem, that would lead to a complete break of our VDFs. We discuss this issue in Subsection 7.5.2.

Parallel isogeny evaluation. Finally, the last attack path would be to find a better parallel algorithm for evaluating isogenies of degree ℓ^T. All known algorithms require to go through each of the T intermediate curves, one after the other. Barring shortcut techniques as described above, it seems unlikely that an algorithm "skipping" intermediate curves could exist. This is not dissimilar from the case of VDFs based on groups of unknown order, where one argues that in order to compute g^{2^T} all intermediate values g^{2^i} must be computed. After all, a 2-isogeny is only a simple generalization of the multiplication-by-2 map of an elliptic curve, it thus seems believable that a chain of 2-isogenies must be evaluated sequentially passing through all intermediate curves.

It is certainly possible to aggregate steps in blocks, e.g., replace two 2-isogenies with one 4-isogeny, as it is typically done in implementations of SIDH/SIKE [CLN16]. This is analogous to replacing n squarings with a single power-of-2^n in group-based VDFs; previous work on parallel modular exponentiation suggests that, in some complexity models, there may be a small asymptotic gain in doing so [BS07], however the viability of these algorithms has never been validated in practice. At any rate, algorithms for parallel modular exponentiation would need to be adapted for isogeny evaluation, and we believe that, in this respect, isogeny-based VDFs can only be as weak as group-based VDFs, but no more. This is certainly the newest and most unusual problem in the area of elliptic curve cryptography, and the one that needs more investigation.

Bounds on T. None of the attacks so far has set an upper bound on T. By the birthday paradox, we shall take T smaller than the square root of the size of the isogeny graph, because a loop in the isogeny walk could be optimized away from Eval. Given that the size of the isogeny graphs is $O(\sqrt{p})$ and $O(p)$ respectively, we obtain bounds of $O(\sqrt[4]{p})$ for $\mathbb{F}_p$, and $O(\sqrt{p})$ for $\mathbb{F}_{p^2}$.

However, these bounds are much higher than the best attacks, that are subexponential in p. Thus T is effectively only bounded by the theoretical limit of being subexponential in λ.

On future attack improvements. While the isogeny shortcut problem is new, we argue that improvements to any of the three attack strategies outlined above would have important consequences in cryptography, beyond our VDF constructions. Pairing inversion is a well-known problem in classical cryptography, and an attack on it will affect a large number of pairing-based protocols [GHV08]. Shortening an isogeny walk from a curve leads to an endomorphism of this curve; this is believed to be hard computational task, which underlies the security of other cryptosystems [CGL09, GPS17]. Finally, faster parallel isogeny computations will benefit other isogeny-based cryptographic protocols, such as key exchange [JD11, CLM+18, AKC+17] and signatures [YAJ+17, DG19].

Quantum security. We briefly analyze our proposals in the post-quantum setting. Obviously, Shor's algorithm breaks the pairing inversion problems in polynomial time, thus our VDFs cannot be considered post-quantum. However, looking at Definition 7.4, we see that this attack can only be applied after the input point Q is given to $\mathcal{A}_1$; thus our VDFs have a chance of being *quantum annoying* as defined there. In a plausible future where quantum computers do exist, but are very expensive and slow, it may still be more interesting to evaluate the VDF in the legitimate way, rather than attack the pairing inversion problem with Shor's algorithm. We argue that, given current knowledge, our VDF over $\mathbb{F}_{p^2}$ is quantum annoying, whereas the one over $\mathbb{F}_p$ is not.

Indeed, as long as the input point Q is unknown, the only strategy currently available for $\mathcal{A}_0$ is to compute an isogeny shortcut, as described previously. In the $\mathbb{F}_{p^2}$ case, this would involve finding a cycle in the isogeny graph through $E/\mathbb{F}_p$ and $E'/\mathbb{F}_{p^2}$, a problem that is believed to be quantum-resistant when E and E' are generic supersingular curves [GPST16, EHL+18].

	Classical		Quantum	
	$\mathbb{F}_p$ graph	$\mathbb{F}_{p^2}$ graph	$\mathbb{F}_p$ graph	$\mathbb{F}_{p^2}$ graph
Computing shortcuts	$L_p(1/2)$	$O(\sqrt{p})$	polylog(p)	$O(\sqrt[4]{p})$
Pairing inversion	$L_p(1/3)$	$L_p(1/3)$	polylog(p)	polylog(p)

Table 7.2: Complexity of the known attacks on the sequentiality of our VDFs, assuming the endomorphism rings of the supersingular curves are unknown (see Subsection 7.5.2 for a polynomial time classical attack when the endomorphism rings are known). Computing shortcuts targets public parameters independently of the input to the VDFs, and can be thus be run as a pre-computation. Pairing inversion attacks a single input point, and must be re-run for every new input.

For the $\mathbb{F}_p$ case, on the other hand, it is enough to compute the structure of $\mathrm{Cl}(\mathrm{End}_p(E))$, along with a basis of "short" generators, a task doable in polynomial time on a quantum computer using Kitaev's generalization of Shor's algorithm [Kit95]. Then an isogeny $\psi : E \to E'$ of lower degree defined over $\mathbb{F}_p$ can be computed by solving a closest vector problem: although polynomial-time lattice reduction algorithms (both classical and quantum) can only reach isogeny degrees exponential in $\log(p)$, this may be enough to break some large delay parameters, and it can be very efficient in practice, as showcased by the signature scheme CSI-FiSh [BKV19]. Finally, since ψ and ϕ are both defined over $\mathbb{F}_p$, the subgroup $\mathbb{G}_2 = E(\mathbb{F}_p)[r]$ is an eigenspace for the endomorphism $\hat{\psi} \circ \phi$; then a discrete logarithm computation in $\mathbb{G}_2$ finds a scalar s such that $[s] \circ \hat{\psi} = \hat{\phi}$ on $\mathbb{G}_2'$.

Security of the identification protocol. For completeness, we briefly come back to the security of the generalization of the BLS identification protocol sketched in Section 7.3.

We are not interested in sequentiality in this case, thus shortcut attacks are not relevant here. Instead, key recovery is equivalent to the problem of finding a secret isogeny $\phi : E \to E'$, given E, E', a basis (P, Q) of $E[r]$, and $\phi(P), \phi(Q)$. This problem is much more similar to classical problems in isogeny based cryptography, and is obviously harder than the isogeny shortcut problem.

The best known classical attacks, both for the $\mathbb{F}_p$ and the $\mathbb{F}_{p^2}$ case, are in the square root of the graph size (respectively, $O(\sqrt[4]{p})$ and $O(\sqrt{p})$). But key recovery is hard even for quantum computers: the best attack for the $\mathbb{F}_p$ case is Kuperberg's algorithm for the *Hidden Shift Problem* [Kup05, Reg04, Kup13, CJS14, BS20, BIJ18, JLLRL18, BLMP19], which finds ϕ in $\exp(\sqrt{\log(p)})$ quantum operations; whereas in the $\mathbb{F}_{p^2}$ case quantum computers give a square-root speedup via Grover's algorithm at best [BJS14].

Hence, both identification protocols have a security property similar to the *quantum annoyance* defined in Definition 7.4: any forgery requires running a new instance of Shor's algorithm, while key recovery is infeasible on quantum computers. This may be a useful replacement for basic BLS signatures in contexts where Shor's algorithm is slow and expensive, and signatures must be produced fast.

Finally, we remark that our protocol, unlike BLS, is succinct, in the sense that the secret isogeny is potentially sub-exponentially larger than the proof of knowledge. At present, this seems rather limited, since our protocol is not zero-knowledge, however we hope that further research may add more useful properties to it.

7.5.2 Shortcut attacks on special curves

We now come back to the shortcut attacks analyzed previously. We saw that the best algorithms available in the general case have exponential or sub-exponential complexity, and are in general not better than a simple pairing inversion attack. However, when the endomorphism ring of the starting curve E is known, a much better algorithm exists, completely breaking sequentiality of our VDFs. We now present a sketch of the attack, and the only known solution to avoid it.

Attack overview. We shall suppose that the delay parameter T is super-linear in $\log(p)$. To simplify our description we also assume that E is the curve defined by the equation $y^2 = x^3 + x$, with j-invariant $j = 1728$. However, the attack can be generalized to an arbitrary curve provided we know its endomorphism ring. It can also be applied to our VDF over $\mathbb{F}_p$, because an attacker is not bound to keep all computations in $\mathbb{F}_p$.

The attack has two main steps. First, we compute an alternative isogeny $\psi : E \to E'$ with a powersmooth and reasonably small degree (polynomial in p). This is achieved using the algorithms of Chapter 6. Second, we compute an endomorphism $\omega \in \mathrm{End}(E)$ such that the actions of $\omega \circ \hat{\psi}$ and $\hat{\phi}$ are identical on $E[r]$. By expressing ω on a set of generators of $\mathrm{End}(E)$, we are able to evaluate $\omega \circ \hat{\psi}$ efficiently on $E[r]$, and thus we can answer evaluation queries in a time much shorter than T.

Computing shortcuts. Let $\phi : E = E_0 \to E_T = E'$ be given as a composition of degree ℓ isogenies. We now show how to compute an alternative isogeny $\psi : E \to E'$ with much shorter degree.

A natural idea to solve this problem is to translate this problem to an analogous problem in the quaternion algebra $B_{p,\infty}$ ramified at p and at infinity, solve the problem in the quaternion algebra, and translate the solution back to the geometric setting. We investigated this problem in Chapter 6, more precisely in Section 6.6.3. The curve E_0 corresponds to Example 6.18 and $\mathrm{End}(E_0)$ is isomorphic to a maximal order $\mathcal{O}_0$ of $B_{p,\infty}$ which is fully known. Algorithms of Chapter 6 require compute torsion points of order $\deg \phi$, which have exponential size in general.

We adapt an idea used in the collision algorithm of [PL17, EHL+18] to avoid this problem. Let $\phi_i : E_0 \to E_i$ correspond to the first i steps of the isogeny. Let $\mathcal{I}_i$ be the corresponding ideal, and let $n(\mathcal{I}_i) = \ell^i$ denote its norm. Assume we have already computed an ideal $\mathcal{J}_i$ in the class of $\mathcal{I}_i$ with powersmooth norm (see Algorithm 6.14). We sketch how to compute an ideal in the class of $\mathcal{I}_{i+1}$ with powersmooth norm.

1. Compute the $\ell + 1$ ideals $\mathcal{K}_{i+1,k}$, $k = 0, \ldots, \ell$ with norm $n(\mathcal{J}_i)\ell$ such that $\mathcal{K}_{i+1,k} \bmod n(\mathcal{J}_i)\mathcal{O}_0 = \mathcal{J}_i$. Algorithms for this task are provided in [KV10], and correspond to writing $\mathcal{J}_i = \mathcal{O}_0 N(\mathcal{J}_i) + \mathcal{O}_0\alpha_i$ using Algorithm 6.2, and then find $\mathcal{K}_{i+1,k}$ of the form $\mathcal{O}_0 n(\mathcal{J}_i)\ell + \mathcal{O}_0\tilde{\alpha}_{i+1,k}$ where $\tilde{\alpha}_{i+1,k}$ is found by solving $\tilde{\alpha}_{i+1,k} = \alpha \bmod n(\mathcal{J}_i)\mathcal{O}_0$ (representing $\mathcal{O}_0/n(\mathcal{J}_i)\mathcal{O}_0$ as in Section 6.3.3).

2. Apply Algorithm 6.14 to each $\mathcal{K}_{i+1,k}$ to obtain new ideals $\mathcal{J}_{i+1,k}$ in the same classes respectively.

3. Translate each ideal $\mathcal{J}_{i+1,k}$ to an isogeny $\psi_{i+1,k}$ using Algorithm 6.8.

4. Identify the (usually unique) k such that the image of $\psi_{i+1,k}$ has j-invariant $j_{i+1} = j(E_{i+1})$.

5. Set $\mathcal{J}_{i+1} = \mathcal{J}_{i+1,k}$.

To obtain the desired isogeny ψ, we repeat those steps for $i = 1, \ldots, T-1$. When $i = T-1$, we additionally set $\psi = \psi_{T,k}$.

The heuristic bounds and the experiments in [KLPT14] show that the degree of ψ_i is polynomial in p (more precisely $O(p^{7/2})$) and the computation can be completed in time $\mathrm{poly}(T, \log(p))$. The isogeny $\psi : E \to E'$ has powersmooth degree much smaller than that of ϕ, and can therefore be evaluated much faster.

Matching image points on the r-torsion. Using the notations of Chapter 6, the endomorphism $\theta := \hat{\psi} \circ \phi$ can be written as $\theta = a_1\mathbf{1} + a_2\mathbf{i} + a_3\mathbf{j} + a_4\mathbf{k}$ with $a_1, a_2, a_3, a_4 \in \mathbb{Z}[1/2]$. Moreover we have

$$a_i = \langle \theta, \alpha_i \rangle := (\theta \circ \hat{\alpha}_i + \alpha_i \circ \hat{\theta})/2$$

for $\alpha_i = \mathbf{1}, \mathbf{i}, \mathbf{j}, \mathbf{k}$ respectively, and these coefficients can be computed using a variant of Schoof's algorithm [Koh96, Theorem 81], by evaluating those maps on small torsion points and applying the Chinese remainder theorem. Note that $|a_i| \leq \deg\theta \deg\alpha_i$, so this computation can be performed in time $\mathrm{poly}(T, \log p)$.

If we now set $\omega = R\hat{\theta}$, where $R = \ell^T/(a_0^2 + a_1^2 + pa_2^2 + pa_3^2)$, then $\omega \circ \hat{\psi} = \hat{\phi}$. But ω can be evaluated at any point $Q \in E[r]$ as

$$\omega(Q) = \sum [Ra_i \bmod r]\, \hat{\alpha}_i(Q),$$

at a cost of only $O(\log(r))$ operations. Thus we can replace $\mathsf{Eval}(Q)$ with the evaluation of $\hat{\psi}$ followed by the evaluation of ω, for a total costs of only $\mathrm{polylog}(p)$, which is less than T by hypothesis.

Countering the attack. The KLPT algorithm only works when the starting curve E has an endomorphism ring that is, in their words, extremal and special. Extremal means that E is defined over $\mathbb{F}_p$, a condition common to all instantiations of our VDF; however only few curves are also special, for example $j(E) = 1728$, or other curves with complex multiplication by an order with small discriminant.

The KPLT algorithm extends to a non-special curve E, when a path $E \to E_0$ to a special curve E_0 is known. Unfortunately, all known methods to select random supersingular curves do so by starting a random walk from some special curve; hence, there is no known way to produce a random supersingular curve $E/\mathbb{F}_p$ without producing a backdoor $E \to E_0$.

At present, the only way to counter the attack presented here is to use a trusted setup to produce a random curve $E/\mathbb{F}_p$, i.e., having a trusted authority (or many trusted authorities engaged in a multi-party protocol) compute a walk $E_0 \to E$ from a special curve E_0, and then throw the backdoor away.

A note on ordinary curves. One can naturally ask whether it is possible to obtain VDFs from ordinary isogeny graphs. Although it is conceivable to have a variant of our VDF over $\mathbb{F}_p$ using ordinary curves, no secure instantiation is currently known. Indeed, all known ordinary pairing-friendly curves are obtained using variations of the CM method, and thus have small quadratic discriminant and small isogeny class. In this case it is possible to compute the structure of $\mathrm{End}(E)$, and do a shortcut attack similar to the one above.

We proceed in two steps as before. We first find an isogeny $\psi : E \to E'$ of small powersmooth degree; since the isogeny class is small, this can even be done by exhaustive search.

Then, we are left with the problem of finding $\omega \in \mathrm{End}(E)$ such that $\omega \circ \hat{\psi} = \hat{\phi}$ when restricted to $E[r]$. We proceed as before: using Schoof's algorithm we compute $\theta = \hat{\psi} \circ \phi = a + b\pi$ for some $a, b \in \mathbb{Q}$, then we set $\omega = \ell^T \hat{\theta}/(a^2 + pb^2)$, and we replace Eval by $\omega \circ \hat{\psi}$.

A family of ordinary pairing friendly elliptic curves with generic discriminant would provide the perfect instantiation for our VDFs, as it would not be vulnerable to any known shortcut attack, and thus would not need a trusted setup. Unfortunately, all known constructions of pairing-friendly elliptic curves use complex multiplication and hence produce curves with small discriminants [FST10].

7.6 Implementation

Our proposed VDFs can be easily implemented using the fundamental blocks already available for pairing-based and isogeny-based cryptography. A drawback of our method being the long setup time and the large evaluation key, we present here an implementation that improves both by orders of magnitude.

7.6.1 Evaluation

We focus on 2-isogenies, as they are the most obvious candidate for an implementation. There are two standard ways to compute a 2-isogeny walk from a curve $E : y^2 = f(x)$. The first is to factor the 2-division polynomial $f(x)$ to obtain all the points of order 2, then use Vélu's formulas [Vél71] to test all directions and step in the wanted one. Since Vélu's formulas also produce the generator of the dual isogeny to the direction one is coming from, this root can be quotiented out from $f(x)$, and thus we are left with solving one square root per curve. The second way is to take a point at random on E and multiply it by the cofactor $\#E/2$. If we obtain a 2-torsion point defining the wanted direction, then we compute it and we move to the next curve; otherwise we try with a different point. Both ways require $O(\log(p))$ operations in the base field for one step, and thus $O(T \log(p))$ operations to compute the full isogeny walk. After the isogeny ϕ is computed, the list of the kernel points can be stored so to be able to evaluate ϕ in $O(T)$ operations. However, this implies storing T points and curves, which may require a large storage.

Fortunately, using isogeny evaluation techniques pioneered in SIDH [DFJP14], and applied in [DPB17] to the CGL hash function [CGL09], it is possible to absorb the $\log(p)$ factor and shorten the evaluation key size by the same amount. For this, we choose a prime of the form $p = 2^n fr - 1$, so that all curves in the isogeny graph have rational points of order 2^{n-1} or 2^n (depending on whether we use $\mathbb{F}_p$-graphs or $\mathbb{F}_{p^2}$-graphs). This way, a single point P_i on E_i can be used to define n (or $n-1$) consecutive steps in the graph, and the corresponding isogeny can be evaluated in $n \log(n)$ operations using the optimal strategy techniques from [DFJP14].

More in detail, in the $\mathbb{F}_{p^2}$ case, the curve E_i has group structure $E(\mathbb{F}_{p^2}) \simeq (\mathbb{Z}/(p+1)\mathbb{Z})^2$, and is usually not defined over $\mathbb{F}_p$. We can compute a point P_i of order 2^n by taking a point at random and multiplying by fr, then verifying that P_i has the wanted order. We check that P_i does not start a backtracking isogeny and we use it to advance n steps in the graph, then we start again.

In the $\mathbb{F}_p$ case, because we chose a curve on the surface, the group structure is $E(\mathbb{F}_p) \simeq \mathbb{Z}/\frac{p+1}{2}\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$ (see [MVO91]), hence the highest order we can get for P_i is 2^{n-1}. Such point P_i will define 2^{n-2} horizontal isogeny steps in the "positive" direction determined by the ideal $(\pi - 1) \subset \mathrm{End}(E)$, plus one last step that is either in the same direction, or going to the floor. To avoid "getting stuck" on the floor, we use P_i to advance $n-2$ steps, then start again.

Using these techniques, only $\approx T/n$ points need to be computed and the full walk is computed in $O(T \log n)$ operations. One has the choice between storing all the intermediate 2-torsion points, or storing only the higher order points P_i. In the first case, we use $O(T)$ storage and evaluation time; in the second case, we use $O(T/n)$ storage and $O(T \log n)$ evaluation time. Since $n \approx \log p$, the slowdown in the second case is likely to be negligible in front of other factors, such as data transfer delays, or speedups due to dedicated hardware.

In practice, we use a projective (x, z)-only Montgomery model for our curves, for which small degree isogeny formulas are the most efficient [CLN16]. Points defined over $\mathbb{F}_{p^2}$ are then stored in $4\log_2(p)$ bits, and a curve is represented by $y^2 = x^3 + ax^2 + x$ using $2\log_2(p)$ bits. The isogeny $\hat{\phi}$ is decomposed in small degree isogenies, and each one is represented by its kernel and its image curve. If we choose to represent $\hat{\phi}$ as a composition of 2-isogenies, its representation is stored in $2T\log_2(p)$ bits. If we decide to represent it as a composition of 2^n-isogenies, storing kernels and curve coefficients requires $T/n(4\log_2(p) + 2\log_2(p)) = 6T\log_2(p)/n$ bits.

7.6.2 Verification

For verification, we apply standard optimization techniques for the pairing computation. We stress that, while most of the implementation efforts on pairing have focused on ordinary elliptic curves with smaller field sizes, such as BN curves [BN06], our situation is somewhat different. Contrary to the cases of ordinary curves studied in Chapter 4 and 5, the Tate pairing is here more efficient than the ate pairing: our curves have a trace t as large as p which is much larger than r as we choose $p = 2^n fr - 1$. Hence, the Tate pairing t_r is preferred because it features a shorter Miller loop than in the ate pairing case. Thus, the verification equation (e.g., in the $\mathbb{F}_p$-case)

$$t_r(\hat{\phi}(Q), P) = t_r(Q, \phi(P))$$

can be checked by computing two Miller loops and one final exponentiation. We use common optimizations for the Miller loop, such as quadratic twist tricks. In the final exponentiation, we benefit from the special form of the prime p, indeed

$$\frac{p^2-1}{r} = (p-1)\frac{p+1}{r} = (p-1)2^n f = (2^n fr - 2)2^n f = 2^{n+1} f(2^{n-1} fr - 1).$$

7.6.3 Measurements

To validate our proposals, we implemented a (non-optimized) proof of concept in SageMath [Sag18]. The code repository is publicly accessible at:

`https://github.com/isogenies-vdf/isogenies-vdf-sage/`.

For a 128-bit secure VDF, we choose a prime r of 256 bits, and set $n = 1244$, $f = 63$ to obtain a 1506-bit prime $p = 2^{1244}63r - 1$. In 2020, discrete logarithm computations in the subgroup of order r of $\mathbb{F}_{p^2}$, using the best available variants of NFS, are believed to require more than 2^{128} computations.

We ran benchmarks on an Intel Core i7-8700 processor clocked at 3.20GHz. We aimed a few minutes evaluation time, and thus we set the delay parameter $T \approx 2^{16}$. Whenever a VDF would be computed in a longer time, we provide the throughput by millisecond. Currently, our pairing implementation is faster over $\mathbb{F}_p$ because these curves benefit of the distortion map to compute the pairing entirely over $\mathbb{F}_p$. Over $\mathbb{F}_{p^2}$, points are twice larger and many additional vertical lines and inversions are needed to compute the pairing. The results are given in Table 7.3.

Protocol	Step	ek size	Time	Throughput
$\mathbb{F}_p$ graph	Setup	238 kb	90 s	0.75isog/ms
	Evaluation	–	89 s	0.75isog/ms
	Verification	–	0.3 s	–
$\mathbb{F}_{p^2}$ graph	Setup	491 kb	193 s	0.35isog/ms
	Evaluation	–	297 s	0.23isog/ms
	Verification	–	4 s	–

Table 7.3: Measurements for our VDFs, on a Intel Core i7-8700 @ 3.20GHz, with SageMath 8.5

We stress that these numbers only show that our VDFs are practical, however they do not say much on how they compare to other VDF proposals. Indeed, while setup and verification can be compared on the basis of their speed (in software), it is mostly meaningless to compare evaluation this way.

The meaningful comparison is on circuit area and clock frequency. For two VDFs, comparing the cost of the evaluation for a fixed delay parameter does not make sense: one step for each function may not have the same cost. In order to get a fair comparison, one needs to measure the cost of a single step of the evaluation loop. At this stage, it is impossible for us to give such numbers, however we can give some qualitative arguments to compare our VDFs to the competitors. At the 128 bits security level, the unit step in RSA-based VDFs is a squaring modulo an RSA modulus of more than 2000 bits. This unit step is roughly comparable to one multiplication in our field $\mathbb{F}_p$. In the simplest case, the unit step in our VDFs is the evaluation of a 2-isogeny over $\mathbb{F}_p$ (or $\mathbb{F}_{p^2}$); using the best formulas for Montgomery curves [Ren18], this requires 2 parallel runs of 2 multiplications each. Thus we expect the circuit for one unit step of our VDF to have roughly double the surface and half the clock frequency. Similar considerations also apply to VDFs based on class groups.

7.7 Conclusion and perspectives

We presented two new candidate Verifiable Delay Functions, based on assumptions from pairing-based and isogeny-based cryptography. Our VDFs are practical, and offer several advantages over previous proposals.

- Both our constructions are optimal and perfectly sound. We observe that the construction based on univariate permutation polynomials of Boneh et al. [BBBF18] also has both properties, but its security relies on an ad hoc limit assumption on the amount of parallelism available to the adversary.
- Unlike the VDF constructions of Pietrzak [Pie18] and Wesolowski [Wes19], ours are inherently non-interactive, the output being efficiently verifiable without attaching a proof.
- By using mathematical tools also used in other cryptographic contexts, our constructions benefit from pre-existing research in these areas both from an efficiency and security point of view.
- While the use of isogenies does not magically make our functions post-quantum (in fact they can be broken with a discrete logarithm computation), one of our two constructions still offers some partial resistance to quantum attacks; we call this property quantum annoyance.

The main drawback of our proposals is that, given current knowledge, the only secure way to instantiate our VDFs requires a trusted setup, or, said otherwise, that our VDFs can be easily backdoored. Indeed, both our setups require to start from a supersingular elliptic curve with unknown endomorphism ring. No general algorithm is known to compute the endomorphism ring of supersingular elliptic curves, however the only known ways to generate supersingular curves involve a random isogeny walk from a curve with small discriminant (e.g., $j = 0$ or $j = 1728$), and it has been shown that knowledge of the isogeny walk permits computing the endomorphism ring in polynomial time [KLPT14, EHL+18]. Hence, the only way to instantiate our VDFs involves a trusted setup that performs a random isogeny walk and then forgets it. We stress that trusted setups also appear in other constructions, and that does not rule them out for practical applications; in fact, the Ethereum cryptocurrency is currently considering standardization of a VDF based on a trusted RSA setup [Dra18]. Furthermore, while it is clear that a trusted setup is necessary in the RSA setting, this looks much less like a fatality in our case: it is totally believable that in the near future a way is found to generate random supersingular curves with unknown endomorphism ring, thus bypassing the need for the trusted setup. Finally, a distributed trusted setup with $n - 1$ threshold security can be efficiently constructed in our case purely from isogeny assumptions, whereas the RSA setting requires heavy multi-party computation machinery and very large bandwidth.

Another limitation on the utility of our VDFs is that the time required to setup public parameters is of the same order of magnitude as that required to evaluate the function; furthermore, validating public parameters requires the same amount of time as evaluating the function, and the evaluator is required to use $O(T)$ storage for evaluating in optimal time. While these drawbacks are acceptable in applications that require delays in the order of minutes or hours (the majority of applications in blockchains), they prevent our VDFs from being used with very long delays. In our implementation, we propose some possible tradeoffs that mitigate these problems, however further research is needed to better address them.

At present, our constructions require a trusted setup to generate initial parameters. It is an important open problem to find an algorithm to generate random supersingular curves in a way that does not reveal their endomorphism ring, and we encourage the community to work on it. As long as such an algorithm is missing, it is interesting to look for efficient multi-party algorithms for doing isogeny walks.

It would also be interesting to reduce the cost of validating public parameters, ideally to a time independent from the delay parameter T. Relatedly, our VDFs have large storage requirements for the evaluator; in our implementation we presented a way to mitigate this issue, however this creates a compromise between storage and evaluation time, that needs to be carefully considered by the evaluator, depending on the intended application. More research on practical ways to mitigate the price of the large storage is desirable.

Here we only sketched the shortcut attack against insecure instances using special curves. It would be interesting to do a more detailed analysis of its complexity, of its limitations, and of its possible generalizations; we leave this as future work. We also encourage research on alternative ways to break the Isogeny Shortcut Problem, for example finding ways to parallelize isogeny evaluation.

Finally, our VDFs can be seen as a generalization of BLS signatures: if the isogeny is kept secret, we obtain a proof of knowledge of an isogeny walk between two curves, that can be used for identification or signatures. At the moment, the only advantage over BLS signatures is a weak form of quantum resistance; we hope that further research would add useful properties to our protocol enabling more applications.

Conclusion

In this thesis, we investigated the algorithmic point of view of curve-based cryptography. It includes applications to pairing-based cryptography and isogeny-based cryptography.

On the one hand, our work studied the arithmetic related to pairing-friendly curves. First, it involves the study of the arithmetic efficiency and the security of the discrete logarithm problem in finite field extensions. Secondly, we estimated the arithmetic on pairing-friendly curves. More precisely, we estimated the cost of the pairing computation on the state-of-the-art curves, and obtained new parameters of curves of embedding degrees five to eight. We finally compare our new curves with the current ones, at the same security level (128-bit). In order to get fair estimations of the security of our new curves (as well as for curves of Chapter 4), it would be interesting to investigate the cost of the Tower variant of NFS on concrete instantiations of parameters. The implementation of the optimal ate pairing on our new curves (more precisely, on $k = 8$ curves that are the most promising for efficiency) is slightly different from the current implementations. The library [AG] investigates an efficient implementation in order to get concrete measurement comparisons with the current Barreto-Naehrig, Barreto-Lynn-Scoot and Kachisa-Schaefer-Scott curves. These new curves of embedding degree $k = 8$ are currently in a standardization process.

On the other hand, we looked at the arithmetic of isogeny-based cryptography. We provided an implementation of algorithms that compute endomorphism rings of supersingular curves, given the knowledge of an isogeny from a curve with small discriminant. This work is related to the cryptanalysis of isogeny-based cryptography, but is also interesting for constructing new primitives such as post-quantum isogeny-based signatures. Recently, the construction of Galbraith, Petit and Silva [GPS20] has been generalized by De Feo, Kohel, Leroux, Petit and Wesolowski. This new construction is called SQISign [FKL$^+$20] and will be presented at Asiacrypt 2020. Our implementation is far from efficient for cryptographic size experiments. It would be interesting to provide an implementation in an open-source language, with a more efficient arithmetic.

Isogeny-based cryptosystems compute large degree morphisms between curves, that often affects the efficiency of the cryptographic applications. This comes from the fact that the best algorithm for computing an isogeny of prime degree d is exponential in $\log_2(d)$. We use this assumption in order to design two verifiable delay functions. The computation requires evaluating many isogenies of prime degree, and the verification is computed using pairings. Our VDF are not post-quantum because the security relies on the discrete logarithm problem over a finite field, and uses a trusted setup. We compare our constructions with several other VDF designed by Wesolowski and Pietrzak. The construction of a post-quantum verifiable delay function is currently an open problem that would have interesting applications in real-world cryptography.

Bibliography

[ACC+19] Gora Adj, Daniel Cervantes-Vázquez, Jesús-Javier Chi-Domínguez, Alfred Menezes, and Francisco Rodríguez-Henríquez. On the cost of computing isogenies between supersingular elliptic curves. In Carlos Cid and Michael J. Jacobson Jr:, editors, *SAC 2018*, volume 11349 of *LNCS*, pages 322–343. Springer, Heidelberg, August 2019.

[Adl79] Leonard M. Adleman. A subexponential algorithm for the discrete logarithm problem with applications to cryptography. *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 55–60, 1979.

[AES01] Advanced Encryption Standard (AES). National Institute of Standards and Technology (NIST), FIPS PUB 197, U.S. Department of Commerce, November 2001.

[AFK+13] Diego F. Aranha, Laura Fuentes-Castañeda, Edward Knapp, Alfred Menezes, and Francisco Rodríguez-Henríquez. Implementing pairings at the 192-bit security level. In Michel Abdalla and Tanja Lange, editors, *PAIRING 2012*, volume 7708 of *LNCS*, pages 177–195. Springer, Heidelberg, May 2013.

[AG] Diego F. Aranha and Conrado P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. `https://github.com/relic-toolkit/relic`.

[AKC+17] Reza Azarderakhsh, Brian Koziel, Matt Campagna, Brian LaMacchia, Craig Costello, Patrick Longa, Luca De Feo, Michael Naehrig, Basil Hess, Joost Renes, Amir Jalali, Vladimir Soukharev, David Jao, and David Urbanik. Supersingular isogeny key encapsulation, 2017.

[AKL+11] Diego F. Aranha, Koray Karabina, Patrick Longa, Catherine H. Gebotys, and Julio Cesar López-Hernández. Faster explicit formulas for computing pairings over ordinary curves. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 48–68. Springer, Heidelberg, May 2011.

[Bar13] Razvan Barbulescu. *Algorithmes de logarithmes discrets dans les corps finis*. thèse de doctorat, Université de Lorraine, Nancy, France, 2013. `https://tel.archives-ouvertes.fr/tel-00925228`.

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 757–788. Springer, Heidelberg, August 2018.

[BBF18] Dan Boneh, Benedikt Bünz, and Ben Fisch. A survey of two verifiable delay functions. Cryptology ePrint Archive, Report 2018/712, 2018.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In Shafi Goldwasser, editor, *ITCS 2012*, pages 326–349. ACM, January 2012.

[BCL12] Reinier M Broker, Denis X Charles, and Kristin E Lauter. Cryptographic applications of efficiently evaluating large degree isogenies, August 2012. US Patent 8,250,367.

[BCM+15] Paulo S. L. M. Barreto, Craig Costello, Rafael Misoczki, Michael Naehrig, Geovandro C. C. F. Pereira, and Gustavo Zanon. Subgroup security in pairing-based cryptography. In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *LATINCRYPT 2015*, volume 9230 of *LNCS*, pages 245–265. Springer, Heidelberg, August 2015.

[BD19] Razvan Barbulescu and Sylvain Duquesne. Updating key size estimations for pairings. *Journal of Cryptology*, 32(4):1298–1336, October 2019.

[BDM18] Olivier Bernard, Renaud Dubois, and Simon Masson. Efficient four-dimensional GLV curve with high security. Cryptology ePrint Archive, Report 2018/305, 2018. `https://eprint.iacr.org/2018/305`.

[Ber06] Daniel J. Bernstein. Curve25519: New Diffie-Hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 207–228. Springer, Heidelberg, April 2006.

[BFLS20] Daniel J. Bernstein, Luca De Feo, Antonin Leroux, and Benjamin Smith. Faster computation of isogenies of large prime degre. *ANTS XIV*, 2020.

[BGG+20] Fabrice Boudot, Pierrick Gaudry, Aurore Guillevic, Nadia Heninger, Emmanuel Thomé, and Paul Zimmermann. Comparing the difficulty of factorization and discrete logarithm: A 240-digit experiment. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 62–91. Springer, Heidelberg, August 2020.

[BGGM15] Razvan Barbulescu, Pierrick Gaudry, Aurore Guillevic, and François Morain. Improving NFS for the discrete logarithm problem in non-prime finite fields. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 129–155. Springer, Heidelberg, April 2015.

[BGJT14] Razvan Barbulescu, Pierrick Gaudry, Antoine Joux, and Emmanuel Thomé. A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 1–16. Springer, Heidelberg, May 2014.

[BGK15] Razvan Barbulescu, Pierrick Gaudry, and Thorsten Kleinjung. The tower number field sieve. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 31–55. Springer, Heidelberg, November / December 2015.

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 416–432. Springer, Heidelberg, May 2003.

[BHKL13] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 967–980. ACM Press, November 2013.

[BIJ18] Jean-François Biasse, Annamaria Iezzi, and Michael J. Jr. Jacobson. A note on the security of CSIDH. In Debrup Chakraborty and Tetsu Iwata, editors, *Progress in Cryptology – INDOCRYPT 2018*, pages 153–168, Cham, 2018. Springer International Publishing.

[BJS14] Jean-François Biasse, David Jao, and Anirudh Sankar. A quantum algorithm for computing isogenies between supersingular elliptic curves. In *International Conference in Cryptology in India*, pages 428–442. Springer, 2014.

[BKLS02] Paulo S. L. M. Barreto, Hae Yong Kim, Ben Lynn, and Michael Scott. Efficient algorithms for pairing-based cryptosystems. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 354–368. Springer, Heidelberg, August 2002.

[BKV19] Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. CSI-FiSh: Efficient isogeny based signatures through class group computations. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 227–247. Springer, Heidelberg, December 2019.

[BL08] Daniel Bernstein and Tanja Lange. Analysis and optimization of elliptic-curve single-scalar multiplication. In Gary L. Mullen, Daniel Panario, and Igor E. Shparlinski, editors, *Finite Fields and Applications - 8th International Conference, Fq8, Melbourne, Australia, July 9-13, 2007, Proceedings*, Contemporary Mathematics Series, pages 1–20, United States, 2008. American Mathematical Society.

[BLMP19] Daniel J. Bernstein, Tanja Lange, Chloe Martindale, and Lorenz Panny. Quantum circuits for the CSIDH: Optimizing quantum evaluation of isogenies. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 409–441. Springer, Heidelberg, May 2019.

[BLS03] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN 02*, volume 2576 of *LNCS*, pages 257–267. Springer, Heidelberg, September 2003.

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, September 2004.

[BN06] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In Bart Preneel and Stafford Tavares, editors, *SAC 2005*, volume 3897 of *LNCS*, pages 319–331. Springer, Heidelberg, August 2006.

[BS07] Daniel Bernstein and Jonathan Sorenson. Modular exponentiation via the explicit chinese remainder theorem. *Mathematics of Computation*, 76(257):443–454, 2007.

[BS20] Xavier Bonnetain and André Schrottenloher. Quantum security analysis of CSIDH. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 493–522. Springer, Heidelberg, May 2020.

[BW05] Friederike Brezing and Annegret Weng. Elliptic curves suitable for pairing based cryptography. *Designs, Codes and Cryptography*, 37(1):133–141, 2005.

[CD20] Wouter Castryck and Thomas Decru. CSIDH on the surface. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 111–129. Springer, Heidelberg, 2020.

[CGL09] Denis X. Charles, Eyal Z. Goren, and Kristin E. Lauter. Cryptographic hash functions from expander graphs. *Journal of Cryptology*, 22(1):93–113, January 2009.

[CH07] Jaewook Chung and Masud A. Hasan. Asymmetric squaring formulae. In *18th IEEE Symposium on Computer Arithmetic (ARITH '07)*, pages 113–122, June 2007.

[CJS14] Andrew Childs, David Jao, and Vladimir Soukharev. Constructing elliptic curve isogenies in quantum subexponential time. *Journal of Mathematical Cryptology*, 8(1):1–29, 2014.

[CL15] Craig Costello and Patrick Longa. Four$\mathbb{Q}$: Four-dimensional decompositions on a $\mathbb{Q}$-curve over the Mersenne prime. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 214–235. Springer, Heidelberg, November / December 2015.

[CLM+18] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: An efficient post-quantum commutative group action. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 395–427. Springer, Heidelberg, December 2018.

[CLN10] Craig Costello, Tanja Lange, and Michael Naehrig. Faster pairing computations on curves with high-degree twists. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010*, volume 6056 of *LNCS*, pages 224–242. Springer, Heidelberg, May 2010.

[CLN16] Craig Costello, Patrick Longa, and Michael Naehrig. Efficient algorithms for Supersingular Isogeny Diffie-Hellman. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016: 36th Annual International Cryptology Conference*, pages 572–601. Springer Berlin Heidelberg, 2016.

[CMO98] Henri Cohen, Atsuko Miyaji, and Takatoshi Ono. Efficient elliptic curve exponentiation using mixed coordinates. In Kazuo Ohta and Dingyi Pei, editors, *ASIACRYPT'98*, volume 1514 of *LNCS*, pages 51–65. Springer, Heidelberg, October 1998.

[CMR17] Sanjit Chatterjee, Alfred Menezes, and Francisco Rodríguez-Henríquez. On instantiating pairing-based protocols with elliptic curves of embedding degree one. *IEEE Trans. Computers*, 66(6):1061–1070, 2017.

[Coh10] Henri Cohen. *A Course in Computational Algebraic Number Theory*. Springer, 2010.

[Coh17] Bram Cohen. Proofs of space and time. Blockchain Protocol Analysis and Security Engineering, 2017.

[Cou06] Jean-Marc Couveignes. Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291, 2006. `http://eprint.iacr.org/2006/291`.

[Cox97] David A. Cox. *Primes of the form $x^2 + ny^2$: Fermat, class field theory, and complex multiplication*. Wiley, 1997.

[CP01] Clifford Cocks and Richard G.E. Pinch. Identity-based cryptosystems based on the Weil pairing. Unpublished manuscript, 2001.

[CP18] Bram Cohen and Krzysztof Pietrzak. Simple proofs of sequential work. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018*, pages 451–467, Cham, 2018. Springer International Publishing.

[CS18] Craig Costello and Benjamin Smith. Montgomery curves and their arithmetic - the case of large characteristic fields. *Journal of Cryptographic Engineering*, 8(3):227–240, 2018.

[DEM05] Régis Dupont, Andreas Enge, and François Morain. Building curves with arbitrary small MOV degree over finite prime fields. *Journal of Cryptology*, 18(2):79–89, April 2005.

[DF17] Luca De Feo. Mathematics of isogeny based cryptography. Lecture notes. École Mathématique Africaine. Thiès, Sénégal, 2017.

[DFHPS16] Luca De Feo, Cyril Hugounenq, Jérôme Plût, and Éric Schost. Explicit isogenies in quadratic time in any characteristic. *LMS Journal of Computation and Mathematics*, 19(A):267–282, 2016.

[DFJP14] Luca De Feo, David Jao, and Jérôme Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *Journal of Mathematical Cryptology*, 8(3):209–247, 2014.

[DG16] Christina Delfs and Steven D. Galbraith. Computing isogenies between supersingular elliptic curves over $\mathbb{F}_p$. *Designs, Codes and Cryptography*, 78(2):425–440, February 2016.

[DG19] Luca De Feo and Steven D. Galbraith. SeaSign: Compact isogeny signatures from class group actions. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 759–789. Springer, Heidelberg, May 2019.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DKS18] Luca De Feo, Jean Kieffer, and Benjamin Smith. Towards practical key exchange from ordinary isogeny graphs. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 365–394. Springer, Heidelberg, December 2018.

[DMPS19] Luca De Feo, Simon Masson, Christophe Petit, and Antonio Sanso. Verifiable delay functions from supersingular isogenies and pairings. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 248–277. Springer, Heidelberg, December 2019.

[DN93] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In Ernest F. Brickell, editor, *Advances in Cryptology — CRYPTO' 92*, pages 139–147, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[DÓSD06] Augusto Jun Devegili, Colm Ó hÉigeartaigh, Michael Scott, and Ricardo Dahab. Multiplication and squaring on pairing-friendly fields. Cryptology ePrint Archive, Report 2006/471, 2006. `http://eprint.iacr.org/2006/471`.

[DPB17] Javad Doliskani, Geovandro C. C. F. Pereira, and Paulo S. L. M. Barreto. Faster cryptographic hash function from supersingular isogeny graphs. Cryptology ePrint Archive, Report 2017/1202, 2017.

[Dra18] Justin Drake. Minimal VDF randomness beacon. Ethereum Research, 2018.

[EHL+18] Kirsten Eisenträger, Sean Hallgren, Kristin E. Lauter, Travis Morrison, and Christophe Petit. Supersingular isogeny graphs and endomorphism rings: Reductions and solutions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 329–368. Springer, Heidelberg, April / May 2018.

[EM14] Andreas Enge and Jérôme Milan. Implementing cryptographic pairings at standard security levels. In Rajat Subhra Chakraborty, Vashek Matyas, and Patrick Schaumont, editors, *Security, Privacy, and Applied Cryptography Engineering - 4th International Conference, SPACE 2014, Pune, India, October 18-22, 2014. Proceedings*, volume 8804 of *Lecture Notes in Computer Science*, pages 28–46. Springer, 2014.

[ES10] Andreas Enge and Andrew V. Sutherland. Class invariants by the CRT method. In Guillaume Hanrot, François Morain, and Emmanuel Thomé, editors, *Algorithmic Number Theory, 9th International Symposium, ANTS-IX, Nancy, France, July 19-23, 2010. Proceedings*, volume 6197 of *Lecture Notes in Computer Science*, pages 142–156. Springer, 2010.

[FGHT17] Joshua Fried, Pierrick Gaudry, Nadia Heninger, and Emmanuel Thomé. A kilobit hidden SNFS discrete logarithm computation. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 202–231. Springer, Heidelberg, April / May 2017.

[FK19] Georgios Fotiadis and Elisavet Konstantinou. TNFS resistant families of pairing-friendly elliptic curves. *Theoretical Computer Science*, 800:73–89, 31 December 2019. `https://ia.cr/2018/1017`.

[FKL+20] Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. Sqisign: compact post-quantum signatures from quaternions and isogenies. Cryptology ePrint Archive, Report 2020/1240, 2020. `https://eprint.iacr.org/2020/1240`.

[FKR12] Laura Fuentes-Castañeda, Edward Knapp, and Francisco Rodríguez-Henríquez. Faster hashing to $\mathbb{G}_2$. In Ali Miri and Serge Vaudenay, editors, *SAC 2011*, volume 7118 of *LNCS*, pages 412–430. Springer, Heidelberg, August 2012.

[FST10] David Freeman, Michael Scott, and Edlyn Teske. A taxonomy of pairing-friendly elliptic curves. *Journal of Cryptology*, 23(2):224–280, April 2010.

[Gal05] Steven Galbraith. Pairings. In Ian F. Blake, Gadiel Seroussi, and Nigel P. Smart, editors, *Advances in Elliptic Curve Cryptography*, London Mathematical Society Lecture Note Series, page 183–214. Cambridge University Press, 2005.

[GHS02] Steven D. Galbraith, Florian Hess, and Nigel P. Smart. Extending the GHS Weil descent attack. In *Advances in cryptology–EUROCRYPT 2002 (Amsterdam)*, volume 2332 of *Lecture Notes in Computer Science*, pages 29–44. Springer, Berlin, 2002.

[GHV08] Steven D. Galbraith, Florian Hess, and Frederik Vercauteren. Aspects of pairing inversion. *IEEE Transactions on Information Theory*, 54(12):5719–5728, 2008.

[GKZ14] Robert Granger, Thorsten Kleinjung, and Jens Zumbrägel. Breaking '128-bit secure' supersingular binary curves - (or how to solve discrete logarithms in $\mathbb{F}_{2^{4\cdot 1223}}$ and $\mathbb{F}_{2^{12\cdot 367}}$). In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 126–145. Springer, Heidelberg, August 2014.

[GLV01] Robert P. Gallant, Robert J. Lambert, and Scott A. Vanstone. Faster point multiplication on elliptic curves with efficient endomorphisms. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, pages 190–200. Springer, 2001.

[GM97] Robert M. Guralnick and Peter Müller. Exceptional polynomials of affine type. *Journal of Algebra*, 194(2):429–454, 1997.

[GMT20] Aurore Guillevic, Simon Masson, and Emmanuel Thomé. Cocks–pinch curves of embedding degrees five to eight and optimal ate pairing computation. *Designs, Codes and Cryptography*, 88:1047–1081, 03 2020.

[Gor93] Daniel M. Gordon. Designing and detecting trapdoors for discrete log cryptosystems. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 66–75. Springer, Heidelberg, August 1993.

[GPS17] Steven D. Galbraith, Christophe Petit, and Javier Silva. Identification protocols and signature schemes based on supersingular isogeny problems. In *Advances in Cryptology - ASIACRYPT 2017*, pages 3–33, 2017.

[GPS20] Steven D. Galbraith, Christophe Petit, and Javier Silva. Identification protocols and signature schemes based on supersingular isogeny problems. *Journal of Cryptology*, 33(1):130–175, January 2020.

[GPST16] Steven D. Galbraith, Christophe Petit, Barak Shani, and Yan Bo Ti. On the security of supersingular isogeny cryptosystems. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 63–91. Springer, Heidelberg, December 2016.

[GS10] Robert Granger and Michael Scott. Faster squaring in the cyclotomic subgroup of sixth degree extensions. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010*, volume 6056 of *LNCS*, pages 209–223. Springer, Heidelberg, May 2010.

[GS19] Aurore Guillevic and Shashank Singh. On the alpha value of polynomials in the tower number field sieve algorithm. Cryptology ePrint Archive, Report 2019/885, 2019. `https://eprint.iacr.org/2019/885`.

[Gt20] Torbjörn Granlund and the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*, 6.2.0 edition, 2020. `http://gmplib.org/`.

[Gui13] Aurore Guillevic. *Arithmetic of pairings on algebraic curves for cryptography.* thèse de doctorat, École Normale Supérieure, Paris, France, 2013. `https://tel.archives-ouvertes.fr/tel-00921940`.

[Har77] Robin Hartshorne. *Algebraic Geometry.* Springer, 1977. Graduate texts in mathematics, no. 52.

[HMV03] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography.* Springer, 2003.

[HSV06] Florian Hess, Nigel P. Smart, and Frederik Vercauteren. The eta pairing revisited. *IEEE Trans. Inf. Theor.*, 52(10):4595–4602, October 2006.

[JAC+19] David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev, David Urbanik, and Geovandro Pereira. SIKE. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

[JD11] David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In Bo-Yin Yang, editor, *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*, pages 19–34. Springer, Heidelberg, November / December 2011.

[JL03] Antoine Joux and Reynald Lercier. Improvements to the general number field sieve for discrete logarithms in prime fields. A comparison with the Gaussian integer method. *Math. Comput.*, 72(242):953–967, 2003.

[JLLRL18] David Jao, Jason LeGrow, Christopher Leonardi, and Luiz Ruiz-Lopez. A polynomial quantum space attack on CRS and CSIDH. In *MathCrypt 2018*, 2018. To appear.

[JLSV06] Antoine Joux, Reynald Lercier, Nigel Smart, and Frederik Vercauteren. The number field sieve in the medium prime case. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 326–344. Springer, Heidelberg, August 2006.

[JMVB09] David Y Jao, Peter L Montgomery, Ramarathnam Venkatesan, and Victor Boyko. Systems and methods for generation and validation of isogeny-based signatures, November 2009. US Patent 7,617,397.

[Jou04] Antoine Joux. A one round protocol for tripartite Diffie-Hellman. *Journal of Cryptology*, 17(4):263–276, September 2004.

[JP14] Antoine Joux and Cécile Pierrot. The special number field sieve in $\mathbb{F}_{p^n}$ - application to pairing-friendly constructions. In Zhenfu Cao and Fangguo Zhang, editors, *PAIRING 2013*, volume 8365 of *LNCS*, pages 45–61. Springer, Heidelberg, November 2014.

[JS10] David Jao and Vladimir Soukharev. A subexponential algorithm for evaluating large degree isogenies. In *ANTS IX: Proceedings of the Algorithmic Number Theory 9th International Symposium*, volume 6197 of *Lecture Notes in Computer Science*, pages 219–233, Berlin, Heidelberg, 2010. Springer.

[JV09] David Y Jao and Ramarathnam Venkatesan. Use of isogenies for design of cryptosystems, March 2009. US Patent 7,499,544.

[KB16] Taechan Kim and Razvan Barbulescu. Extended tower number field sieve: A new complexity for the medium prime case. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 543–571. Springer, Heidelberg, August 2016.

[Kit95] Alexey Yuri Kitaev. Quantum measurements and the abelian stabilizer problem. *arXiv preprint quant-ph/9511026*, 1995.

[KLPT14] David R. Kohel, Kristin Lauter, Christophe Petit, and Jean-Pierre Tignol. On the quaternion-isogeny path problem. *LMS Journal of Computation and Mathematics*, 17(A):418–432, 2014.

[Koh96] David Kohel. *Endomorphism rings of elliptic curves over finite fields*. PhD thesis, University of California at Berkley, 1996.

[KSS08] Ezekiel J. Kachisa, Edward F. Schaefer, and Michael Scott. Constructing Brezing-Weng pairing-friendly elliptic curves using elements in the cyclotomic field. In Steven D. Galbraith and Kenneth G. Paterson, editors, *PAIRING 2008*, volume 5209 of *LNCS*, pages 126–135. Springer, Heidelberg, September 2008.

[KT16] Takeshi Koshiba and Katsuyuki Takashima. Pairing cryptography meets isogeny: A new framework of isogenous pairing groups. Cryptology ePrint Archive, Report 2016/1138, 2016.

[KT19] Takeshi Koshiba and Katsuyuki Takashima. New assumptions on isogenous pairing groups with applications to attribute-based encryption. In Kwangsu Lee, editor, *Information Security and Cryptology – ICISC 2018*, pages 3–19, Cham, 2019. Springer International Publishing.

[Kup05] Greg Kuperberg. A subexponential-time quantum algorithm for the dihedral hidden subgroup problem. *SIAM Journal of Computing*, 35(1):170–188, 2005.

[Kup13] Greg Kuperberg. Another Subexponential-time Quantum Algorithm for the Dihedral Hidden Subgroup Problem. In Simone Severini and Fernando Brandao, editors, *8th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2013)*, volume 22 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20–34, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[KV10] Markus Kirschmer and John Voight. Algorithmic enumeration of ideal classes for quaternion orders. *SIAM Journal on Computing*, 39(5):1714–1747, 2010.

[KW19] Thorsten Kleinjung and Benjamin Wesolowski. Discrete logarithms in quasi-polynomial time in finite fields of fixed characteristic. Cryptology ePrint Archive, Report 2019/751, 2019. `https://eprint.iacr.org/2019/751`.

[LLJ93] Arjen K. Lenstra and Hendrik W. Lenstra Jr., editors. *The development of the number field sieve*, volume 1554 of *LNM*. Springer, 1993. `https://doi.org/10.1007/BFb0091534`.

[LS14] Patrick Longa and Francesco Sica. Four-dimensional Gallant-Lambert-Vanstone scalar multiplication. *Journal of Cryptology*, 27(2):248–283, April 2014.

[LW15] Arjen K. Lenstra and Benjamin Wesolowski. A random zoo: sloth, unicorn, and trx. *IACR Cryptology ePrint Archive*, 2015:366, 2015.

[Mes86] Jean-François Mestre. La méthode des graphes. Exemples et applications. In *Proceedings of the international conference on class numbers and fundamental units of algebraic number fields (Katata, 1986)*, Nagoya, 1986. Nagoya University.

[MMV13] Mohammad Mahmoody, Tal Moran, and Salil Vadhan. Publicly verifiable proofs of sequential work. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pages 373–388. ACM, 2013.

[Mon05] Peter L. Montgomery. Five, six, and seven-term Karatsuba-like formulae. *IEEE Transactions on Computers*, 54:362–369, March 2005.

[MRV99] Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, pages 120–130, Oct 1999.

[MVO91] Alfred Menezes, Scott Vanstone, and Tatsuaki Okamoto. Reducing elliptic curve logarithms to logarithms in a finite field. In *STOC '91: Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 80–89, New York, NY, USA, 1991. ACM.

[NV10] Phong Q. Nguyen and Brigitte Vallée, editors. *The LLL Algorithm - Survey and Applications*. ISC. Springer, Heidelberg, 2010.

[Pie18] Krzysztof Pietrzak. Simple verifiable delay functions. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*, volume 124 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 60:1–60:15, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[Piz80] Arnold Pizer. An algorithm for computing modular forms on $\gamma_0(n)^*$. *Journal of Algebra 64*, pages 340–390, 1980.

[Piz90] Arnold K. Pizer. Ramanujan graphs and Hecke operators. *Bulletin of the American Mathematical Society (N.S.)*, 23(1), 1990.

[Piz98] Arnold K. Pizer. Ramanujan graphs. In *Computational perspectives on number theory (Chicago, IL, 1995)*, volume 7 of *AMS/IP Stud. Adv. Math.* Amer. Math. Soc., Providence, RI, 1998.

[PL17] Christophe Petit and Kristin Lauter. Hard and easy problems for supersingular isogeny graphs. Cryptology ePrint Archive, Report 2017/962, 2017.

[PSNB11] Geovandro C.C.F. Pereira, Marcos A. Simplício, Michael Naehrig, and Paulo S.L.M. Barreto. A family of implementation-friendly BN elliptic curves. *Journal of Systems and Software*, 84(8):1319 – 1326, 2011.

[PW18] Cécile Pierrot and Benjamin Wesolowski. Malleability of the blockchain's entropy. *Cryptography and Communications*, 10(1):211–233, Jan 2018.

[Rab83] Michael O. Rabin. Transaction protection by beacons. *Journal of Computer and System Sciences*, 27(2):256–267, 1983.

[Reg04] Oded Regev. A subexponential time algorithm for the dihedral hidden subgroup problem with polynomial space. arXiv:quant-ph/0406151, June 2004.

[Ren18] Joost Renes. Computing isogenies between Montgomery curves using the action of (0, 0). In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018*, pages 229–247. Springer, Heidelberg, 2018.

[RS06] Alexander Rostovtsev and Anton Stolbunov. Public-Key Cryptosystem Based On Isogenies. Cryptology ePrint Archive, Report 2006/145, 2006. `http://eprint.iacr.org/2006/145`.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978.

[RSW96] Ronald L. Rivest, Adi Shamir, and David A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1996.

[Sag18] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 8.0)*, 2018.

[Sch93] Oliver Schirokauer. Discrete logarithms and local units. *Philos. Trans. Roy. Soc. London Ser. A*, 345(1676):409–423, 1993. `http://rsta.royalsocietypublishing.org/content/345/1676/409`, `http://doi.org/10.1098/rsta.1993.0139`.

[Sch95] René Schoof. Counting points on elliptic curves over finite fields. *Journal de théorie des nombres de Bordeaux*, 7(1):219–254, 1995.

[Sil86] Joseph H. Silverman. *The arithmetic of elliptic curves*, volume 106 of *Graduate texts in mathematics*. Springer, 1986.

[SJK+17] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris-Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *IEEE Symposium on Security and Privacy*, pages 444–460. IEEE Computer Society, 2017.

[Smi16] Benjamin Smith. The $\mathbb{Q}$-curve construction for endomorphism-accelerated elliptic curves. *Journal of cryptology*, 29(4):806–832, October 2016.

[ST15] Joseph H. Silverman and John T. Tate. *Rational Points on Elliptic Curves*. Springer, 2nd edition, 2015.

[Sut10] Andrew V. Sutherland. Computing hilbert class polynomials with the chinese remainder theorem. *Mathematics of Computation*, 80(273):501–538, May 2010.

[Tan09] Seiichiro Tani. Claw finding algorithms using quantum walk. *Theoretical Computer Science*, 410(50):5285–5297, 2009.

[Tea17] The CADO-NFS Development Team. CADO-NFS, an implementation of the number field sieve algorithm, 2017. Release 2.3.0.

[Vél71] Jacques Vélu. Isogénies entre courbes elliptiques. *Comptes Rendus de l'Académie des Sciences de Paris*, 273:238–241, 1971.

[Ver10] Frederik Vercauteren. Optimal pairings. *IEEE Trans. Inf. Theory*, 56(1):455–461, 2010.

[Voi20] John Voight. Quaternion algebras, 2020. Draft available at `https://math.dartmouth.edu/~jvoight/quat.html`.

[vW99] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28, January 1999.

[Wes19] Benjamin Wesolowski. Efficient verifiable delay functions. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 379–407, Cham, 2019. Springer International Publishing.

[WS07] Claire Whelan and Michael Scott. The importance of the final exponentiation in pairings when considering fault attacks. In Tsuyoshi Takagi, Tatsuaki Okamoto, Eiji Okamoto, and Takeshi Okamoto, editors, *PAIRING 2007*, volume 4575 of *LNCS*, pages 225–246. Springer, Heidelberg, July 2007.

[YAJ+17] Youngho Yoo, Reza Azarderakhsh, Amir Jalali, David Jao, and Vladimir Soukharev. A post-quantum digital signature scheme based on supersingular isogenies. In Aggelos Kiayias, editor, *Financial Cryptography and Data Security*, pages 163–181, Cham, 2017. Springer International Publishing.

[ZL12] Xusheng Zhang and Dongdai Lin. Analysis of optimum pairing products at high security levels. In Steven D. Galbraith and Mridul Nandi, editors, *INDOCRYPT 2012*, volume 7668 of *LNCS*, pages 412–430. Springer, Heidelberg, December 2012.

Résumé

Cette thèse étudie l'algorithmie de plusieurs applications cryptographiques liées aux courbes elliptiques et aux isogénies de courbes elliptiques. D'une part, nous étudions le compromis entre efficacité et sécurité concernant les courbes à couplages pour un niveau de sécurité de 128 bits. La menace des récentes avancées sur le calcul de logarithme discret dans certains corps finis nous oriente vers l'étude de nouvelles courbes à couplage. Nous effectuons une comparaison de l'efficacité de ces nouvelles courbes avec celles utilisées actuellement en estimant le temps de calcul pratique. D'autre part, nous présentons la cryptographie à base d'isogénies de courbes supersingulières, considérées actuellement comme résistantes aux ordinateurs quantiques. Nous portons une attention particulière à la sécurité de ces protocoles en apportant une implémentation des calculs d'idéaux connectants entre ordres maximaux d'algèbres de quaternions. Enfin, nous présentons deux constructions de fonctions à délai vérifiables, basées sur des calculs de couplages et d'évaluations d'isogénies de grand degré friable. Ces dernières ne sont pas considérées comme résistantes aux ordinateurs quantiques, mais apportent plusieurs nouveautés par rapport aux constructions actuelles. Nous analysons leur sécurité et effectuons une comparaison entre toutes ces fonctions à un niveau de sécurité de 128 bits.

Mots-clés: courbes elliptiques, isogénies, algèbre de quaternions, couplage, fonction à délai vérifiable.

Abstract

This thesis studies the algorithmic of several cryptographic applications related to elliptic curves and isogenies of elliptic curves. On the one hand, we study the tradeoff between efficiency and security in pairing-based cryptography at the 128-bit security level. The threat of the recent improvements on the discrete logarithm computation over specific finite fields lead us to study new pairing-friendly curves. We give a comparison of efficiency between our new curves and the state-of-the-art curves by estimating the measurement in practice. On the other and, we present isogeny-based cryptography, considered to be post-quantum resistant. We look at a concrete implementation of cryptanalysis based on connecting ideals between maximal orders of quaternion algebras. Finally, we present two constructions of verifiable delay functions based on computations of pairings and isogenies of large smooth degree. These functions are not considered to be post-quantum resistant, but bring several new properties compared to the current constructions. We analyse their security and give a comparison of all the known functions at the 128-bit security level.

Keywords: elliptic curves, isogenies, quaternion algebra, pairing, verifiable delay function.