Comparaison expérimentale d'architectures de crypto-processeurs pour courbes elliptiques et hyper-elliptiques

Gabriel Gallin, Arnaud Tisserand, Nicolas Veyrat-Charvillon

Compas'2015 : Parallélisme/ Architecture / Système
Lille, France, du 30 juin au 3 juillet 2015

Comparaison expérimentale d'architectures de crypto-processeurs pour courbes elliptiques et hyper-elliptiques.

Gabriel Gallin[2,1], Arnaud Tisserand[2,1] et Nicolas Veyrat-Charvillon[3,1]

[1]IRISA, [2]CNRS – [3]Université Rennes 1 – INRIA. 6 rue de Kerampont, 22305 Lannion.

Résumé
Dans ce papier, nous présentons des implantations sur FPGA de différentes configurations des unités de calcul d'un crypto-processeur pour courbes elliptiques et hyper-elliptiques. Nous comparons expérimentalement les performances et coûts relatifs de primitives classiques pour ces deux crypto-systèmes avec un même niveau de sécurité théorique. Nos résultats expérimentaux montrent qu'HECC est environ 40 % plus rapide qu'ECC pour un même coût en surface et à niveau de sécurité théorique équivalent.

Mots-clés : ECC, HECC, crypto-processeur, sécurité matérielle, exploration d'architecture, circuit FPGA.

---

1. Introduction

Pour assurer la sécurité de nombreuses applications, il est nécessaire d'utiliser des primitives de *cryptographie asymétrique*, ou cryptographie à clé publique, dans les protocoles comme l'échange de clé secrète, la signature numérique ou certains chiffrements spécifiques. La *cryptographie sur courbes elliptiques* (ECC en anglais, cf. [6]) est devenue le standard en cryptographie asymétrique dans de nombreux pays. Elle supplante RSA du fait de ses bien meilleures performances et de son moindre coût. Par exemple, pour un même niveau de sécurité théorique, RSA 2048 bits correspond à ECC 224 bits. Une évolution vers la *cryptographie sur courbes hyper-elliptiques* (HECC) est actuellement étudiée pour des futures générations de crypto-systèmes asymétriques [4]. HECC offre un même niveau de sécurité théorique qu'ECC pour des clés et des éléments de corps finis deux fois plus petits mais au prix de calculs intermédiaires plus nombreux. Afin de comparer objectivement ECC et HECC, il convient de les implanter, les optimiser et les analyser dans un même cadre expérimental.

Notre groupe de recherche étudie et développe des opérateurs arithmétiques, des accélérateurs matériels et des protections pour ECC depuis de nombreuses années et pour HECC depuis peu. À moyen terme, nous souhaitons évaluer les compromis possibles entre le temps de calcul, le coût de mise en œuvre (surface de silicium et consommation d'énergie) et la robustesse face à certaines attaques physiques (en particulier par analyse de canaux cachés cf. [11]).

Dans ce papier, nous comparons les performances relatives, sur FPGA, de solutions ECC et HECC pour un même niveau de sécurité théorique. Nous évaluons les surfaces et temps de calcul d'un grand nombre de configurations des unités arithmétiques.

2. État de l'art

Des présentations très complètes d'ECC et HECC sont disponibles dans les livres de référence [6] et [4]. Les primitives (H)ECC font appel à trois niveaux d'opérations résumés ci-dessous.

Au niveau le plus bas, on trouve les opérations de base dans le *corps finis* : addition/soustraction (A), multiplication (M), carré (S) et inversion (I). Dans ce papier, nous utilisons le corps $\mathbb{F}_p$ où p est un grand premier quelconque de ℓ bits. En pratique, $\ell \in \{100, \ldots, 600\}$ selon le crypto-système et le niveau de sécurité considérés. Notre crypto-processeur supporte aussi $\mathbb{F}_{2^m}$ ou $\mathbb{F}_p$ pour des p particuliers comme les (pseudo)-Mersennes préconisés dans [13]. Pour des raisons de place, nous n'utilisons ici que $\mathbb{F}_p$ avec p quelconque (c.-à-d. sans écriture particulière). Nous utilisons $\mathbb{F}_p$ car c'est, a priori, le corps qui conduit à la plus grande robustesse théorique.

Au niveau intermédiaire, on trouve les opérations sur les *points* de la courbe : *addition* de points, notée $\texttt{ADD}(\mathbf{P}, \mathbf{Q}) = \mathbf{P}+\mathbf{Q}$, et *doublement* de point, noté $\texttt{DBL}(\mathbf{P}) = \mathbf{P}+\mathbf{P}$, où les coordonnées des points **P** et **Q** sont des éléments du corps. Différents systèmes de coordonnées ont été proposés, nous utilisons les coordonnées projectives du fait de leurs bonnes performances [6, Sec. 3.2]. HECC utilise des représentations particulières des points, nous employons les coordonnées projectives de la représentation de Mumford [4, Sec. 14.1.2]. En pratique, ADD et DBL sont des séquences d'opérations sur les coordonnées ou les représentations des points, et donc d'opérations dans le corps fini. On trouve sur le site de référence [1] les descriptions des séquences de calcul pour différentes courbes et représentations des points.

Enfin au niveau le plus haut, on trouve la *multiplication scalaire* notée $[k]\mathbf{P}$ avec k un grand entier appelé *scalaire* (c.-à-d. la *clé* secrète ou publique selon l'utilisation de la primitive) et **P** un point de la courbe. Un algorithme de $[k]\mathbf{P}$ est essentiellement une boucle qui parcourt tous les chiffres de k et effectue des ADD et/ou DBL en fonction de la valeur de chaque chiffre k_i. Plusieurs techniques de recodage de clés ont été proposées pour réduire le nombre d'opérations au niveau courbe (cf. [6, Sec. 3.3] et [3]). Certaines de ces techniques de recodage permettent d'accélérer encore plus la multiplication scalaire au prix de quelques pré-calculs.

L'intérêt principal d'HECC par rapport à ECC est d'offrir, pour un même niveau de sécurité théorique, des tailles de corps et de scalaires *deux fois plus petits* (c.-à-d. $\ell_{\text{HECC}} \approx \frac{1}{2}\ell_{\text{ECC}}$, cf. [4]). Mais HECC requiert des séquences d'opérations bien plus complexes pour les opérations ADD et DBL que celles d'ECC. Ainsi pour HECC, la boucle de multiplication scalaire nécessite environ moitié moins d'itérations, mais chaque itération nécessite beaucoup plus d'opérations sur le corps elles-mêmes moins coûteuses que pour ECC.

Pour ce travail, nous avons choisi des paramètres pour (H)ECC et des techniques communément employés dans la littérature et résumés à la table 1. Pour ECC, nous avons sélectionné $\ell_{\text{ECC}} = 256$ bits selon les préconisations actuelles comme bon niveau de sécurité à moyen terme. La comparaison est donc faite avec $\ell_{\text{HECC}} = 128$ bits pour HECC. Les séquences choisies pour ADD et DBL dans HECC sont tirées de [9] et fréquemment utilisées dans la littérature. Les algorithmes de $[k]\mathbf{P}$ considérés, pour ECC et HECC, sont ceux qui parcourent k de gauche à droite avec les recodages suivants : binaire standard (BIN), NAF pour *non-adjacent form* (cf. [6, Sec. 3.3, p. 98]) et méthodes à fenêtre de type *w*NAF avec $w \in \{3, 4\}$ (cf. [6, Sec. 3.3, p. 100]).

Jusqu'ici, très peu d'implantations matérielles ont été publiées pour HECC et elles portent essentiellement sur $\mathbb{F}_{2^m}$ [7, 2, 14]. À notre connaissance, pour HECC sur $\mathbb{F}_p$, on trouve uniquement des implantations logicielles comme [8].

	taille $\mathbb{F}_p$	ADD	DBL	source
ECC	$\ell_{ECC} = 256$ bits	$12\mathtt{M} + 2\mathtt{S}$	$7\mathtt{M} + 3\mathtt{S}$	[1]
HECC	$\ell_{HECC} = 128$ bits	$40\mathtt{M} + 4\mathtt{S}$	$38\mathtt{M} + 6\mathtt{S}$	[9]

TABLE 1 – Paramètres et coûts, en nombre d'opérations, considérés pour cette étude.

3. Architecture et flot logiciel développés

3.1. Architecture de notre crypto-processeur ECC/HECC

Notre crypto-processeur est représenté en figure 1 et se compose de plusieurs blocs : les *unités arithmétiques* (UA) en charge des opérations au niveau corps fini ; la *mémoire des points* ; l'*unité de recodage* des scalaires ; la *mémoire de programme* ; le *bloc de contrôle global* (CTRL) en charge du fonctionnement du processeur ; et enfin l'*interface d'entrée/sortie* non représentée sur la figure 1. Les opérations aux niveaux courbe (ADD et DBL) et multiplication scalaire ([k]**P**) sont réalisées en logiciel à l'aide du code exécuté par le processeur.

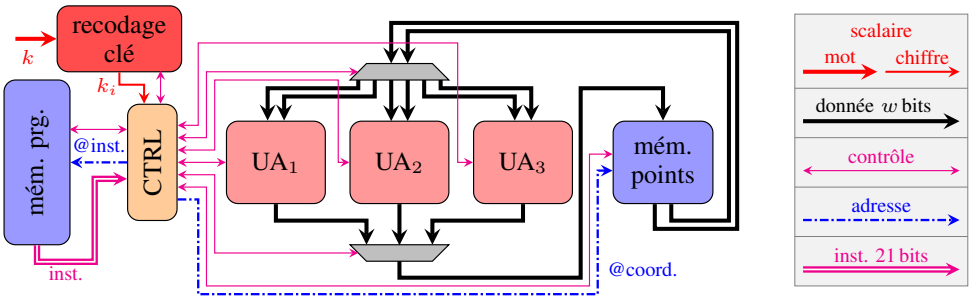


FIGURE 1 – Architecture globale de notre crypto-processeur pour une configuration avec 3 unités arithmétiques (l'interface d'entrée/sortie externe n'est pas représentée).

Les UA sont optimisées pour l'addition/soustraction, la multiplication et l'inversion sur $\mathbb{F}_p$. Plusieurs tailles de multiplieurs, plus ou moins rapides, sont disponibles, voir la section 3.2. L'ajout d'une UA dédiée au carré sera étudiée prochainement (nous supposons donc $1\mathtt{M} = 1\mathtt{S}$ dans ce papier). Chaque unité est autonome. Une fois les opérandes chargés, le calcul se déroule en utilisant les registres internes de l'UA. L'UA signale la fin du calcul au contrôleur global pour récupération du résultat.

La mémoire des points (à droite de la figure 1 et réalisée en BRAM dans le FPGA) stocke les coordonnées ou représentations des points. Les valeurs stockées sont des éléments de $\mathbb{F}_p$.

L'unité de recodage (en haut à gauche de la figure 1) stocke les clés [1] et implante l'algorithme de recodage sélectionné avant implantation. Elle parcourt les chiffres de k et transmet au contrôleur chaque nouveau chiffre recodé k_i pour sélection de la bonne opération ADD ou DBL.

La mémoire de programme, réalisée en BRAM dans le FPGA, et le contrôleur global s'occupent du stockage, du décodage et de l'exécution des instructions du programme. Le jeu d'instructions et le fonctionnement interne du crypto-processeur ont été conçus pour les programmes rencontrés dans les applications. Le temps de calcul des UA pouvant être important et variable,

1. Sans lecture externe possible pour des raisons de sécurité.

il y a une instruction pour le lancement du calcul dans une UA donnée et une autre pour l'attente du résultat. L'instruction de lancement n'est pas bloquante (c.-à-d. $\texttt{PC} \leftarrow \texttt{PC} + 1$ à chaque cycle) pour supporter des exécutions parallèles. Mais celle d'attente du résultat est bloquante pour garantir le respect des dépendances de données.
L'interface d'entrée/sortie permet de charger le programme, le scalaire k, le point $\mathbf{P}$ et les paramètres de la courbe. En fin de calcul, elle permet de récupérer le résultat $[k]\mathbf{P}$. Toutes les tâches de l'interface ont des durées extrêmement courtes par rapport à celles des calculs.
Nous avons défini plusieurs objectifs pour la conception du crypto-processeur. Nous souhaitons avoir une architecture flexible qui s'adapte à différents besoins applicatifs et à différents FPGA (en particulier des petits circuits à bas coût). Pour cela, les données internes sont découpées en mots de w bits, avec $w \leq \ell$, afin d'obtenir de bonnes fréquences de fonctionnement et de faibles surfaces. Dans ce papier, nous utilisons $w = 32$ bits sur un petit Spartan 6 LX75, et cela permet d'obtenir des fréquences un peu au-dessus de 200 MHz. Une exploration sur l'impact de w sera conduite dans un futur proche. Le nombre et le type des UA sont paramétrables. Ceci permet d'adapter les performances et le degré de parallélisme interne en fonction des besoins applicatifs. Un ensemble de paramètres forme une configuration, sous la forme d'un ensemble de fichiers VHDL, qui peut être synthétisée et implantée sur le circuit. Le type d'algorithme de recodage de clé est aussi paramétrable. Certaines contraintes existent entre les différents paramètres pour qu'une configuration soit cohérente.

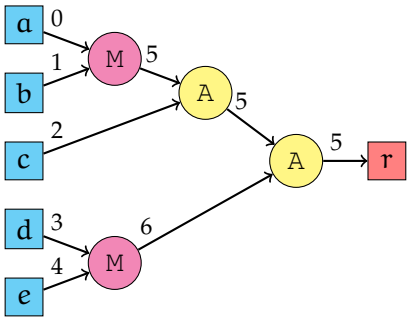
3.2. Multiplieurs sur $\mathbb{F}_p$

Le produit de 2 nombres de ℓ bits (avec $\ell \in \{128, 256\}$) modulo un premier de ℓ bits est une opération coûteuse. Chaque multiplication peut nécessiter un grand nombre de cycles d'horloge sur un petit multiplieur. Par contre, dans les applications (H)ECC, un seul additionneur/soustracteur (naturellement très petit et très rapide) et une seule UA d'inversion (très peu utilisée) sont nécessaires dans l'architecture.
Nos multiplieurs utilisent l'algorithme de Montgomery [12] et sont optimisés pour deux paramètres (fixés statiquement avant implantation) : le nombre de multiplieurs n_M présents dans l'architecture et le nombre de sous-blocs parallèles n_B dans un seul multiplieur. Le paramètre n_M permet d'augmenter les performances de calcul et de s'adapter aux possibilités de parallélisme dans les séquences `ADD` et `DBL`. Le paramètre n_B est le nombre de sous-blocs de w bits actifs dans un multiplieur. Pour $n_B = 1$ la surface est très petite mais le nombre de cycles maximal. Pour des valeurs de n_B supérieures, la vitesse de multiplication augmente mais la surface aussi. En explorant différentes valeurs pour n_M et n_B, nous essayons d'exploiter au mieux les possibilités de parallélisme dans les séquences de calcul au niveau courbe pour un budget surface donné.
En pratique, nos multiplieurs ont 3 étages de pipeline interne et on utilisera n_B dans $\{1, 2, 4\}$ pour ECC et $\{1, 2\}$ pour HECC. Les multiplieurs pour des valeurs comme $n_B = 3$ demandent un contrôle complexe et ne sont pas très efficaces en pratique pour $\ell \in \{128, 256\}$. Pour HECC, utiliser $n_B = 4$ n'est pas efficace car la taille du corps étant petite ($\ell = 128$), avoir $n_B = 4$ sous-blocs de $w = 32$ bits avec un pipeline interne à 3 étages conduit à un trop faible taux d'utilisation de ces sous-blocs.
Pour le moment, nous n'utilisons pas les blocs DSP des FPGA, mais nous travaillons sur ce point pour une prochaine version.

3.3. Programmation de notre crypto-processeur

Actuellement la programmation du crypto-processeur se fait au niveau assembleur. L'utilisateur décrit le code assembleur à exécuter, puis celui-ci est traduit en binaire par le programme

$r = ((a \times b) + c) + (d \times e))$




```
1   read fu_mul_0, 0, 1          lit opérandes a et b
2   launch fu_mul_0              lance ab
3   read fu_mul_1, 3, 4          lit opérandes d et e
4   launch fu_mul_1              lance de
5   wait fu_mul_0                attend la fin de ab
6   write fu_mul_0, 5            écrit le résultat de ab
7   set OPMODE, 0                passe en mode addition (+)
8   read fu_add_sub_0, 5, 2      lit ab et opérande c
9   launch fu_add_sub_0          lance (ab) + c
10  wait fu_mul_1                attend la fin de de
11  write fu_mul_1, 6            écrit le résultat de de
12  wait fu_add_sub_0            attend la fin de (ab) + c
13  write fu_add_sub_0, 5        écrit le résultat de (ab) + c
14  read fu_add_sub_0, 5, 6      lit (ab) + c et de
15  launch fu_add_sub_0          lance ((ab) + c) + (de)
16  wait fu_add_sub_0            attend la fin de ((ab) + c) + (de)
17  write fu_add_sub_0, 5        écrit ((ab) + c) + (de)
```

TABLE 2 – Exemple de séquence d'opérations sur $\mathbb{F}_p$, son graphe de dépendances, et le code assembleur correspondant pour $n_M = 2$.

d'assemblage que nous avons développé. En pratique, nous avons des scripts qui permettent de faire l'ordonnancement des opérations dans les séquences de calcul au niveau courbe à partir de leur description mathématique. La gestion de la boucle sur les chiffres de k dans la multiplication scalaire $[k]\mathbf{P}$ se fait en utilisant des instructions spécifiques liées à l'unité de recodage de clé. Une instruction attend un nouveau chiffre k_i fournit par l'unité de recodage. Puis pour chaque chiffre recodé, on effectue une comparaison et le branchement à la fonction réalisant la séquence `ADD` ou `DBL` correspondante au nouveau chiffre k_i. À la fin du parcours du scalaire, une interruption permet de sortir de la boucle.

La table 2 présente un petit exemple de calcul avec $n_M = 2$ multiplieurs. En ligne 7, l'instruction `set OPMODE` force l'UA d'addition/soustraction soit en mode addition (0) soit en mode soustraction (1). Avec un seul additionneur dans l'architecture, il ne faut pas oublier de bien positionner ce mode avant chaque nouveau type d'opération. Du fait du système actuel de chargement des opérandes dans les UA, nous pouvons avoir quelques transferts redondants vers/depuis la mémoire des points. Leur impact sur la vitesse de calcul est extrêmement faible en pratique ($\ll 1\,\%$), mais il conviendra d'étudier cela pour des aspects de robustesse aux attaques physiques. Pour les éviter, il faudra très probablement modifier un peu le jeu d'instruction et le contrôle.

Nous travaillons sur le développement d'un petit compilateur spécialisé qui permettra de programmer le crypto-processeur à partir du langage Python dans le logiciel mathématique Sage.

4. Étude expérimentale réalisée

Afin de pouvoir comparer objectivement les performances et coûts relatifs de ECC et HECC, nous les avons implantés en utilisant différentes configurations de notre crypto-processeur sur le même FPGA, un Spartan 6 LX75 Xilinx, avec un effort d'optimisation identique. L'implantation a été faite avec l'environnement ISE 14.6 de Xilinx, où chaque configuration du crypto-processeur a été synthétisée, placée–routée, et enfin validée par des simulations intensives.

Afin de mesurer les temps de calcul, nous avons simulé dans Modelsim, au niveau « cycle près bit près », des multiplications scalaires complètes pour des clés tirées au hasard. Dans chaque cas, les résultats retournés ont été validés numériquement par Sage.

Les paramètres d'exploration des différentes configurations sont le nombre de multiplieurs n_M

recodage	BIN	NAF	3NAF	4NAF
surface, slices (FF/LUT)	517 (1347/1433)	536 (1366/1445)	560 (1370/1454)	547 (1374/1460)
fréquence, MHz	229	234	235	231

TABLE 3 – Impact de l'unité de recodage sur le coût matériel et la fréquence du crypto-processeur complet (pour ECC, les variations des résultats sont comparables pour HECC).

disponibles et leur largeur interne n_B (cf. section 3.2).

4.1. Résultats d'implantation sur FPGA

Dans un premier temps, nous avons évalué l'impact du type d'unité de recodage de clé dans le crypto-processeur complet. La table 3 présente les résultats de synthèse obtenus pour les différentes unités de recodage testées avec ECC. Ces unités utilisent une faible surface, même pour la plus complexe 4NAF, et ne changent quasiment pas la fréquence de fonctionnement. Les résultats pour HECC sont parfaitement similaires. Dans la suite, nous rapporterons tous les résultats en utilisant l'unité de recodage 4NAF qui donne les meilleures performances.
Dans un deuxième temps, nous avons comparé le coût matériel, en surface, pour différentes configurations du nombre de multiplieurs n_M et de leur taille n_B pour ECC et pour HECC. La table 4 présente ces résultats. On constate que les paramètres (n_M, n_B) influent directement sur la surface du crypto-processeur, mais n'ont qu'un impact limité sur sa fréquence de fonctionnement. En effet, il a été conçu de manière à minimiser la perte de fréquence lorsque l'on ajoute des UA ou lorsqu'on utilise des multiplieurs plus rapides (et donc plus gros).

	n_M	BRAM	$n_B = 1$		$n_B = 2$		$n_B = 4$	
			surface slices (FF/LUT)	fréq. MHz	surface slices (FF/LUT)	fréq. MHz	surface slices (FF/LUT)	fréq. MHz
ECC	1	3	547 (1374/1460)	231	573 (1476/1625)	233	673 (1674/1875)	233
	2	3	722 (1776/1903)	220	811 (1979/2210)	227	942 (2377/2701)	220
	3	3	810 (2174/2236)	221	915 (2480/2698)	215	1130 (3077/3430)	214
	4	3	952 (2569/2656)	215	1100 (2977/3282)	217	1512 (3771/4293)	216
	5	3	1064 (2982/3136)	210	1405 (3492/3902)	206	1722 (4487/5122)	209
HECC	1	4	514 (1336/1374)	235	549 (1434/1513)	234		
	2	4	646 (1716/1783)	220	737 (1912/2055)	234		
	3	4	732 (2092/2075)	224	826 (2386/2485)	225		
	4	4	870 (2476/2424)	218	1022 (2868/2987)	214		
	5	4	976 (2865/2773)	219	1115 (3355/3465)	210		
	6	4	1089 (3233/3092)	203	1240 (3821/3908)	208		
	7	4	1145 (3601/3426)	213	1372 (4287/4365)	205		
	8	4	1281 (3981/3809)	191	1552 (4765/4890)	183		
	9	4	1379 (4363/4051)	202	1691 (5245/5277)	199		
	10	4	1543 (4739/4435)	196	1856 (5719/5801)	198		
	11	4	1547 (5114/4750)	189	1936 (6192/6240)	198		
	12	4	1738 (5499/5128)	191	2100 (6675/6771)	188		

TABLE 4 – Résultats d'implantation du crypto-processeur pour ECC (256 bits) et HECC (128 bits) selon différentes configurations (n_M, n_B) sur un FPGA Spartan 6 LX75.

Dans un troisième temps, nous avons mesuré les temps de calcul pour les configurations implantées ci-dessus. La table 5 fournit les temps moyens de 50 multiplications scalaires complètes avec des k tirés au hasard. La variance observée sur le temps de calcul est très faible comme illustré dans la table 6.

	n_B	n_M											
		1	2	3	4	5	6	7	8	9	10	11	12
HECC	1	15.6	8.6	5.7	4.7	3.9	3.7	3.3	3.6	3.4	3.5	3.6	3.6
	2	11.9	6.2	4.5	3.6	3.2	2.8	2.8	3.0	2.7	2.7	2.8	2.9
ECC	1	28.1	15.3	12.4	12.4	12.7							
	2	17.7	9.6	8.3	8.0	8.4							
	4	11.1	6.2	5.4	5.1	5.3							

TABLE 5 – Temps de calcul moyen, mesuré en ms, pour des $[k]\mathbf{P}$ avec des k aléatoires sur ECC (256 bits) et HECC (128 bits) selon différentes configurations (n_M, n_B) de l'architecture.

configuration	ECC (1,1)	ECC (3,4)	HECC (1,1)	HECC (6,2)
temps moyen [ms]	28.2	5.3	15.5	2.8
écart type [ms]	0.289	0.056	0.324	0.045

TABLE 6 – Statistiques du temps de calcul pour $[k]\mathbf{P}$ dans quelques configurations (n_M, n_B).

4.2. Comparaisons

La figure 2 permet de comparer facilement les solutions explorées pour ECC (256 bits) et HECC (128 bits). Chaque point (rond ou carré) représente une configuration définie par le couple de paramètres (n_M, n_B). Les lignes en pointillé précisent les meilleurs compromis disponibles.

Notre architecture, du fait de sa flexibilité, nous permet de proposer un large éventail de configurations, et peut s'adapter aux besoins des applications. Une vaste exploration des paramètres nous a permis de déterminer les configurations les plus intéressantes sur le FPGA utilisé.

En étudiant le rapport, à surface équivalente, des temps de calcul pour ECC et HECC, on constate que ce rapport est proche de 60 %.

La figure 3 illustre, pour les configurations les plus efficaces observées, d'une part le taux d'utilisation moyen des multiplieurs, d'autre part le surcoût matériel et l'accélération obtenus par rapport à la solution HECC utilisant un seul multiplieur sur un bloc ($n_M = n_B = 1$). On constate que le gain en vitesse va de concert avec le surcoût matériel tant que les multiplieurs peuvent être utilisés en parallèle. Cette aptitude dépend directement des séquences de calcul au niveau courbe (`ADD` et `DBL`). Dans le cas d'ECC, le gain obtenu diminue dès 3 multiplieurs, alors que les séquences d'HECC permettent une parallélisation des calculs efficace au moins jusqu'à 5 multiplieurs. Au delà, le fait d'ajouter des multiplieurs ne permet qu'un faible gain en vitesse par rapport au surcoût en surface.

En l'absence de résultats d'implantations matérielles pour HECC sur $\mathbb{F}_p$ dans la littérature, nous avons limité notre comparaison avec d'autres références au cas d'ECC. La table 7 présente

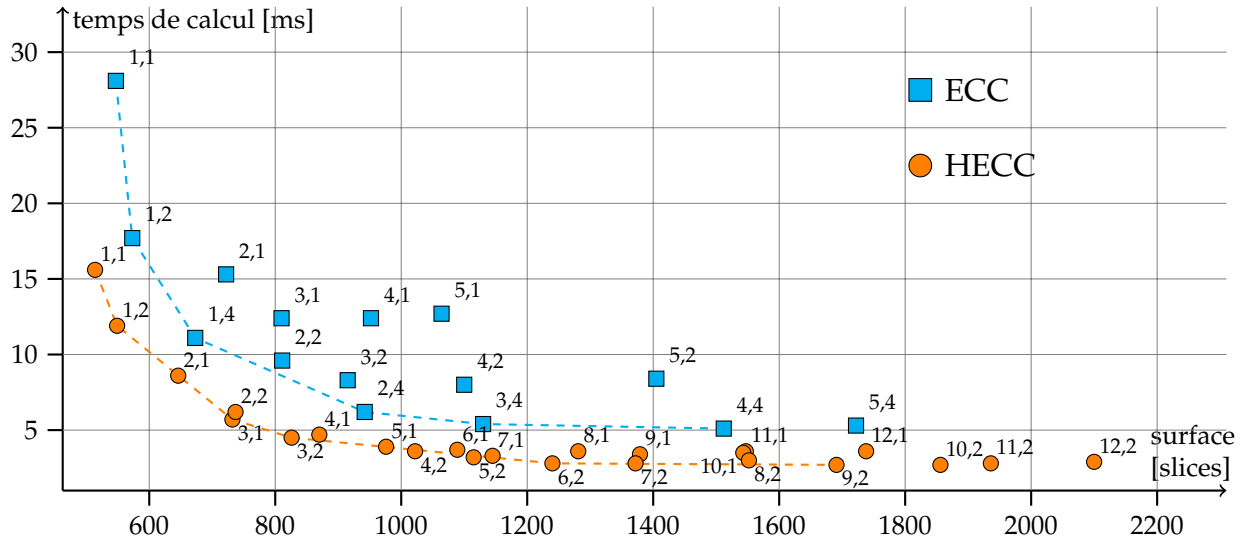


FIGURE 2 – Compromis temps–surface pour [k]**P** sur ECC (256 bits) et HECC (128 bits) pour différentes configurations (n_M, n_B) de l'architecture.

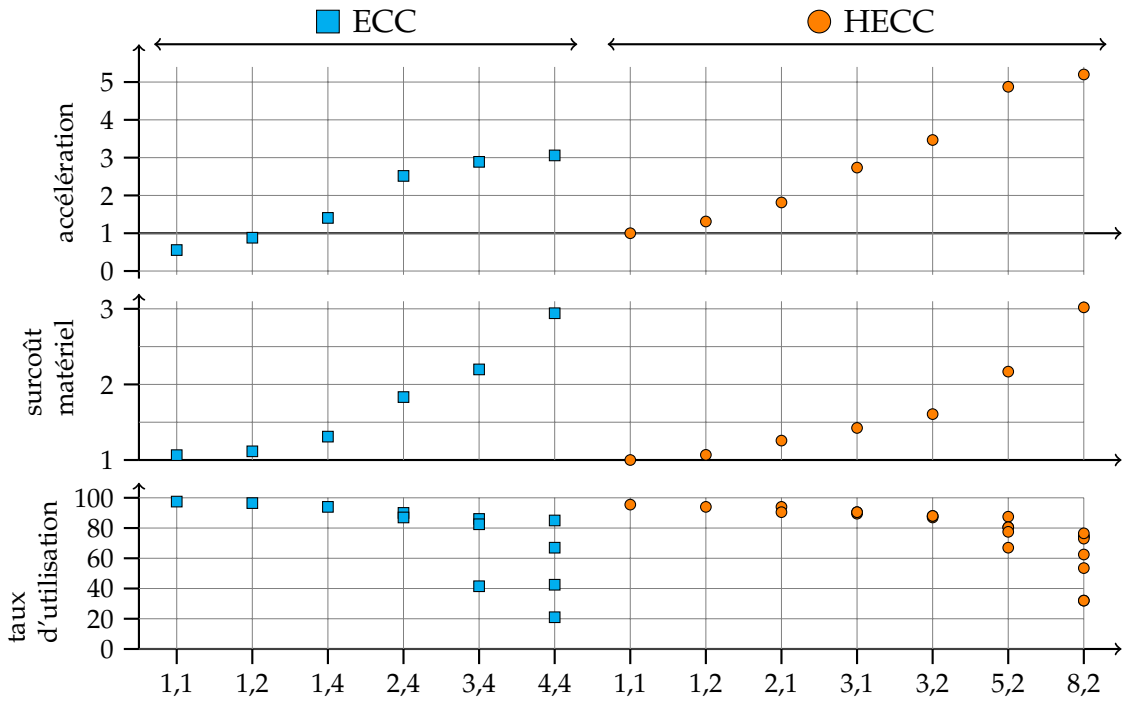


FIGURE 3 – Accélération, surcoût matériel et taux d'utilisation moyen des multiplieurs pour les configurations les plus efficaces. Toutes les valeurs sont normalisées par rapport à la configuration la plus petite d'HECC (c.-à-d. $n_M = 1$ et $n_B = 1$).

des résultats d'implantation de crypto-processeurs ECC sur $\mathbb{F}_p$ de la littérature sur des FPGA proches (utilisant aussi des LUT-6 pour le Virtex 5). Nous voyons que notre solution est tout à fait performante par rapport aux solutions sans blocs DSP. Les résultats de [10] sont bien meilleurs en vitesse mais au prix de l'utilisation de 37 blocs DSP. Une comparaison directe en surface n'est pas possible. Nous pensons obtenir de meilleurs résultats lorsque nous utiliserons des blocs DSP. Nous ferons aussi une comparaison avec plus de solutions de l'état de l'art et sur plus de familles différentes de FPGA.

Source	FPGA	surface slices / blocs DSP	fréquence MHz	temps [k]**P** ms
ECC 1,2	Spartan 6	573 / 0	233	17.7
ECC 1,4		673 / 0	233	11.1
ECC 2,4		942 / 0	220	6.2
ECC 3,4		1 130 / 0	214	5.4
[10]	Virtex-5	1 725 / 37	291	0.38
	Virtex-4	4 655 / 37	250	0.44
[5]	Virtex-4	13 661 / 0	43	9.2
		20 123 / 0	43	7.7

TABLE 7 – Comparaison avec des crypto-processeurs ECC sur $\mathbb{F}_p$ de la littérature.

5. Conclusion

Dans ce papier, nous avons présenté un crypto-processeur paramétrable pour la cryptographie sur courbes elliptiques (ECC) et hyper-elliptiques (HECC). En explorant le nombre d'unités arithmétiques parallèles et leurs tailles, nous pouvons proposer un grand nombre de compromis vitesse–surface pour répondre au mieux à divers besoins applicatifs. Nos résultats expérimentaux confirment bien que HECC est plus efficace qu'ECC à niveau de sécurité théorique équivalent. Par exemple, pour un même coût en surface, HECC 128 bits est environ 40 % plus rapide qu'ECC 256 bits.
Dans l'avenir, nous allons optimiser notre architecture et étudier le support de nouveaux types d'unités de calcul. Nous analyserons les gains offerts par HECC par rapport à ECC pour d'autres niveaux de sécurités (tailles de corps et de clés). De plus, nous étudierons les aspects de robustesse aux attaques par canaux cachés en mesurant l'impact des différents paramètres architecturaux et algorithmiques sur la réussite des attaques. Enfin, nous essayerons de proposer des contre-mesures pour notre architecture.

Remerciements

Ce travail a été financé en partie par les projets PAVOIS (ANR-12-BS02-002-01, `http://pavois.irisa.fr/`) et HAH (Labex CominLab et Lebesgue, `http://h-a-h.inria.fr/`).

Bibliographie

1. Bernstein (D. J.) et Lange (T.). – Explicit-formulas database. – `http://hyperelliptic.org/EFD/`.

2. Boston (N.), Clancy (T.), Liow (Y.) et Webster (J.). – Genus two hyperelliptic curve coprocessor. *In : Proc. Cryptographic Hardware and Embedded Systems (CHES)*, pp. 400–414. – Springer, juin 2003.
3. Chabrier (T.) et Tisserand (A.). – On-the-fly multi-base recoding for ECC scalar multiplication without pre-computations. – In *Proc. 21st Symposium on Computer Arithmetic (ARITH)*, pp. 219–228. IEEE Computer Society, avril 2013.
4. Cohen (H.) et Frey (G.) (édité par). – *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. – Chapman & Hall/CRC, juillet 2005, *Discrete Mathematics and Its Applications*.
5. Ghosh (S.), Alam (M.), Roychowdhury (D.) et Gupta (I.). – Parallel crypto-devices for GF(p) elliptic curve multiplication resistant against side channel attacks. *Computers and Electrical Engineering*, vol. 35, n. 2, mars 2009, pp. 329–338.
6. Hankerson (D.), Menezes (A.) et Vanstone (S.). – *Guide to Elliptic Curve Cryptography*. – Springer, 2004.
7. Kim (H.), Wollinger (T.), Choi (Y.), Chung (K.) et Paar (C.). – Hyperelliptic Curve Coprocessors on a FPGA. *In : Proc. 5th International Workshop on Information Security Applications (WISA)*, pp. 360–374. – Springer, 2005.
8. Lange (T.). – Efficient arithmetic on genus 2 hyperelliptic curves over finite fields via explicit formulae. – Cryptology ePrint Archive, Report 2002/121, 2002. `http://eprint.iacr.org/`.
9. Lange (T.). – Formulae for Arithmetic on Genus 2 Hyperelliptic Curves. *Applicable Algebra in Engineering, Communication and Computing*, vol. 15, n. 5, février 2005, pp. 295–328.
10. Ma (Y.), Liu (Z.), Pan (W.) et Jing (J.). – A high-speed elliptic curve cryptographic processor for generic curves over GF(p). – In *Proc. 20th International Workshop on Selected Areas in Cryptography (SAC)*, *LNCS*, volume 8282, pp. 421–437. Springer, août 2013.
11. Mangard (S.), Oswald (E.) et Popp (T.). – *Power Analysis Attacks : Revealing the Secrets of Smart Cards*. – Springer, 2007.
12. Montgomery (P. L.). – Modular multiplication without trial division. *Mathematics of Computation*, vol. 44, n. 170, avril 1985, pp. 519–521.
13. National Institute of Standards and Technology (NIST). – FIPS 186-2, digital signature standard (DSS), 2000.
14. Sakiyama (K.), Batina (L.), Preneel (B.) et Verbauwhede (I.). – Superscalar coprocessor for high-speed curve-based cryptography. *In : Proc. Cryptographic Hardware and Embedded Systems (CHES)*, pp. 415–429. – Springer, 2006.