Architecture d'un crypto processeur ECC sécurisé contre les attaques physiques

Simon Pontie

Architecture d'un crypto processeur ECC sécurisé contre les attaques physiques.

Simon PONTIE
Univ. Grenoble Alpes, TIMA, F-38031 Grenoble
CNRS, TIMA, F-38031 Grenoble
46, avenue Félix Viallet
38031 GRENOBLE Cedex, France

E-mail : simon.pontie@imag.fr

Résumé

Les systèmes embarqués doivent supporter de plus de fonctions. C'est le cas du chiffrement et de l'authentification. Les contraintes apportées par les systèmes embarqués remettent en question l'utilisation de l'algorithme RSA pour répondre à ce besoin : il devient très intéressant d'assurer ces fonctions en utilisant la cryptographie sur les courbes elliptiques, nécessitant moins de ressources et plus économes en énergie. Il existe de nombreuses attaques sur les fonctions cryptographiques mettant à profit un accès physique au matériel les implémentant : attaques par analyse simple ou différentielle de la puissance consommée (SPA, DPA), attaque par analyse du temps de calcul. Nous présenterons un coprocesseur robuste et sécurisé capable d'effectuer toutes les opérations critiques inhérentes aux courbes elliptiques et minimisant la fuite d'information par canaux cachés (temps de calcul et puissance consommée).

1. Introduction

Lorsque de nombreuses entités doivent échanger des messages de manière sécurisée, la cryptographie asymétrique permet de simplifier les infrastructures d'échange de clés. Cette solution est habituellement réservée à des systèmes complexes. La multiplication des systèmes embarqués nous pousse à implémenter des algorithmes à clés publiques sur ceux-ci. Les courbes elliptiques appliquées à la cryptographie asymétrique [1] sont particulièrement adaptées à ce type de systèmes puisqu'elles permettent de minimiser les ressources nécessaires par rapport à l'algorithme RSA [2]. En effet, une clé ECC (Elliptic Curve Cryptography) de 163 bits est aussi forte qu'une clé RSA de 1024 bits [3].

Nous avons mis en œuvre un crypto-processeur supportant les courbes elliptiques. Dans la partie 2 nous expliciterons la manière dont les courbes elliptiques sont exploitées en cryptographie. Nous montrerons dans la troisième partie que comme toutes les implémentations de fonctions cryptographiques, celles basées sur les courbes elliptiques sont sensibles à des attaques par canaux cachés. Dans la quatrième partie nous détaillerons la contre-mesure que nous avons mise en œuvre contre les attaques par analyse de puissance et celles par analyse du temps de calcul. Dans la partie 5 nous détaillerons l'architecture qui nous a permis d'implémenter cette contre-mesure. Dans la sixième partie nous évaluerons notre crypto-processeur en le comparant à d'autres implémentations.

TIMA is Partner of the Labex PERSYVAL Lab (ANR-11-LABX-0025).

2. Les courbes elliptiques en cryptographie

Tous les protocoles cryptographiques basés sur les courbes elliptiques reposent sur la difficulté d'inverser l'opération de multiplication scalaire. Cette opération s'appelle le logarithme discret. Les notions mises en œuvre dans la cryptographie basée sur les courbes elliptique sont hiérarchisées en quatre niveaux (figure 1). Cette hiérarchie est importante car elle se retrouvera dans l'architecture du processeur.

En cryptographie, une courbe elliptique est définie par son équation, le corps sur lequel elle s'applique et un point générateur. Les deux corps utilisables sont les corps de Galois GF(p) et GF(2^d), respectivement de caractéristiques p et 2. La plupart des protocoles rendent obligatoire le support du corps GF(p) et optionnel le support de GF(2^d), nous nous intéresserons donc plus particulièrement à GF(p). Ce corps est communément appelé corps premier de caractéristique p. Les lois d'additions et de multiplications sont les lois classiques modulo p. L'élément nul est 0 et l'élément neutre est 1. Ceci définit le niveau 1 de la hiérarchie sur les courbes elliptiques (figure 1). C'est dans ce corps que seront exprimées les coordonnées des points de la courbe et l'équation de celle-ci. Toutes les courbes elliptiques dans ce corps sont isomorphes à une courbe de Weierstraβ réduite de la forme suivante :

$$E : y^2 = x^3 + a_4 x + a_6 \text{ sur GF}(p).$$

On peut définir géométriquement l'addition de deux points de la courbe. Pour illustrer cette construction

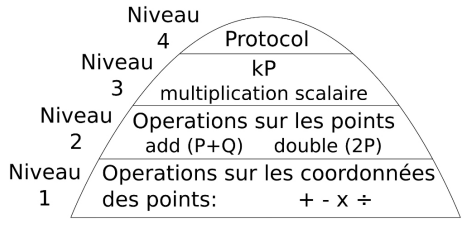

Figure 1 : Hiérarchie des concepts sur les courbes elliptiques

géométrique nous nous appuierons sur la figure 2 qui représente une courbe elliptique sur le corps des réels. L'addition de deux points (P et Q) de la courbe se construit en identifiant le point d'intersection entre la droite passant par ces deux points et la courbe. Le résultat est le symétrique de ce point d'intersection par rapport à l'axe des abscisses ($P + Q$). Si P et Q sont confondus, il s'agit d'un doublage et la construction géométrique précédente ne fonctionne plus. Le doublage d'un point (R) est construit de la même manière mais en prenant la droite tangente à R. Dans la pratique, l'addition et le doublage sont implémentés par des formules sur les coordonnés des points découlant de cette construction géométrique. Ces deux opérations forment le niveau 2 dans la hiérarchie des courbes elliptiques (figure 1).

La multiplication scalaire forme le troisième niveau. Cette opération est le produit entre un scalaire et un point de la courbe. Plusieurs algorithmes permettent de réaliser cette opération à partir de l'addition et du doublage de points. Nous présenterons ici le plus simple d'entre eux.

L'algorithme « Double and Add » permet de calculer le produit kP à partir d'un scalaire k et d'un point P. On initialise un accumulateur avec le point à l'infini qui est l'élément neutre du groupe formé par les points de la courbe. On parcourt k bit à bit du poids fort au poids faible. Pour chaque bit on réalise un doublage de l'accumulateur ; si ce bit est non nul on ajoute P à l'accumulateur. Il faut donc en moyenne n doublages et n/2 additions, n étant le nombre de bits du scalaire.

Le niveau 4 est le niveau protocolaire. Il existe différents protocoles basés sur les courbes elliptiques. Tous sont basés sur la difficulté de retrouver k lorsque l'on connait les points Q et G tel que Q=kG. Ceci est le problème du logarithme discret. Le groupe décrit précédemment est un des rares dans lequel aucun algorithme n'a été trouvé pour résoudre ce problème en temps sous-exponentiel. La clé secrète est un scalaire qui dans le cas d'un déchiffrement sera utilisé dans une multiplication scalaire. Cette opération est donc critique puisqu'elle fait intervenir la clé secrète. Nous allons voir par la suite qu'une implémentation naïve de cette multiplication peut être la source de fuites d'informations sur la valeur de la clé secrète.

3. Attaques physiques

Dans cette partie nous présenterons trois attaques exploitant des canaux cachés. Nous nous appuyons sur les faiblesses de deux algorithmes de multiplication scalaire.

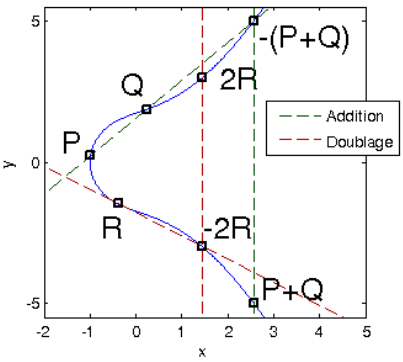

Figure 2 : Addition et doublage de points sur $y^2 = x^3 + 2x + 3$

Le temps de calcul est la source d'information la plus facilement exploitable pour accélérer le calcul d'une clé privée à partir de sa clé publique. Nous allons l'illustrer sur l'algorithme « Double and Add » précédemment présenté. Cet algorithme nécessite toujours n doublages, n étant le nombre de bits pour coder l'ordre de la courbe. Le nombre d'additions est le poids de Hamming du scalaire. Il est donc aisé de retrouver ce poids à partir du temps de calcul [4] car les variations du temps nécessaire pour effectuer un doublage ou une addition sont faibles.

Il existe une autre attaque basée sur la non-uniformité des opérations d'additions et de doublages. En observant la puissance consommée par le crypto-processeur au cours d'un chiffrement, il est possible de reconstruire l'ordre des opérations internes par identification de motifs. Si l'algorithme « Double and Add » a été implémenté, un attaquant peut facilement retrouver la valeur de la clé secrète à partir d'une seule trace de puissance consommée, c'est l'attaque par SPA (Single Power Analysis) [5]. La figure 3 montre la puissance consommée par un coprocesseur implémentant l'algorithme « Double and Add » sur une technologie AMS 0,35 µm. On peut facilement différentier les deux motifs de consommation qui correspondent aux doublages et aux additions durant la multiplication scalaire. Il est trivial de reconstruire la clé secrète (*179*).

Il existe de nombreux algorithmes de multiplication scalaire résistants à ces deux attaques. Par exemple, l'échelle de Montgomery qui, contrairement à l'algorithme précédent, scanne le scalaire de droite à gauche et utilise deux points dont un est toujours égal à l'addition entre l'autre et P. Il n'est pas sensible à une attaque par SPA ni une attaque par analyse du temps de calcul car il y a toujours une addition après les doublages. Par contre il est sensible à une attaque plus évoluée sur la puissance consommée : l'attaque par DPA (Differential Power Analysis) [5]. C'est une attaque statistique basée sur l'exploitation d'un grand nombre de traces de puissance consommée par des multiplications scalaires entre la clé secrète et différents points P.

Nous allons maintenant présenter un algorithme résistant à ces trois attaques. Puis nous détaillerons la manière dont nous l'avons implémenté.

4. Multiplication scalaire résistante

Notre algorithme est basé sur un parcours du scalaire de gauche à droite avec fenêtrage. Cela signifie que l'on parcourt la clé secrète par digit de plus de 1 bit. Ceci permet d'économiser des additions. En contrepartie, il faudra pré-calculer les valeurs dP pour toutes les valeurs possibles du digit d.

Dans le but de résister aux attaques par DPA, nous tirerons au hasard la taille de chaque fenêtre au cours de

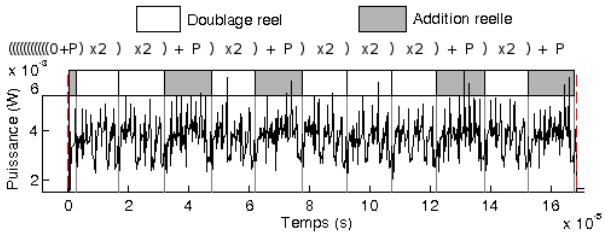

Figure 3 : Trace de puissance de l'algorithme « Double and Add »

la multiplication scalaire. Une attaque par SPA comptant les doublages espaçant les additions sera capable de déterminer la taille de chaque fenêtre. C'est pourquoi nous injecterons au cours du calcul des additions fictives. Ces additions fictives combinées au fenêtrage aléatoire permettront de désynchroniser les attaques par DPA. Pour minimiser la prédiction de la position de ces additions fictives, elles seront placées aléatoirement au cours de chaque calcul. Pour résister aux attaques par analyse du temps de calcul nous réaliserons toujours autant d'opérations.

La taille de chaque fenêtre est choisie aléatoirement dans l'intervalle [*WMIN*,*WMAX*]. *WMAX* correspond à la taille maximale de fenêtre autorisée. Elle imposera la taille de la table de points pré-calculés qui contiendra tous les points *dP* avec *d* entre *1* et 2^{WMAX}-*1*. *WMIN* est la taille minimale d'une fenêtre. Le nombre maximal d'additions sera atteint lorsque notre scalaire sera partitionné en fenêtres ayant toutes la taille *WMIN*. Nous appellerons cette configuration le pire cas. Pour réaliser toujours autant d'additions au cours d'une multiplication scalaire, nous comparerons le calcul réel au pire cas en temps réel. Ceci nous permet de connaitre à tout moment le nombre d'additions que nous avons économisées grâce au fenêtrage. Toutes ces économies seront dépensées en additions fictives pour obtenir un nombre d'additions constant pour n'importe quels partitionnements du scalaire. Ces additions fictives permettent de dissimuler la taille des fenêtres utilisées. Elles doivent donc avoir lieux au cours du calcul. La mise en œuvre de ceci est réaliser par un compteur d'additions fictives à effectuer (*fictive_cpt*) qui sera incrémenté dès qu'une addition sera économisée et décrémenté dès qu'une addition fictive sera insérée. Pour terminer le calcul avec ce compteur vide, nous augmentons la probabilité d'insérer une opération fictive quand la valeur du compteur est élevée.

5. Architecture

L'architecture de notre coprocesseur suit la hiérarchie des courbes elliptiques (figure 1). Le premier niveau implémente l'arithmétique sur le corps GF(*p*) qui est basé sur un additionneur-soustracteur supportant des opérandes de la taille du plus grand entier premier *p* que l'on souhaite supporté. Un multiplieur de Montgomery [6] utilisant cet additionneur-soustracteur nous permet de réaliser l'opération suivante :

$$C = A \times B \times 2^{-r} \text{ modulo } p$$

r est le nombre de bits nécessaires pour coder *p*. Ce multiplieur est performant mais en contre partie nous devons travailler dans le domaine de Montgomery pour s'affranchir du terme 2^{-r}. L'image de *a* dans ce domaine est *A* tel que :

$$A = a \times 2^{r} \text{ modulo } p$$

En travaillant dans ce domaine, nous réalisons une opération de multiplication puisqu'en multipliant *A* et *B* nous obtenons bien l'image de *a* x *b* qui est notée *C* :

$A \times B \times 2^{-r} = a \times 2^{r} \times b \times 2^{r} \times 2^{-r} = (a \times b) \times 2^{r} = c \times 2^{r} = C$

Pour entrer dans le domaine de Montgomery on utilise le multiplieur avec les opérandes *a* et 2^{2r}. Pour sortir du domaine de Montgomery il suffit d'utiliser les opérandes *1* et *A*. L'addition et la soustraction modulaire sont réalisées en utilisant le même additionneur-soustracteur que le multiplieur. Elles utilisent la retenue en sortie de celui-ci pour comparer le résultat avec *p* et effectuer l'opération de réduction en modulo. La multiplication, l'addition et la soustraction sur GF(*p*) sont implémentées dans le même composant pour permettre de mutualiser l'accumulateur interne.

Le composant *registres courbe* contient quatre registres sur *r* bits. Le premier contient *p*. Le deuxième contient 2^{2r} modulo *p* pour pouvoir entrer dans le domaine de Montgomery. Les deux derniers registres sont les coefficients a_4 et a_6 de la courbe exprimés dans le domaine de Montgomery. Le multiplieur est utilisé pour vérifier la valeur 2^{2r} modulo *p* et transposer les deux coefficients dans le domaine de Montgomery. La valeur de chacun de ces registres est disponible en sortie.

Le niveau 2 est schématisé sur la figure 4. Il est implémenté dans un seul composant permettant de réaliser l'addition et le doublage de points. Pour accélérer le calcul nous utilisons une représentation Jacobienne [7] des points qui utilise trois coordonnées par point (*X,Y,Z*). La représentation classique d'un point utilise les coordonnées affines (*x,y*).

$$(x , y) = (X/Z^2 , Y/Z^3)$$

Ce système de coordonnées nécessite de stocker une troisième coordonnée mais permet de ne réaliser qu'une seule division par multiplication scalaire (pour exprimer le point final en coordonnées affine). En effet l'avantage de cette représentation est que les formules d'addition et de doublage de points ne nécessitent pas d'inversion [7].

Le composant de niveau 2 (figure 4) contient 3 registres de *r* bits qui représentent les coordonnées du point *Q* qui servira d'accumulateur. Il contient aussi 5 autres registres pour pouvoir réaliser les opérations $Q = Q + P$ et $Q = 2 \times Q$. Tous les calculs sont réalisés dans le domaine de Montgomery avec une seule instance de notre multiplieur-additionneur-soustracteur sur GF(*p*). Le composant permet aussi d'effectuer trois autres opérations simples sur le point accumulateur Q : initialisation, transposition dans le domaine de Montgomery et sortie de ce domaine. Il est possible de désactiver l'écriture sur *Q* pour réaliser des opérations fictives. Ces opérations sont identiques aux opérations réelles mais sans l'écriture finale dans l'accumulateur *Q*. À la fin de l'opération un signal d'erreur peut être transmis si une des opérandes était le point à l'infini ou si l'on a tenté d'additionner deux points égaux.

Le niveau 3 contient quatre composants : un générateur pseudo aléatoire, une table de points pré-calculés, le composant *kP_po* qui contient la partie opérative de la multiplication scalaire et le composant *kP_pc* qui contient la partie contrôle de cette multiplication.

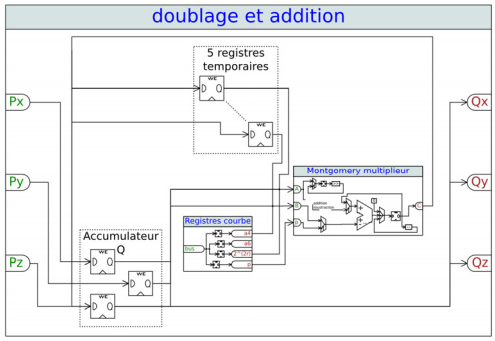

Figure 4 : Partie opérative des niveaux 1 et 2.

La clé secrète est inscrite dans un registre à décalage de *kP_po*. À chaque doublage de *Q*, ce registre est décalé vers la gauche. À la fin de la fenêtre, les bits ayant débordés du registre à décalage forment le digit qui nous permet de sélectionner le point de la table que nous devons ajouter à *Q*. Ce composant implémente aussi le compteur d'additions fictives à effectuer (*fictive_cpt*). Il laisse à la partie contrôle la possibilité de prendre la décision d'insérer une opération fictive à la fin de chaque addition et doublage, qu'ils soient réels ou fictifs.

La partie contrôle se charge du remplissage de la table de points pré-calculés à la réception du point *P* en exploitant *kP_po*. Ces points sont stockés en coordonnées Jacobiennes dans le domaine de Montgomery. Après cette étape elle charge la clé *k* dans la partie opérative et démarre le calcul de la multiplication scalaire *kP*. À la fin de chaque fenêtre, la partie contrôle tire au hasard la taille de la prochaine fenêtre. Après chaque addition et doublage réalisé par la partie opérative, la partie contrôle tente d'insérer une opération fictive si *fictive_cpt* n'est pas vide. Plus *fictive_cpt* est grand plus il y aura de chances d'effectuer l'addition fictive. Cette opération est une addition entre l'accumulateur Q et un point pris au hasard dans la table de points pré-calculés.

6. Discussion

Dans cette partie nous allons comparer notre coprocesseur à quatre autres implémentations sur des cibles FPGAs (figure 5). Pour effectuer ceci, nous avons normalisé leurs performances par rapport à quatre synthèses de notre coprocesseur sur les mêmes FPGAs et avec une arithmétique sur le même nombre de bits.

Il est possible de se protéger d'une attaque SPA en utilisant des formules d'addition et de doublage unifiées [9] mais cette contre-mesure ne protège pas contre les attaques par analyse du temps de calcul. Beaucoup d'implémentations de crypto-processeur ECC choisissent l'échelle de Montgomery comme algorithme de multiplication scalaire car il permet de se protéger contre les attaques par SPA et celles par analyse du temps de calcul 00. Par contre, cet algorithme est sensible aux attaques par DPA [5]. Il est possible de ralentir cette attaque en utilisant une représentation aléatoire des points dans un domaine de Montgomery variable 0. La contre-mesure que nous proposons étant sur l'algorithme de multiplication scalaire, elle peut facilement être combinée avec une telle représentation.

Notre coprocesseur occupe plus de slices que la plupart des autres implémentations : 14886 slices pour le support de toutes les courbes sur GF(*p*) avec une taille maximale pour *p* de 256 bits. Ceci est principalement dû à la table de points pré-calculés qui occupe 53% de la surface totale. La taille de cette table peut être réduite en diminuant *WMAX* (ici à 4 bits) au détriment du niveau de protection contre les attaques par DPA. Avec la même conséquence, il est aussi possible de diminuer le temps de calcul en augmentant *WMIN* (ici à 2 bits).

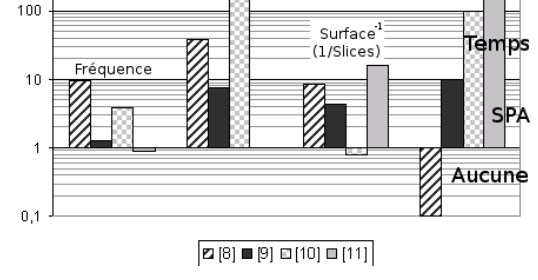

Figure 5 : Comparaison entre notre coprocesseur et quatre autres implémentations

Nous avons fait le choix de ne pas utiliser les composants spécifiques aux FPGAs (DSP, BRAM) pour garder la compatibilité avec les cibles ASICs. Pour minimiser la surface nous n'utilisons qu'un seul multiplieur. Nous pouvons voir que le temps de calcul peut être optimisé en utilisant plusieurs multiplieurs 0 mais ceci a un énorme coup en surface. L'article [8] ne supportant que quelques courbes montre que l'utilisation de DSPs permet de réduire le chemin critique et donc d'augmenter sensiblement la fréquence d'horloge.

7. Conclusions

Nous avons montré que le fenêtrage dans la multiplication scalaire peut être utilisé pour implémenter une contre-mesure aux attaques par canaux cachés (SPA, DPA, analyse du temps de calcul). La modularité de celle-ci permet de choisir le niveau de protection (DPA) en fonction des contraintes de surfaces et de temps de calcul. La mise en œuvre d'attaques réelles sur cible FPGA permettra de quantifier précisément la protection contre les attaques par DPA. Il est prévu d'implémenter une arithmétique de base pipelinée pour réduire le temps de calcul.

Références

[1] Koblitz, Neal. "Elliptic curve cryptosystems." Mathematics of computation 48, no. 177 (1987): 203-209.

[2] Rivest, Ronald L., Adi Shamir, and Len Adleman. "A method for obtaining digital signatures and public-key cryptosystems." Communications of the ACM 21, no. 2 (1978): 120-126

[3] Lenstra, Arjen K., and Eric R. Verheul. "Selecting cryptographic key sizes." Journal of cryptology 14, no. 4 (2001): 255-293.

[4] Kocher, P. C. (1996, January). Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Advances in Cryptology—CRYPTO'96 (pp. 104-113). Springer Berlin Heidelberg.

[5] Kocher, P., Jaffe, J., & Jun, B. (1999, January). Differential power analysis. In Advances in Cryptology—CRYPTO'99 (pp. 388-397). Springer Berlin Heidelberg.

[6] Montgomery, P. L. (1985). Modular multiplication without trial division. Mathematics of computation, 44(170), 519-521.

[7] Cohen, H., Frey, G., Avanzi, R., Doche, C., Lange, T., Nguyen, K., & Vercauteren, F. (Eds.). (2005). Handbook of elliptic and hyperelliptic curve cryptography. CRC press.

[8] Güneysu, T., & Paar, C. (2008). Ultra high performance ECC over NIST primes on commercial FPGAs. In Cryptographic Hardware and Embedded Systems–CHES 2008 (pp. 62-78). Springer Berlin Heidelberg.

[9] Kim, C. H., Kwon, S., & Hong, C. P. (2008). FPGA implementation of high performance elliptic curve cryptographic processor over GF (2^{163}). Journal of Systems Architecture, 54(10), 893-900.

[10] Nagaraja, S., & Sridhar, V. (2013, September). A Unified Architecture for a Dual Field ECC Processor Applicable to AES. In Computational Intelligence, Modelling and Simulation (CIMSim), 2013 Fifth International Conference on (pp. 321-326). IEEE.

[11] Morales-Sandoval, M., et al. "A reconfigurable GF (2^M) elliptic curve cryptographic coprocessor." Programmable Logic (SPL), 2011 VII Southern Conference on. IEEE, 2011.