FPGA2 - Analyse documentaire et critique
Extending RISC-V for HW acceleration of Post-Quantum Scheme LAC
DOI 10.23919/DATE48585.2020.9116567

Legoueix Nicolas n°28709604

28 janvier 2022

Table des matières

Première partie

Résumé

1 Contexte

1.1 Nécessité de nouveaux protocoles, travail effectué

La majorité des communications actuelles sont réalisées à l'aide de l'algorithme de cryptographie RSA. Cet algorithme de cryptographie asymétrique utilise deux clés pour le chiffrement des données : une clé publique -crée par un des interlocuteurs et rendu publique- servant à chiffrer les données et une clé privée -connue seulement de l'interlocuteur devant déchiffrer les données et indispensable à ce processus-. Cette méthode se base sur le fait qu'il est impossible de déchiffrer un message sans la clé privée en un temps dit raisonnable. En effet, le temps nécessaire pour calculer des factorielles (sur lesquels se base RSA) étant exponentiel, il est de plus en plus difficile de déchiffrer RSA par attaques de force brute. [1].
Cependant, l'avènement des processeurs quantiques, et l'arrivée prochaine de vrais ordinateurs quantiques remet en question la résistance de l'algorithme RSA. En effet, ces derniers seraient théoriquement capables de déchiffrer une communication RSA en seulement quelques heures de travail. [2]. Il devient donc nécessaire de développer des algorithmes résistants aux attaques quantiques, ainsi que de créer le matériel qui sera capable d'exécuter ces nouveaux algorithmes. Parmi eux figurent l'algorithme LAC qui es présenté dans l'article proposé.

Cet article propose une implémentation accélérée matériellement de l'algorithme LAC. Les auteurs y décrivent un total de quatre accélérateurs matériels visant à mitiger les effets des principaux goulots d'étranglements de LAC, tous intégrés dans le pipeline d'un coeur RISC-V modifié. Les auteurs décrivent également les modifications apportées à cette ISA afin de pouvoir utiliser leur matériel.

1.2 Algorithme LAC

L'algorithme se déroule en trois étapes :
— Génération des clés : Un premier interlocuteur génère une clé publique et une clé privée. Cette génération est effectuée à partir d'une graine qui est ensuite étendue via l'algorithme SHA256. Une multiplication polynomiale afin de créer une instance RLWE (Ring Learning With Error).
— Chiffrement : Le message à envoyer est chiffré via la clé publique à l'aide d'un encodeur BCH, puis transformé en un nombre polynomial. Enfin, ce polynôme est "camouflé" dans l'instance RLWE qui introduit de polynômes parasites dans le message chiffré. Le message est ensuite envoyé.
— Déchiffrement : L'instance RLWE est reçue par le second interlocuteur en possession de la clé privée. On soustrait ensuite de l'instance le plus gros des polynômes parasites, puis on utilise un décodeur BCH (et la clé privée) pour soustraire le bruit restant. On obtient alors le message déchiffré.
Il est important de remarquer trois opérations utilisées dans cet algorithme : la génération de polynômes lors de la génération des clés, la multiplication de polynômes (lors de la création d'instance RLWE et lors de la suppression des polynômes parasites pendant le déchiffrement), et la correction d'erreurs.
La multiplication de polynômes est particulièrement complexe, il serait donc intéressant d'accélérer cette opération avec du matériel.

2 Implémentations

2.1 Accélération matérielle

Quatre accélérateurs sont décrits dans cet article, chacun visant à diminuer les effets des goulots d'étranglements inhérents à l'algorithme LAC.

2.1.1 MUL-TER : Multiplicateur Polynomial Ternaire

LAC se base sur la multiplication de nombre polynomiaux pour le chiffrement et le traitement du bruit lors du déchiffrement. Étant donné que beaucoup de multiplications sont effectuées sur ce type de nombres et que ce sont des opérations complexes, il est naturel de vouloir accélérer leur traitement. Dans le cas présent, les

1. Actuellement, le record est RSA-250 soit 829bits, établit en février 2020 par F. Boudot, P. Gaudry, A. Guillevic, N. Heninger, E. Thomé et P. Zimmermann. Les clés privées actuelles font actuellement entre 1024 et 2048 bits. Temps requis : 2700 années CPU, c'est a dire qu'il aurait fallu 2700 années de travail à un seul coeur (dans ce cas, un Intel Xeon Gold 6130 a 2.1Ghz) pour résoudre le problème. Voir le papier de recherche publié : DOI 10.1007/978-3-030-56880-1_3 , page 30
2. Voir DOI 10.22331/q-2021-04-15-433

auteurs sont partis d'un multiplicateur existant et y ont ajouté la capacité à effectuer des calculs de convolutions négatives.
MULTER est un banc de MAU (Modular Arithmetic Units) capables d'effectuer des additions, soustractions ou une opération neutre (forwarding) afin de réaliser une convolution bouclée à partir de deux polynômes.

2.1.2 MUL-CHIEN

L'algorithme LAC introduit des erreurs lors du (dé)chiffrement. Il est donc nécessaire de savoir détecter et corriger ces dernière efficacement. Le module MUL-CHIEN implémente en matériel l'algorithme de recherche de Chien afin de rechercher la racine des erreurs détectées. Cette opération nécessite de multiplier les éléments d'un polynôme par une constante. Pour plus de flexibilité et de simplicité, les auteurs ont développé leur propre multiplicateur pour tenir ce rôle : MUL-GF (Gallois Field Multiplier) capable d'effectuer quatre opérations en parallèle en seulement neufs cycles à partir du travail d'autres chercheurs. MUL-CHIEN est composé d'un banc de MUL-GF.

2.1.3 SHA256

La génération des nombre polynomiaux, vitale à l'algorithme LAC, est faite par un accélérateur développé par les mêmes chercheurs que ceux écrivant cet article. [3].

2.1.4 MOD-Q

MOD-Q est un module implémentant l'algorithme de Barrett servant à effectuer de la réduction de modulos en temps constant.

2.2 Jeu d'instructions

Le processeur utilisé est une déclinaison du RISC-V nommée RISCY. Il s'agit d'un processeur 32 bits avec un pipeline a 4 étages (IF, ID, EX et WB) implémentant les sets d'instructions de base I (opérations sur entiers), C (instructions compressées) et M (multiplication et division d'entiers).
Le pipeline de base a été modifié afin d'y intégrer l'unité de calculs qui englobe les accélérateurs développés : dans l'étage EX, en plus de l'ALU (Arithmetic and Logic Unit) on retrouve une PQ-ALU (Post-Quantum ALU). Afin d'utiliser cette dernière, quatre nouvelles instructions ont été ajoutées, toutes utilisant le format d'instructions R (elles ne nécessitent pas l'utilisation d'immédiats). Un nouvel opcode est utilisé (bits 6 à 0 : 0x77 soit 0111 0111 en binaire) pour ces instructions. Le champ func3 (bits 14 à 12) est utilisé pour déterminer quel accélérateur doit être utilisé. Les champs rs1 (bits 19 à 15), rs2 (bits 24 à 20) et rd (bits 11 à 7) sont utilisés en tant que buffers vers d'autres registres dans le GPR (General Purpose Register bank). Ce choix s'explique par le fait que les registres d'entrée (rs1 & 2) et de sortie (rd) ne sont pas suffisamment grands pour pouvoir contenir les opérandes et résultats des calculs effectués par la PQ-ALU. Ces instructions sont :

— pq.mul_ter
— pq.mul_chien
— pq.sha256
— pq.modq

3. Voir DOI :10.1007/978-3-030-23425-6_13 : Efficient Hardware/Software Co-design for NTRU

Deuxième partie

Analyse Critique

3 Choix du RISC-V

On pourrait tout d'abord questionner le choix du RISC-V comme jeu d'instructions. Il existe en effet d'autres jeux d'instructions qui seraient en mesure d'implémenter ce genre d'algorithme.
x86 ou Arm viennent a l'esprit.

3.1 Arm

Arm est très établis dans le monde des micro-contrôleurs notamment avec son Cortex M4 ou le Cortex A53. Il est donc très courant que les cartes de développement soient équipées de processeurs Arm. Les résultats obtenus par une implémentation d'une solution à un problème donné sont ainsi très souvent donnés pour ce type de plate-forme.
Cependant, le développement d'IP matérielles sur plate-forme Arm n'est pas aisé particulièrement en raison de la nature fermée des technologies proposées. Il est donc difficile d'obtenir des accélérateurs hautement intégrés aux processeurs. Les durées de développement en sont également plus longues
Un autre désavantage majeur d'Arm est que l'utilisation et la modification de sa propriété intellectuelle est payante, ce qui rend le coût de développement de nouveau matériel plus élevé.
Enfin, on pourrait mentionner les potentiels conflits d'intérêts induits par le fait d'utiliser du matériel appartenant à une entreprise pour des applications de sécurité. En effet, dans un contexte géopolitique tendu, il serait légitime de se méfier de matériel produit par une entreprise basée dans un autre pays.

3.2 RISC-V

RISC-V est un jeu d'instruction libre et open source. Par essence, toute personne est donc libre de le modifier a sa guise. Cette philosophie facilite grandement la facilité et la rapidité de développement de nouveau matériel, notamment grâce au partage de ressources au sein de la communauté.
De plus, il s'agit d'une ISA (Instruction Set Architecture) modulaire, permettant aux développeurs d'y ajouter leur propres instructions. Cette propriété rend le RISC-V idéal pour la création d'accélérateurs matériels. Elle permet également de n'implémenter que ce qui est nécessaire au micro-contrôleur. Cela signifie qu'il est possible de minimiser la taille du circuit en évitant d'embarquer du matériel ne servant pas. C'est notamment un avantage dans le cadre du développement de matériel embarqué dont l'espace est limité et dont la consommation énergétique doit être contenue.
Il est également possible d'en modifier directement le pipeline, ce qui permet d'y intégrer des accélérateurs matériels. Ce fort niveau d'intégration permet d'éviter les pertes de performances dues à l'attente du processeur lors de l'utilisation des bus de communication.
En revanche, comme il a pû l'être constaté dans l'article proposé, il existe des disparités de performances entre les deux jeux d'instructions. Dans ce cas, de l'ordre de 20%.

4 SHA256 et Keccak

L'article mentionne la possibilité de remplacer le module de génération polynomiale SHA256 par un module implémentant l'algorithme de Keccak. Ce changement n'a pas été effectué car :

- Keccak a un chemin de données plus profond que SHA256 (1600 bits contre seulement 256 bits), ce qui signifie une plus grande complexité.
- SHA256 requiert beaucoup moins de ressources matérielles que Keccak, ce qui le rend potentiellement plus adapté à des applications embarquées. On remarque en effet dans les chiffres donnés une multiplication par 10 de l'utilisation de LUTs en utilisant Keccak (10.435 contre 1.031), et l'utilisation de 2,7 fois plus de registres.

Le principal avantage de l'algorithme de Keccak serait ainsi une performance supérieure pour la génération de nombres polynomiaux en comparaison à SHA256. Afin de faire un choix informé sur le choix de l'algorithme le plus adapté, il serait intéressant remettre quelques aspects en perspective :

- La carte de développement utilisée ici (Zynq Ultrascale+ ZCU102) est équipée de plusieurs centaines des milliers de LUTs [4]. Une fois ces deux nombres côte à côte, l'augmentation engendrée par l'utilisation de

4. Voir site de Xilinx : `https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html` : "System Logic Cells (K) : 600"

l'algorithme de Keccak ne semble pas nécessairement conséquente (bien qu'elle ferait passer le nombre total de LUTs par le coeur RISC-V de 53.819 à 63.223 soit une augmentation de 20% environ).

— En prenant en considération cette augmentation de ressources utilisées, il est nécessaire de revenir sur les performances supérieures de l'algorithme de Keccak. De quelle magnitude Keccak est il plus performant que SHA256 ? En regardant la table II de l'article, on constate que l'étape de génération des clés, mobilisant essentiellement le module SHA256, représente environ 25% du temps d'exécution total en nombre de cycles. Une implémentation effectuée par d'autres chercheurs [5] est parvenue à un temps de 13.774 cycles par exécution [6]. Pour rappel, l'étape de génération des clés la plus rapide une fois l'architecture proposée optimisée prenait 542.814 cycles. Même en admettant que les gains soient moindre pour l'application visée, et pour la plate-forme visée, il semble raisonnable d'annoncer que l'utilisation de l'algorithme de Keccak permettrai d'économiser quelques centaines de milliers de cycles sur l'étape de génération des clés soit environ 15% du temps total d'exécution de LAC. Cette estimation optimiste ne prends pas en compte les nombreux accès mémoire supplémentaires induits par le chemin de données très large de cet algorithme [7]. En comparant les gains de performance potentiels apportés par l'algorithme de Keccak à son coût en ressources, il est légitime de questionner l'utilité de son utilisation, surtout dans le cadre d'un déploiement sur un système embarqué. (20% de ressources en plus, pour 15% de performances en plus)

— De même, l'utilisation d'un nombre conséquent de ressources supplémentaires, et l'augmentation du nombre d'accès mémoires requis par l'algorithme pose des questions d'efficacité énergétique. Un papier de recherche comparant l'efficacité de différents algorithmes de chiffrement en établissant un rapport surface / bande passante pour des cibles FPGA Xilinx et Altera démontre que l'algorithme Keccak est beaucoup plus efficace sur ce point, notamment comparé au standard SHA-2 qui inclus SHA256 (on parle de environ 1450 slices pour 13.000 Mb/s avec Keccak pour seulement 4.000 Mb/s pour le même nombre de slices avec SHA256). [8]
Keccak faisant donc preuve d'une plus grande efficacité, on peut supposer qu'à bande passante équivalente, cet algorithme soit plus efficace d'un point de vue énergétique.

Le choix de l'intégration ou non de l'algorithme de Keccak au sein du coeur RISC-V développé dans le cadre de ce projet est difficile. Ce dernier est a la fois plus efficace et plus efficace énergétiquement. Cependant, son implémentation entraînerai une augmentation significative des ressources utilisées, et donc de la surface totale requise sur le silicium. Ces deux critères sont souvent de grande importance dans le milieu des systèmes embarqués auquel se prédestine l'architecture décrite dans cet article, c'est pourquoi on pourrait dire qu'il serait préférable d'utiliser SHA-256.

5 Résultats

La table II de l'article présente un profilage des temps d'exécutions de l'algorithme LAC. Elle y compare l'exécution d'un code de référence, c'est à dire non optimisé sur les plate-formes ARM (Cortex M4) et RISC-V. On peut y voir que si la plate-forme Arm est plus rapide à exécuter l'algorithme de référence que la plate-forme RISC-V, cette dernière récupère un très net avantage une fois optimisée. Il aurait été intéressant de comparer ces résultats post-optimisation à une implémentation utilisant une plate-forme Arm ayant été optimisée.
Enfin, les auteurs ne mentionnent pas la taille du code généré (bien que l'on puisse supposer que le RISC-V soit plus compact a ce niveau, étant donné qu'il a permis la création d'instructions dédiées aux quatre accélérateurs) ni l'emprunte mémoire de leur implémentation.

5. Voir : Efficient Cryptography on the RISC-V Architecture, DOI 10.1007/978-3-030-30530-7_16 pages 8 et 9
6. Sur une plate-forme très différente de la notre (HiFive1 avec SoC FE310-G000), la comparaison est donc difficile
7. Également mentionné dans Efficient Cryptography on the RISC-V Architecture, DOI 10.1007/978-3-030-30530-7_16 page 8
8. Voir : Comprehensive Evaluation of High-Speed and Medium-Speed Implementations of Five SHA-3 Finalists Using Xilinx and Altera FPGAs