Chiffrement homomorphe pour la protection de calculs sur processeur embarqué

Thomas Hiscock
CEA-LETI
Email : thomas.hiscock@cea.fr

***Résumé*—Les processeurs embarqués de part leur proximité avec l'utilisateur se doivent d'offrir une sécurité renforcée. Les attaques utilisant la consommation, le rayonnement électromagnétique ou d'autres canaux auxiliaires s'avèrent être redoutables pour extraire des informations sensibles.**

Ce papier présente une architecture de processeur intégrant du chiffrement homomorphe et capable d'effectuer des opérations sur des données chiffrées. Sous l'hypothèse que quelques composants bien identifiés sont résistants à de telles attaques, l'architecture proposée offre un cadre d'exécution sécurisé pour ses applications.

I. Introduction

Les processeurs embarqués sont plus que jamais proches de l'utilisateur (*Smartphones*, tablettes, cartes à puce, objets connectés, ...). C'est pourquoi la sécurité de ces systèmes doit être pensée en prenant en compte qu'un adversaire a un accès physique au système.

Aujourd'hui un large panel de méthodes permettent de protéger relativement bien les mémoires externes (DRAM, Flash, ...) contre l'observation et la modification [1]. Cependant il n'est pas évident d'étendre cette protection à un processeur qui serait sujet à des attaques par canaux auxiliaires. Ces attaques analysent des propriétés physiques du système à l'exécution (consommation, rayonnement électromagnétique) dans le but de retrouver des informations sensibles comme des clés cryptographiques. Il est évidement possible d'appliquer les contre-mesures connues (masquage, logiques équilibrées, circuits asynchrones, ...) au circuit complet du processeur, mais cela aurait un impact important sur l'architecture, la taille et les performances du circuit.

Dans ce papier je présente une approche très générique à ce problème, basée sur du chiffrement homomorphe pour protéger le stockage et la manipulation des données par le processeur. Les chiffrements homomorphes permettent d'effectuer des calculs sur des données chiffrées, nous verrons comment le processeur peut travailler directement sur des données chiffrées, réduisant ainsi considérablement le risque de fuite d'information. Grâce à cette approche, garantir la sécurité se ramène en grande partie à sécuriser la fonction de déchiffrement du schéma, et ce indépendamment de l'algorithme à protéger. Nous verrons également comment produire une implémentation matérielle plus efficace de cette solution grâce à une procédure de "bootstraping" légère (Section III-C).

A. Modèle des attaques

Nous considérons un processeur embarqué standard (représenté figure 1). L'objectif est de protéger les données critiques qu'il peut être amené à manipuler. Notons que nous ne cherchons pas ici à protéger la confidentialité du programme lui même, ce dernier peut être parfaitement connu (e.g., un AES).

Aussi nous nous plaçons dans le cadre où un attaquant peut :

— Observer toutes les données externes au processeur (RAM, Flash, etc...).

— Effectuer des attaques par observations (du premier ordre) sur le système et en particulier sur le processeur.

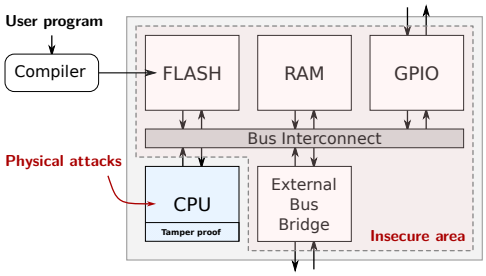

FIGURE 1. Exemple schématique d'un processeur considéré dans ce papier

Les attaques par injection de fautes ne sont pas considérées dans ce papier.

II. État de l'art

Comme évoqué dans l'introduction, on trouve dans la littérature un certain nombre d'architectures permettant de fournir un contexte d'exécution sécurisé aux programmes lorsque les mémoires peuvent être modifiées ou observées. Citons le processeur commercial DS5200 [2], l'architecture XOM [3], AEGIS [4], HIDE ou encore CryptoPage [5].

Cette problématique bénéficie même d'un traitement théorique rigoureux en cryptographie. Goldreich et al. ont montrés [6] qu'à l'aide d'un processeur sécurisé complètement opaque (i.e., seules ses entrées/sorties sont observables ou modifiables), il est possible de protéger n'importe quel programme. Pour cela, toutes les données entrantes et sortantes du processeur sont chiffrées et authentifiées. Le problème se réduit alors à rendre les accès mémoires (les adresses émises) indifférenciables d'un algorithme à un autre. Une machine possédant une telle propriété est appelée *Oblivious RAM* (ORAM), depuis les travaux de Goldreich de nombreux

progrès ont été faits tant pour améliorer les méthodes [7] que pour développer des applications réelles [8].

May et al. [9], [10] ont proposés un certain nombre de contre-mesures possibles à intégrer dans les processeurs pour mitiger les attaques par canaux auxiliaires. Des évaluations de certaines d'entres elles plus quelques nouvelles ont été faites dans [11] et semblent prometteuses. Cependant, il est difficile d'estimer la protection apportée lorsque l'application exécutée n'est pas connue à l'avance.

Plusieurs travaux ont montrés comment obfusquer (au sens théorique [12]) un algorithme et l'exécuter sur du matériel partiellement observable [13], [14]. La solution présentée dans cet article s'inspire clairement de ces travaux. Notons cependant que nous ne cherchons pas à obtenir une propriété aussi forte que l'obfuscation, nous cherchons seulement à protéger les données, la protection de l'algorithme n'est pas nécessaire. Ceci permet de réduire considérablement la complexité de la solution en utilisant "seulement" du chiffrement homomorphe.

III. Chemin de donnée homomorphe

A. Chiffrement homomorphe, brève introduction

Un schéma homomorphe est un chiffrement à clé publique classique, qui possède la propriété très particulière de pouvoir évaluer des fonctions sur des éléments chiffrés. Un tel schéma est défini par les fonctions suivantes :

- $\texttt{Init}(\lambda)$, qui en fonction d'un paramètre de sécurité λ, génère une paire clé publique, clé privée (pk, sk).
- $\texttt{Encrypt}_{pk}(m)$, la fonction de chiffrement prenant en paramètre la clef publique pk
- $\texttt{Decrypt}_{sk}(c)$, la fonction de déchiffrement utilisant la clé privée sk
- $\texttt{Eval}_{pk}(f, c_0, \dots c_n)$, qui évalue une fonction f sur un ensemble de chiffrés. Cette fonction a seulement besoin de la clé publique.

Une fonction est représentée par un circuit, c'est-à-dire un graphe acyclique, dont les nœuds sont annotés avec des opérations. La propriété fondamentale de ces schémas est que l'évaluation sur les données chiffrée est équivalente à une évaluation sur les données en clair. Formellement, si

$$c = \texttt{Eval}_{pk}(f, \texttt{Encrypt}_{pk}(m_0), \cdots, \texttt{Encrypt}_{pk}(m_n))$$

alors

$$\texttt{Decrypt}_{sk}(c) = f(m_0, \cdots, m_n)$$

Si un schéma ne peut évaluer qu'une classe restreinte de fonctions, on parle de schéma «Somewhat Homomorphic» (SHE). C'est le cas de RSA où des chiffrés peuvent être multipliés, si l'on déchiffre on obtient le produit des messages. De même, le schéma de Paillier, est homomorphe pour l'addition. Il existe également des schémas qui peuvent évaluer un grand nombre d'additions, mais très peu de multiplications [15]. À l'opposé un schéma pouvant évaluer une fonction arbitraire sur des données chiffrées est dit complètement homomorphe «Fully Homomorphic» (FHE).

La construction d'un tel schéma serait bien trop longue à détailler ici. Notons que Gentry a construit le premier schéma complètement homomorphe en 2009 [16].

Les schémas complètement homomorphes, les plus efficaces à ce jour sont le BGV [17], FV [15] ou encore le GSW [18].

B. Vue d'ensemble

Supposons que nous ayons à notre disposition un schéma complètement homomorphe. Le programme à protéger est tout d'abord transformé en un circuit équivalent évaluable par le schéma homomorphe. Cette étape effectue plusieurs opérations essentielles :

- Étendre les opérations complexes vers leurs équivalents logiques (transformer une addition arithmétique, en additionneur logique par exemple)
- Supprimer les branchements conditionnels. Chaque branche d'une structure conditionnelle est alors évaluée et le résultat est sélectionné par un multiplexeur (qui peut être implémenté avec des portes logiques élémentaires).
- Dérouler les boucles. Ceci impose que les boucles du programmes soient de taille connue à la compilation.

Une fois le circuit équivalent généré, il peut être évalué de manière homomorphe. Lors de cette évaluation, aucune fuite d'information n'est possible, tous les calculs sont effectués sur des données chiffrées, ils auraient tout à fait pu être confiés à un tiers (co-processeur, cloud, ...). Une fois le calcul effectué, le résultat est déchiffré.

Ainsi, le processeur doit avoir accès à la fonction de déchiffrement et donc implicitement à la clé secrète du schéma. On comprend donc que pour garantir la sécurité de ce schéma, nous devons ajouter les restrictions suivantes :

1) La fonction de déchiffrement doit être résistante aux attaques physiques. Pour cela il est possible d'appliquer une méthode de masquage générique [19] à la fonction de déchiffrement.
2) La fonction de déchiffrement ne doit pas pouvoir être utilisée pour déchiffrer des données arbitraires. La seule manière d'éviter cela est que l'on soit capable de générer une preuve que le calcul du circuit a bien été effectué. Ceci peut être réalisé à l'aide d'argument universels [20].

C. Masquage du premier ordre

Si le schéma homomorphe utilisé supporte l'évaluation de l'opération «XOR», notée $\oplus$, il est possible d'appliquer un masque aléatoire de manière homomorphe. Le masque en question peut provenir d'un générateur de nombre aléatoire, puis être chiffré ou bien être généré de manière homomorphe.

Dans de telles conditions il est possible de mettre en place le schéma présenté figure 2. Les schémas «Somewhat Homomorphic» ne peuvent évaluer qu'un ensemble fini de fonctions, après quoi le résultat n'est plus garantit de déchiffrer vers une valeur correcte. Pour pouvoir évaluer des circuits plus profond, il faut «remettre à jour» périodiquement un chiffré, cette procédure porte le nom de «bootstrapping»[16]

et consiste à effectuer un déchiffrement suivi rechiffrement et ce de manière chiffrée. Cette étape essentielle permet de transformer un schéma «Somewhat Homomorphic» en un schéma «Fully Homomorphic».

Dans ce cas, nous exploitons le fait que l'implémentation matérielle est partiellement sécurisée pour effectuer ce rechiffrement dans le domaine non chiffré. L'ajout d'un masque permet de prévenir les attaques par observations du premier ordre. Ceci permet de réduire considérablement les paramètres du schéma homomorphe, poussé à l'extrême, le schéma peut être paramétré pour évaluer seulement un «XOR» et un «AND».

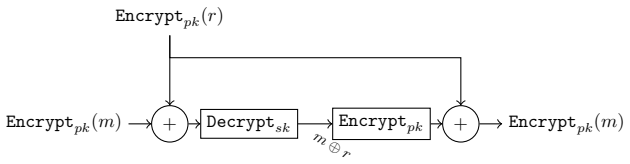

FIGURE 2. Circuit proposé pour réduire le bruit d'un message chiffré par un schéma homomorphe. Un masque aléatoire est appliqué avant de déchiffrer, puis est retiré une fois rechiffré.

D. Aspects Pratiques

J'implémente actuellement cette architecture en langage de description matériel pour une cible FPGA (Altera Arria V). Ceci dans le but mesurer la faisabilité de cette méthode et d'évaluer sa sécurité en pratique. Le schéma homomorphe actuellement utilisé est celui de Fan et Vercauteren [15], basé sur le problème Ring-LWE.

E. Limites

Si cette solution est très intéressante pour sa généricité, elle présente néanmoins quelques inconvénients. Le premier étant que les algorithmes utilisés sont très complexes et impliquent des calculs sur des grands ensembles de données. Par conséquent il n'est pas impossible que le coût de protéger la fonctions de déchiffrement soit en réalité plus important que de protéger un processeur de base avec des techniques de masquage classiques.

Une autre limite est qu'il n'est pas évident d'étendre cette protection à un modèle ou serait prise en compte les attaques par fautes. Notons quand même, les schémas homomorphes présentent par construction une forme de résistance à certaines fautes.

IV. CONCLUSION

À travers cet article nous avons décrit comment intégrer du chiffrement homomorphe dans un processeur. Grâce à cela, tous les calculs impliquant des données critiques peuvent être évalués de manière chiffrée, éliminant toute forme de fuite d'informations.

En contre partie, il faut garantir la sécurité d'un composant bien défini : la fonction de déchiffrement du schéma utilisé. L'avantage majeur de cette méthode est qu'une fois ce composant protégé, n'importe quel algorithme peut être exécuté de manière sécurisée. Ce résultat est à comparer aux contre-mesures classiques [19] qui s'appliquent à un circuit précis et dont la complexité dépend du circuit à protéger.

RÉFÉRENCES

[1] M. Henson and S. Taylor, "Memory encryption : a survey of existing techniques," *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, p. 53, 2014.

[2] D. Semiconductor. DS5002FP secure microprocessor chip. [Online]. Available : https ://datasheets.maximintegrated.com/en/ds/DS5002-DS5002FP.pdf

[3] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz, "Architectural support for copy and tamper resistant software," *ACM SIGPLAN Notices*, vol. 35, no. 11, pp. 168–177, 2000.

[4] G. E. Suh, C. W. O'Donnell, and S. Devadas, "AEGIS : A single-chip secure processor," *Information Security Technical Report*, vol. 10, no. 2, pp. 63–73, 2005.

[5] G. Duc and R. Keryell, "Cryptopage : An efficient secure architecture with memory encryption, integrity and information leakage protection," in *Computer Security Applications Conference, 2006. ACSAC'06. 22nd Annual*. IEEE, 2006, pp. 483–492.

[6] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious RAMs," *Journal of the ACM (JACM)*, vol. 43, no. 3, pp. 431–473, 1996.

[7] E. Stefanov, M. Van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path oram : an extremely simple oblivious ram protocol," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 299–310.

[8] M. Maas, E. Love, E. Stefanov, M. Tiwari, E. Shi, K. Asanovic, J. Kubiatowicz, and D. Song, "Phantom : Practical oblivious computation in a secure processor," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 311–324.

[9] D. May, H. L. Muller, and N. P. Smart, "Non-deterministic processors," in *Information Security and Privacy*. Springer, 2001, pp. 115–129.

[10] ——, "Random register renaming to foil dpa," in *Cryptographic Hardware and Embedded Systems—CHES 2001*. Springer, 2001, pp. 28–38.

[11] F. Bruguier, P. Benoit, L. Torres, L. Barthe, M. Bourree, and V. Lomne, "Cost-effective design strategies for securing embedded processors," 2015.

[12] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang, "On the (im) possibility of obfuscating programs," in *Annual International Cryptology Conference*. Springer, 2001, pp. 1–18.

[13] N. Bitansky, R. Canetti, S. Goldwasser, S. Halevi, Y. T. Kalai, and G. N. Rothblum, "Program obfuscation with leaky hardware," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2011, pp. 722–739.

[14] S. Goldwasser and G. N. Rothblum, "How to compute in the presence of leakage," *SIAM Journal on Computing*, vol. 44, no. 5, pp. 1480–1549, 2015.

[15] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption." *IACR Cryptology ePrint Archive*, vol. 2012, p. 144, 2012.

[16] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, 2009, crypto.stanford.edu/craig.

[17] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. ACM, 2012, pp. 309–325.

[18] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors : Conceptually-simpler, asymptotically-faster, attribute-based," in *Advances in Cryptology–CRYPTO 2013*. Springer, 2013, pp. 75–92.

[19] Y. Ishai, A. Sahai, and D. Wagner, "Private circuits : Securing hardware against probing attacks," in *Annual International Cryptology Conference*. Springer, 2003, pp. 463–481.

[20] B. Barak and O. Goldreich, "Universal arguments and their applications," in *Computational Complexity, 2002. Proceedings. 17th IEEE Annual Conference on*. IEEE, 2002, pp. 194–203.