Titre : Étude de l'impact de la micro-architecture dans l'analyse de fuites d'un code masqué sur le processeur Arm Cortex M3

Durée : 5 à 6 mois

Équipe d'accueil : Laboratoire LIP6, équipe Alsoc

Encadrants : Quentin Meunier, Karine Heydemann

Contexte

Les attaques par canaux auxiliaires (SCA) consistent à mesurer des grandeurs physiques d'un système informatique (consommation, radiations électro-magnétiques) durant l'exécution d'un algorithme cryptographique, et de les utiliser afin de retrouver la clé de chiffrement [1, 2]. Ces attaques ciblent plus particulièrement les dispositifs embarqués, tels que les cartes de paiement, pour lesquels la clé de chiffrement ne doit pas être connue de l'utilisateur (ce qui n'est en général pas le cas sur les machines de bureau ou serveurs).

Les SCA n'ont cessé de gagner en popularité depuis le début des années 2000. Pour limiter leur impact, deux techniques principales sont apparues : le "hiding" [3] et le masquage [4]. Le "hiding" consiste à ajouter du bruit dans le code exécuté (exécution d'instructions inutiles, des instructions dans un ordre aléatoire, etc.). Le masquage consiste quant à lui à décomposer le secret en plusieurs parties (shares), dont seule la recomposition permet de déduire de l'information, et d'appliquer un traitement indépendant à chacune des parties.

Différents travaux se sont intéressés à prouver à l'aide d'outils de tels schémas de masquage : il s'agit, à partir d'une description du programme, de vérifier pour toutes les expressions obtenues que leurs distributions sont statistiquement indépendantes des secrets (typiquement de la clé de chiffrement) [5, 6]. Cela se base sur l'hypothèse que la valeur observée de la fuite physique est liée à la valeur de l'expression manipulée à ce moment. Cela peut être pertinent dans certains cas, mais il apparait que bien souvent un modèle plus pertinent est la transition entre deux valeurs consécutives. Les méthodes existantes se sont alors adaptées en conséquence, en considérant les transitions entre variables.

Néanmoins, si l'on revient à la source de cette fuite, c'est-à-dire à l'origine de cette consommation énergétique, on voit qu'elle est liée au changement d'état des transistors du processeur. Considérer des transition entre les variables n'a donc qu'un intérêt limité, puisque c'est entre les registres qu'il faut considérer les transitions. Le travail [7] a adopté cette approche, mais c'est à ce jour un des seuls car il nécessite de partir du code assembleur de l'application, puisque l'allocation des registres n'est connue qu'à ce moment (une même variable peut être chargée dans deux registres différents, et un même registre peut servir au chargement consécutifs de variables différentes).

Si cette approche met en évidence des fuites plus proches de la réalité, elle n'est malgré tout pas totalement satisfaisante, car elle suppose que les registres du processeurs sont limités à ceux manipulables par le logiciel assembleur (banc de registres, ou registres généraux). Or, les architectures de processeur non-triviales (plus précisément dès qu'il y a présence d'un pipeline) possèdent des registres "cachés", faisant partie de la micro-architecture du processeur mais qu'il est impossible de considérer au niveau assembleur.

Réaliser une analyse de fuites d'un code masqué en vue d'en garantir l'absence de fuite ne peut donc pas uniquement se faire sur le code, mais doit aussi prendre en compte l'architecture du processeur [8].

Sujet du stage

Dans l'objectif de proposer une telle analyse tenant compte de la micro-architecture, nous avons récemment modélisé le pipeline du processeur Arm Cortex M3 à partir d'une description Verilog. Cette modélisation a permis d'identifier des sources potentielles de fuite en lien avec la micro-architecture. Des expérimentations basées sur des vecteurs de tests (séquences d'instructions assembleur spécifiques) sur une réalisation physique du processeur ont permis de caractériser ces fuites à l'aide de tests statistiques. Cela a permis par ailleurs de valider la modélisation réalisé et un prototype d'analyse de code binaire intègre cette modélisation. Ce prototype est capable d'indiquer des séquences d'instructions potentiellement vulnérables dans un code binaire masqué. Toutefois, nous n'avons pas démontré que les vulnérabilités trouvées corrrespondent bien à des fuites d'une part et qu'elles sont exploitables pour mener des attaques sur des codes réels d'autre part.

L'objectif du stage est donc en premier lieu de :
1. Mettre en évidence les fuites de valeurs secrètes sur des codes masqués réels à l'aide de tests statistiques
2. Monter, sur ces codes, des attaques exploitant des registres internes à la micro-architectures
3. Patcher ces codes à l'aide du prototype d'analyse pour supprimer toutes les fuites trouvées dans celui-ci, et vérifier expérimentalement que l'on n'observe plus de fuite sur le processeur réel.

À l'issue de ce travail, il est envisagé de valoriser celui-ci par la rédaction et la soumission d'un article dans une conférence du domaine.

Par la suite, plusieurs pistes de travail sont possibles :
- Réaliser une comparaison expérimentale des fuites entre le Cortex-M3 et le Cortex-M4 pour les codes masqués réels
- Étudier l'architecture du Cortex-M4 et réaliser une modélisation de celui-ci dans le protoype d'analyse
- Analyser l'adéquation entre plusieurs modèles de fuite théoriques et des fuites réelles

Il est à noter qu'une poursuite en thèse est possible à l'issue du stage si les résultats sont satisfaisants. Ces travaux s'inscrivent dans le cadre d'un projet collaboratif ANR qui démarre debut 2021.

Pour atteindre l'objectif principal et se familiariser avec le sujet du stage et l'environnement de l'étude à mener, les étapes préalables suivantes seront mises en oeuvre :

- Étude bibliographique sur le masquage et les fuites
- Étude de l'architecture du processeur Arm Cortex M3
- Prise en main de la carte de mesure et de l'outil d'analyse de code binaire développé dans l'équipe

Compétences souhaitées

Le candidat devra avoir une bonne connaissance des sujets suivants :
- Architecture des processeur, notamment pipeline
- Jeux d'instructions assembleur (connaitre le ARM est un plus)
- Notions de sécurité informatique générales
- Attaques par canaux auxiliaires : DPA, CPA
- Maitrise du langage python
- Connaissances de base en statistiques

Le stage étant à la croisée de plusieurs domaines, le candidat devra être capable de se former sur les aspects ne relevant pas de sa spécialité.

Références

[1] S. Mangard, E. Oswald, and T. Popp. Power analysis attacks: Revealing the secrets of smart cards,Vol. 31. Springer Science & Business Media, 2008.

[2] Q. L. Meunier. FastCPA: Efficient Correlation Power Analysis Computation with a Large Number of Traces, in CS2'19, 2019, Valencia, Spain

[3] N. Belleville, D. Couroussé, K. Heydemann and H.-P. Charles. Automated software protection for the masses against side-channel attacks, ACM Transactions on Architecture and Code Optimization (TACO) 15.4 (2018): 47.

[4] Y. Ishai, A. Sahai, D. Wagner. Private circuits: Securing hardware against probing attacks, in Annual International Cryptology Conference. Springer, Berlin, Heidelberg, 2003.

[5] G. Barthe, S. Belaïd, G. Cassiers, P.-A. Fouque, B. Grégoire, F.-X. Standaert. maskVerif: Automated Verification of Higher-Order Masking in Presence of Physical Defaults, European Symposium on Research in Computer Security. Springer, Cham, 2019.

[6] Q. L. Meunier, I. Ben El Ouahma, and K. Heydemann. SELA: a Symbolic Expression Leakage Analyzer. International Workshop on Security Proofs for Embedded Systems. 2020.

[7] I. Ben El Ouahma, Q. L. Meunier, K. Heydemann, E. Encrenaz. Side-channel robustness analysis of masked assembly codes using a symbolic approach, Journal of Cryptographic Engineering (JCEN), March 2019

[8] D. Zoni, A. Barenghi, G. Pelosi, W. Fornaciari. A Comprehensive Side-Channel Information Leakage Analysis of an In-Order RISC CPU Microarchitecture, ACM Transactions on Design Automation of Electronic Systems (TODAES) 23.5 (2018): 57.