Attestation à distance de microprocesseurs vérifiée formellement

Jonathan CERTES, Benoît MORGAN
Institut de Recherche en Informatique de Toulouse (IRIT) - email : *prenom.nom*@irit.fr

***Abstract*—L'attestation à distance permet de s'assurer qu'une machine distante possède des propriétés de sécurité [1] et rend possible l'établissement d'une racine de confiance dynamique. Les architectures modernes d'attestation à distance allient primitives de signature symétrique avec du matériel qui assure la confidentialité du secret.**

Nos travaux se basent sur une méthode d'attestation à distance co-conçue et prouvée, VRASED [2]. Sa preuve est établie à l'aide de méthodes formelles et son implémentation est réalisée sur une architecture simple de type microcontrôleur. Nous proposons d'étendre cette méthode aux microprocesseurs.

I. Introduction

L'attestation à distance consiste à vérifier si une machine *prover* possède, à un instant donné, les propriétés de sécurité acceptables pour une machine de confiance distante appelée *verifier* [1]. Ces propriétés de sécurité sont généralement vérifiées à l'aide d'un protocole question-réponse comme illustré sur la figure 1.

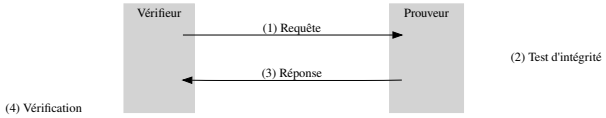

Fig. 1. Protocole d'attestation à distance [2]

Dans un premier temps, le *verifier* demande l'attestation au *prover* (1). Le *prover* calcule une preuve d'intégrité sur sa mémoire et son environnement (2) et retourne au *verifier* le résultat (3). A partir du résultat, le *verifier* s'assure alors que l'état du *prover* est valide (4).

Les premiers travaux d'attestation à distance de processeurs complexes utilisent seulement du logiciel et basent l'authenticité de la réponse sur un temps d'exécution attendu [3], [4]. Des travaux plus récents utilisent la cryptographie pour mesurer l'authenticité de la réponse [5], [6]. Cette méthode implique le maintien d'un secret et nécessite donc du support matériel pour les contrôles d'accès. Les constructeurs de matériels ont apporté des solutions d'environnement d'exécution sécurisée, ou TrEE (*Trusted Execution Environments*), qui permettent l'établissement d'une racine de confiance statique, par exemple au démarrage du système. Parmi ces solutions, nous pouvons citer *SGX* ou *Trustzone*. L'attestation à distance ne se limite pas à leur simple utilisation car elle rend possible l'établissement d'une racine de confiance dynamique : i.e. au *run-time*. Le *verifier* peut donc s'assurer de l'intégrité du *prover* à un instant donné, par exemple avant la transmission de données sensibles.

L'approche formelle apporte une réponse aux exigences de sécurité de l'attestation à distance. Elle permet l'établissement d'une preuve, basée sur des axiomes ou des propriétés démontrées, de la sécurité du système et de son implémentation. La vérification formelle s'effectue généralement en trois étapes. D'abord, le système (logiciel ou matériel) est modélisé, par exemple sous forme d'un automate. Ensuite, les propriétés que doit satisfaire le système sont formellement énoncées, entre autres la confidentialité du secret dans le cas de l'attestation à distance. Finalement, une preuve ou du *model-checking* démontre que le système vérifie les propriétés.

La section II fait un récapitulatif de l'état de l'art sur l'attestation à distance, généralement implémentée sur une architecture simple, et sa vérification formelle. Une approche d'extension aux microprocesseurs est décrite dans la section III et un exemple de vérification formelle de propriétés est donné dans la section IV.

II. Etat de l'art

Eldefrawy et al. proposent *SMART* [5], une modification matérielle pour le *prover*, sur un microcontrôleur Texas Instrument MSP430. Celle-ci définit une région mémoire protégée où la clé est stockée. Lugou et al. [7] ont tenté de proposer une méthode unifiée de vérification d'architectures de sécurité co-conçues. Ils ont appliqué cette méthode sur *SMART* et modélisé leur système avec *Proverif*. Cette méthode ne peut passer à l'échelle et comporte des imprécisions comme expliqué par Eldefrawy et al. dans [8]. Dans ces travaux, les propriétés de sécurité sont assurées par une extension matérielle moniteur, capable de redémarrer le système en cas de (future) compromission d'un secret. La sécurité est garantie par du *model-checking*.

De Oliveira Nunes et al. [2] proposent VRASED (*Verifiable Remote Attestation for Simple Embedded Devices*), une méthode d'attestation à distance co-conçue et prouvée. L'implémentation est également réalisée sur un microcontrôleur MSP430. Une modification lourde du cœur du processeur fournit des points d'entrée, tels que le pointeur d'instruction ou les signaux d'interruption, à une extension matérielle.

L'approche proposée par VRASED nécessite l'implémentation d'un logiciel attestant accompagné d'un module matériel garantissant les propriétés de sécurité. Le

logiciel est basé sur une bibliothèque de cryptographie certifiée (fonctionnement correct, sans RTE, résistance à certains *side-channels*, etc.), qui calcule le *HMAC* d'une région à attester à partir d'un secret partagé et stocke le résultat dans un espace mémoire dédié. Le module matériel assure le contrôle d'accès au secret ainsi que l'immuabilité et l'atomicité de l'exécution du logiciel attestant. Le modèle de l'attaquant est le suivant : l'attaquant peut modifier entièrement le code et l'environnement du *prover* lorsque ce n'est pas explicitement protégé par le matériel.

Les étapes suivantes décrivent l'obtention de la preuve de *soundness* et sécurité :

1) La *soundness* et la sécurité sont spécifiées sous forme d'invariants. Parmi ceux-ci :
   - Tout accès direct au secret ne peut être effectué par du code autre que le logiciel attestant.
   - Tout espace mémoire écrit par le logiciel attestant ne peut être lu par l'attaquant.
2) Des hypothèses sont exprimées dans le même formalisme que les invariants. Parmi celles-ci :
   - Tant que les espaces mémoire stockant des données utilisées par le code attestant ne varient pas durant son exécution, le calcul du *HMAC* est correct (conformité fonctionnelle du code).
   - Le code attestant ne peut être modifié par un attaquant (immuabilité du code).
   - L'exécution du code attestant ne peut être interrompue et se fait de la première instruction à la dernière (atomicité du code).
3) La preuve de *soundness* et sécurité est obtenue par réécriture : i.e. lorsque l'implication des invariants par la conjonction des hypothèses est une tautologie.
4) Le *model-checking* valide que le système vérifie les hypothèses. Un outil de traduction [9] transforme la description en langage matériel du système en un automate sur lequel les propriétés sont vérifiées.

Cette méthode de conception de système d'attestation à distance prouvée est adaptée aux microcontrôleurs simples. Notre objectif est de réutiliser au mieux ses preuves de *soundness* et sécurité pour l'appliquer aux microprocesseurs.

III. Approche d'extension aux microprocesseurs

Nous souhaitons étendre l'approche proposée par VRASED aux microprocesseurs. A moins de travailler sur une architecture *open-source*, il est impossible d'effectuer des modifications lourdes du cœur. Des observations équivalentes doivent donc être trouvées pour obtenir des points d'entrée tels que la valeur du pointeur d'instruction ou l'information qu'une interruption est levée.

A. Environnement

Des *systems on chip* (*SoC*) modernes, tels que le Xilinx Zynq-7000, proposent des microprocesseurs ARM étroitement intégrés avec un circuit logique programmable FPGA et fournissent un environnement de travail adapté. Une ségrégation des espaces mémoire dans la partie logique programmable du *SoC*, alliée à un module matériel qui contrôle les accès, permet de garantir des propriétés de sécurité.

Les microprocesseurs ARM possèdent une interface de *debug* nommée *coresight* qui permet de réaliser des traces : une collecte non-invasive de données retraçant l'activité du microprocesseur sans le ralentir. Lors de l'exécution du logiciel attestant, l'activation des traces et des instructions spécifiques fournissent les points d'entrée nécessaires à l'obtention des propriétés de sécurité. En particulier, le module PTM (*Program Flow Trace Macrocell*) de *coresight* génère des traces lors d'une interruption ou d'un branchement indirect, en fournissant l'adresse de destination.

Pour garantir les propriétés d'immuabilité et d'atomicité, la présence d'un circuit logique sur FPGA est donc couplée avec l'utilisation de l'interface de *debug*. Une trace est générée lorsqu'une interruption est levée durant l'exécution du logiciel attestant. Les première et dernière instructions sont choisies de telle sorte qu'une trace soit générée et contienne la valeur du pointeur d'instruction : un branchement indirect qui a pour destination l'adresse de l'instruction suivante.

B. Approche descendante

Une extension aux microprocesseurs apporte des contraintes supplémentaires et augmente la surface d'attaque. Pour s'y adapter, une approche descendante est considérée.

Dans un premier temps, on considère les microprocesseurs où l'on fait abstraction des mémoires cache, de l'unité de gestion mémoire (MMU) et de la capacité de l'attaquant de modifier la configuration de *coresight*. Ce système à prouver est appelé le modèle 0 : il est basé sur celui décrit dans VRASED et sa définition de la sécurité est identique à celle qui y est énoncée ; la *soundness* est à prouver.

Ensuite, une contrainte supplémentaire est ajoutée à la définition de la sécurité ; par exemple : "la configuration de *coresight* n'est pas modifiable par l'attaquant". Un invariant supplémentaire est ajouté à la spécification. Le système doit alors vérifier de nouvelles hypothèses pour que la preuve reste valide et l'appellation du modèle est incrémentée. Deux approches distinctes sont alors considérées :

- Soit le nouveau système est également traduit en automate et la vérification des hypothèses (anciennes et nouvelles) se fait par *model-checking* : l'approche est alors identique à celle proposée par VRASED. Cette méthode est adaptée aux systèmes de taille réduite.
- Soit une relation de simulation est établie : si tout état du modèle précédent est simulé par le nouveau modèle, alors le nouveau modèle profite de sa preuve de sécurité [10]. Seuls les nouveaux invariants restent à prouver. Cette méthode est adaptée au cas où une explosion d'état apparaît lors du *model-checking*.

Ces opérations sont répétées pour toutes les contraintes apportées par une extension aux microprocesseurs. A chaque itération, le système est concrétisé et la preuve est établie.

IV. Exemple de vérification de propriétés

Cette section décrit la vérification de propriétés sur le modèle 0 par *model-checking*. La spécification de la sécurité est identique à celle énoncée par VRASED mais la valeur du pointeur d'instruction et les signaux d'interruption ne sont pas accessibles.

VRASED propose un automate qui vérifie des propriétés permettant de prouver l'atomicité du code attestant [2]. Les étiquettes des transitions sont exprimées en fonction des booléens suivants :

- $(pc = 0)$: l'instruction exécutée est à l'adresse de démarrage ;
- $(pc = CR_{min})$: l'instruction exécutée est la première du code attestant ;
- $(pc = CR_{max})$: l'instruction exécutée est la dernière du code attestant ;
- $(pc > CR_{min} \wedge pc < CR_{max})$, ou $(pc \in CR)$: l'instruction exécutée est au milieu du code attestant ;
- irq : une interruption matérielle est levée

L'utilisation de *coresight* et d'instructions choisies dans le code attestant fournit les propositions élémentaires suivantes :

- $traceEN$: les traces sont activées, i.e. le pointeur d'instruction du microprocesseur contient une adresse dans le code attestant.
- $hasTrace$: une trace est générée par *coresight*. Ceci est causé par une interruption ou un branchement indirect, en première et dernière instruction du code attestant.
- $@CR_{min}$: la première instruction du code attestant est exécutée.
- $@CR_{max}$: la dernière instruction du code attestant est exécutée.

L'automate représenté sur la figure 2, un transducteur, accepte ces propositions élémentaires : il les transforme en un langage acceptable par l'automate de VRASED. Les étiquettes des transitions de ce dernier doivent parfois être adaptées.

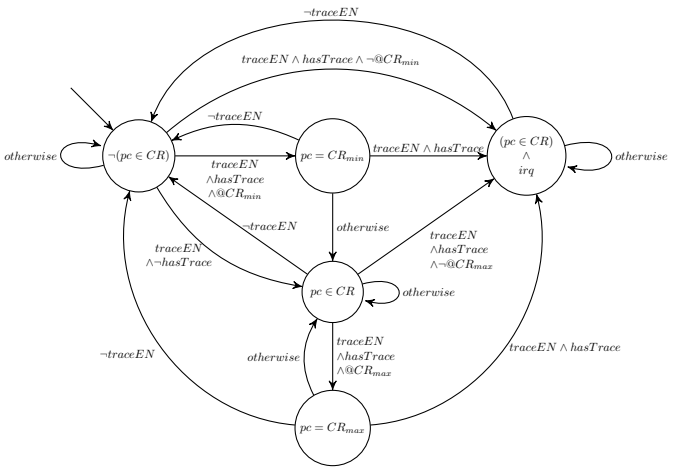

Fig. 2. Transducteur *coresight* vers VRASED

On réalise ensuite le produit synchronisé des deux automates : les transitions de l'automate de VRASED sont gardées par les changements d'états du transducteur. Les propriétés garantissant l'atomicité sont exprimées dans [2], elles sont vérifiées sur ce produit par *model-checking*.

La *soundness* de l'atomicité est également exprimée et vérifiée. Elle garantit qu'une exécution où le pointeur d'instruction entre par la première instruction dans le code attestant en sort dans le futur, si aucune attaque n'est détectée, par la dernière.

V. Conclusion

Les méthodes d'attestation à distance modernes allient primitives de signature avec du matériel qui assure la confidentialité du secret. VRASED propose une méthode d'attestation à distance co-conçue et prouvée, qui se base sur des méthodes formelles, dont l'implémentation est réalisée sur une architecture simple de type microcontrôleur.

Notre approche d'extension aux microprocesseurs ARM est la suivante : l'interface de *debug coresight* permet une collecte non-invasive de données retraçant l'activité du processeur. Alliée à un automate dédié, elle fournit des observations équivalentes. Une approche descendante permet de s'adapter aux concrétisations successives du système et du modèle de l'attaquant.

Dans le cas d'un système complexe où le *model-checking* engendre une explosion d'états, démontrer une simulation d'automate est envisagé pour obtenir la preuve que le système vérifie ses spécifications.

References

[1] G. Coker, J. D. Guttman, P. Loscocco, A. L. Herzog, J. K. Millen, B. O'Hanlon, J. D. Ramsdell, A. Segall, J. Sheehy, and B. T. Sniffen, "Principles of remote attestation," *Int. J. Inf. Sec.*, vol. 10, no. 2, pp. 63–81, 2011.

[2] I. D. O. Nunes, K. Eldefrawy, N. Rattanavipanon, M. Steiner, and G. Tsudik, "VRASED: A verified hardware/software co-design for remote attestation," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 1429–1446.

[3] A. Seshadri, A. Perrig, L. van Doorn, and P. K. Khosla, "SWATT: software-based attestation for embedded devices," in *2004 IEEE Symposium on Security and Privacy (S&P 2004), 9-12 May 2004, Berkeley, CA, USA*, 2004, p. 272.

[4] A. Seshadri, M. Luk, A. Perrig, L. van Doom, and P. K. Khosla, "Pioneer: Verifying code integrity and enforcing untampered code execution on legacy systems," in *Malware Detection*, 2007, pp. 253–289.

[5] K. Eldefrawy, G. Tsudik, A. Francillon, and D. Perito, "Smart: Secure and minimal architecture for (establishing dynamic) root of trust." in *NDSS*, vol. 12, 2012, pp. 1–15.

[6] J. Noorman, J. V. Bulck, J. T. Mühlberg, F. Piessens, P. Maene, B. Preneel, I. Verbauwhede, J. Götzfried, T. Müller, and F. C. Freiling, "Sancus 2.0: A low-cost security architecture for iot devices," *ACM Trans. Priv. Secur.*, vol. 20, no. 3, pp. 7:1–7:33, 2017.

[7] F. Lugou, L. Apvrille, and A. Francillon, "Toward a methodology for unified verification of hardware/software co-designs," *Journal of Cryptographic Engineering*, pp. 1–12, 2016.

[8] K. Eldefrawy, I. O. Nunes, N. Rattanavipanon, M. Steiner, and G. Tsudik, "Formally verified hardware/software co-design for remote attestation," *arXiv preprint arXiv:1811.00175*, 2018.

[9] A. Irfan, A. Cimatti, A. Griggio, M. Roveri, and R. Sebastiani, "Verilog2smv: A tool for word-level verification," in *2016 Design, Automation & Test in Europe Conference & Exhibition, DATE 2016, Dresden, Germany, March 14-18, 2016*, 2016, pp. 1156–1159.

[10] R. Milner, "An algebraic definition of simulation between programs," in *Proceedings of the 2nd International Joint Conference on Artificial Intelligence. London, UK, September 1-3, 1971*, 1971, pp. 481–489.