Analyse d'un problème posé par Intel SGX : la communication sécurisée entre une enclave et un périphérique

Florian Lugou
Ingénieur chez Prove & Run – florian.lugou@provenrun.com

Ludovic Apvrille
Chercheur chez Telecom ParisTech – ludovic.apvrille@telecom-paristech.fr

Nous abordons ici une des questions posées en marge du modèle implémenté par Intel SGX : celle de la sécurisation de la communication entre une enclave et un périphérique.

Mots clés : ENVIRONNEMENT D'EXÉCUTION SÉCURISÉ / DÉVELOPPEMENT SÉCURISÉ / JEU D'INSTRUCTIONS

Dans le numéro 99 de MISC, Alexandre Adamski vous présentait la technologie *Intel Software Guard eXtensions (SGX)*. L'introduction de cette extension architecturale dans des processeurs disponibles sur le marché s'inscrit dans un processus plus global de réflexion critique de la communauté scientifique autour de la question de l'isolation des applications s'exécutant dans un contexte potentiellement malveillant. Le modèle de haut-niveau implémenté par Intel SGX a le premier mérite de proposer aux développeurs d'applications une délimitation simple et précise entre les éléments logiciels dont l'isolation est garantie par l'architecture, et ceux dont le développeur doit supposer qu'ils peuvent être corrompus.

On peut penser que la présentation d'un tel modèle, supporté par une implémentation s'y conformant est un pas important sur le chemin de la sécurisation des applications, en particulier dans les domaines où les applications et l'environnement matériel appartiennent à différentes entités (dans le modèle des *Infrastructure as a Service* par exemple). Il faut cependant garder à l'esprit que cette technologie pose toujours des questions quant à la validité du modèle. Nous proposons dans cet article de vous faire partager des réflexions autour d'une question particulière soulevée par Intel SGX : celle de la communication entre une application sécurisée (enclave) et un périphérique, que ce soit un clavier, une carte réseau, une carte graphique, un FPGA (Field-Programmable Gate Array) ou tout autre.

1 Intel SGX

Afin de permettre une lecture autonome de cet article, nous commençons par présenter les objectifs et mécanismes d'Intel SGX. Cette présentation se focalise sur les thèmes abordés dans le reste de l'article de façon à limiter les redondances avec le numéro précédent, auquel nous renvoyons les lecteurs intéressés par plus de détails.

1.1 Modèle de haut-niveau

D'un point de vue logiciel, l'objectif d'Intel SGX est de permettre l'isolation de certains éléments du reste de la pile logicielle, laquelle se compose des autres applications utilisateurs, du système d'exploitation sous-jacent et de l'hyperviseur lorsqu'il existe. En pratique, cela implique en particulier que l'architecture SGX est capable de garantir l'isolation d'un élément logiciel, même lorsque le système d'exploitation est malveillant ou corrompu (par exemple par un malware).

L'isolation ainsi garantie par Intel SGX consiste en la confidentialité des données et du code de l'application et en leur intégrité : l'enclave peut manipuler des données secrètes et fournir une preuve que ces données n'ont pas été modifiées par le système d'exploitation. D'autres mécanismes complètent ces capacités d'isolation en permettant à une enclave de *sceller* une donnée, c'est-à-dire de la stocker de manière sécurisée afin de l'utiliser lors d'une prochaine exécution de l'enclave. Certains proposent également une méthode pour que l'enclave puisse fournir à une application tierce une preuve cryptographique de son intégrité (opération aussi appelée attestation).

La spécificité d'Intel SGX est de rendre possible cette isolation, tout en conservant un modèle traditionnel où le système d'exploitation a l'exclusivité de l'allocation des ressources matérielles aux différentes applications. D'une part, le système d'exploitation peut allouer plus ou moins de cycles processeur à l'enclave, mettre certaines de ses pages mémoire en cache et transmettre les différents signaux matériels vers l'application de son choix. D'autre part, il ne peut pas savoir comment ces ressources allouées à l'enclave sont utilisées par elle. L'efficacité du modèle implémenté par Intel SGX tient en grande partie à la clarté de cette séparation entre responsabilité logistique du noyau et opérations réalisées par l'enclave.

1.2 Les aspects architecturaux

Le modèle de haut-niveau ainsi ébauché est implémenté par Intel SGX au moyen de modifications architecturales que l'on peut séparer grossièrement en deux classes. D'une part, le processeur et la *Memory Management Unit (MMU)* sont modifiés afin d'assurer à l'enclave un accès exclusif à son espace mémoire. Lorsqu'une adresse virtuelle doit être résolue, la MMU vérifie que l'adresse physique résultante appartient bien à la même enclave qui en a fait la requête. Pour cela, à chaque page d'une enclave est associée une entrée dans une partie de la mémoire appelée *Enclave Page Cache Map (EPCM)* qui est accessible seulement au processeur. D'autre part, de nouvelles instructions sont ajoutées au processeur afin que les applications puissent interagir avec Intel SGX : créer de nouvelles enclaves, appeler des fonctions implémentées par une enclave, procéder à une attestation, etc.

Comme on le voit, la validité du modèle d'Intel SGX est assurée par un ensemble de fonctions implémentées par le processeur et certains des éléments matériels qui l'assistent. Cela signifie qu'un attaquant contrôlant ces modules matériels est en mesure de contourner les protections apportées par Intel SGX. Les attaques matérielles sont donc en partie exclues du modèle d'attaquant. Il existe cependant des cas d'utilisation où il est pertinent d'envisager la possibilité d'une attaque physique visant à compromettre les données d'une enclave. C'est par exemple le cas lorsque la gestion d'un serveur physique est déléguée à une entité différente de celle possédant l'enclave s'exécutant sur le serveur. Afin de limiter le matériel qu'un utilisateur d'Intel SGX doit supposer inviolable, une autre modification matérielle s'ajoute à celles mentionnées au paragraphe précédent : le bus entre le processeur et la mémoire externe est protégé par des mécanismes de cryptographie afin d'empêcher les fuites de données et les modifications malveillantes. Cette fonctionnalité, implémentée par une extension au contrôleur mémoire interne au processeur (appelée *MEE* pour *Memory Encryption Engine*), permet que les données appartenant à une enclave soient automatiquement chiffrées lorsqu'elles quittent le processeur pour être stockées dans la mémoire physique externe.

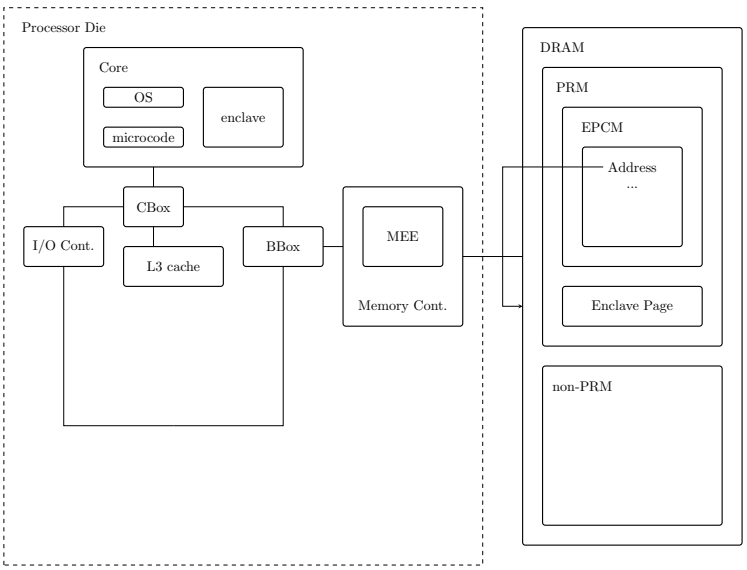

Fig. 1 : Représentation schématique de l'architecture Intel SGX

1.3 Les types d'attaques et d'attaquants

Les modifications matérielles présentées précédemment permettent de décrire un modèle d'attaquant dont les capacités sont clairement définies. D'un point de vue logiciel, Intel SGX suppose qu'un attaquant contrôle l'ensemble des éléments excepté l'enclave. Cela englobe les éléments privilégiés (noyau et hyperviseur), les applications utilisateur (même lorsqu'elles partagent en partie leur espace d'adressage avec l'enclave) et les autres enclaves. À des fins de complétude, il faut mentionner que certaines étapes de la vie d'une enclave (le chargement et l'attestation) nécessitent la coopération d'autres enclaves fournies par Intel, auxquelles il est donc nécessaire de faire confiance.

D'un point de vue matériel, un attaquant est supposé contrôler l'ensemble de l'environnement autour du die du processeur. En particulier, un attaquant peut espionner le bus mémoire, voire remplacer la mémoire par un module matériel qu'il contrôle, espionner et modifier l'ensemble des communications avec les périphériques ou encore intercepter les échanges entre différents processeurs.

Finalement, Intel SGX ne protège pas contre les attaques par canaux auxiliaires quels qu'ils soient. Il a été en particulier montré dans plusieurs travaux qu'Intel SGX est vulnérable à des attaques par canaux auxiliaires basées sur les mécanismes de cache (telles que Spectre dans [2]).

1.4 Difficultés

Intel SGX protège donc une enclave contre un attaquant particulièrement puissant. Ce modèle d'attaquant impose cependant des contraintes à l'enclave qu'elle doit respecter afin qu'Intel SGX puisse garantir son isolation. Outre la protection contre les fuites par canaux auxiliaires mentionnée précédemment, une autre limitation vient du fait que l'enclave ne peut pas faire confiance aux données fournies par le noyau, en particulier dans le cas d'un appel système (un type d'attaque dénommé *Iago*). Finalement, et c'est sur cette difficulté que nous nous focaliserons dans le reste de cet article, l'enclave doit assurer elle-même la sécurité des données qu'elle doit échanger avec des périphériques.

2. Problèmes de sécurité entre une enclave SGX et un périphérique

La question des périphériques peut sembler anecdotique. Plusieurs scénarios cependant justifient l'intérêt que nous y portons. L'isolation garantie nativement par Intel SGX impose une limite forte sur les capacités de l'attaquant à manipuler un périphérique. Dans cette section, nous discutons du type d'applications pour lesquelles ce déséquilibre entre fortes garanties vis-à-vis de la mémoire et faibles garanties vis-à-vis des périphériques se fait particulièrement sentir.

2.1 Scénarios de communication

Un premier exemple est donné par l'application de gestion de mots de passe qui vous a été présentée dans l'article «Développer une application sécurisée avec Intel SGX » du numéro 99. Dans cette application, une enclave est utilisée afin de stocker des mots de passe qui peuvent être récupérés ultérieurement en fournissant à l'enclave un mot de passe générique permettant d'en débloquer l'accès. Si l'on garde à l'esprit le modèle d'attaquant supposé par Intel SGX, on comprend que **la sécurité apportée par Intel SGX peut être aisément contournée au moyen d'un *keylogger* implémenté soit dans le logiciel du pilote du clavier**, soit en matériel au niveau du bus sur lequel le clavier sera branché. L'attaquant peut alors récupérer le mot de passe maître et ainsi accéder à l'ensemble des mots de passe protégés par l'enclave.

Afin de prendre du recul par rapport à l'exemple sus-cité, présentons d'autres cas d'utilisation plus génériques pour lesquelles l'enclave a besoin d'établir une communication sécurisée avec un périphérique. Le premier cas d'utilisation, correspondant au gestionnaire de mots de passe, est celui d'une application **requérant une donnée secrète** de la part d'un utilisateur. Le cas symétrique est celui où **une donnée confidentielle doit être transmise** à l'utilisateur (on pourrait penser à un affichage sécurisé pour présenter des données bancaires à l'utilisateur). De manière similaire, l'enclave peut avoir besoin de transmettre une donnée confidentielle à un périphérique afin que celui-ci lui applique une transformation, soit parce que **l'enclave n'en est pas capable** (cas d'un *Hardware Security Module* par exemple), soit parce que **le périphérique est plus performant** pour opérer la transformation requise (dans le cas d'un accélérateur matériel). Une autre possibilité est une application travaillant sur des données émises par un périphérique nécessitant une preuve que **les données n'ont pas été forgées** par une autre entité que le périphérique attendu.

Bien que le domaine ciblé par les architectures Intel Skylake discrédite *a priori* le dernier cas d'utilisation, les

autres scénarios sont loin d'être inimaginables dans le cas de serveurs dont la maintenance est confiée à un prestataire externe. L'utilisation de machines performantes pour appliquer un algorithme de transformation à un ensemble conséquent de données est en particulier une réalité assez commune dans les architectures d'informatique en nuage modernes, à tel point que Microsoft intègre depuis plusieurs années des FPGA à ses serveurs Azure.

2.2 Nouvelles propriétés de sécurité souhaitées

Dans les différents scénarios proposés, les propriétés souhaitées pour le canal de communication entre l'enclave et le périphérique ne sont pas toutes les mêmes. Mais avant de définir ces propriétés, intéressons-nous aux attaquants contre lesquels ces propriétés devront être garanties. Lorsque nous discuterons des différentes solutions possibles dans la prochaine section, nous envisagerons les capacités des attaquants selon deux axes orthogonaux : premièrement, nous discuterons de **l'efficacité d'une solution en fonction de sa capacité à protéger contre un attaquant logiciel** (une autre application ou le système d'exploitation). Deuxièmement, nous nous intéresserons à **sa résistance contre un attaquant capable de manipuler le bus périphérique** (en se concentrant sur les bus PCIe) ou le bus mémoire. Comme pour Intel SGX, nous ignorerons les attaques par canaux auxiliaires.

Face à ces attaquants, nous chercherons quelles propriétés sont garanties par les différentes solutions proposées. Ces propriétés sont les suivantes : **l'authentification** des deux partis (le périphérique auprès de l'enclave et inversement), **la confidentialité** des données échangées, **l'intégrité** de ces données et la possibilité de **détecter des rejeux**.

3. Quelques pistes de solution

Différentes solutions à ce problème de communication sécurisée entre enclave et périphérique peuvent être proposées. Nous les présentons ici et discutons de leur intérêt avant de détailler l'une d'entre elles qui nous paraît la plus intéressante.

3.1 Sécuriser un accès à un périphérique sans Intel SGX

Le problème qui nous intéresse ne date pas de l'introduction de la technologie Intel SGX. D'autres projets préexistants ont proposé des solutions afin de protéger un canal de communication entre une application et un périphérique dans un contexte où la pile logicielle privilégiée était (en partie) corrompue. Parmi ces solutions, une part importante s'appuie sur une application (un pilote périphérique) supposé non corrompu afin de chiffrer les informations reçues du périphérique et de les transmettre à l'application (ou l'opposé). Pour garantir la sécurité du pilote, ces solutions nécessitent l'utilisation d'une technologie d'isolation souvent implémentée en logiciel (comme un hyperviseur) dans laquelle il est nécessaire de faire également confiance.

Si l'on compare le modèle d'attaquant supposé par ce type de protection, on s'aperçoit que celui-ci est nettement plus strict que celui d'Intel SGX : l'attaquant n'est pas en mesure de manipuler le bus périphérique et il est nécessaire de faire confiance à plusieurs éléments logiciels. De plus, porter directement ce type de solution sur une architecture SGX ne permet pas de se protéger contre un attaquant plus puissant. En effet, l'architecture Intel ne garantit en soit aucune protection contre les attaques matérielles ciblant le bus périphérique. Finalement, il faut noter que même si l'on implémentait le pilote dans une enclave afin de bénéficier de l'isolation garantie par Intel SGX, cette isolation ne concernerait que la mémoire du pilote. Un hyperviseur ou système d'exploitation serait toujours en mesure d'intercepter les échanges entre le pilote et le périphérique.

3.2 Chiffrement logiciel

Si l'on souhaite conserver le modèle d'attaquant matériel utilisé par Intel SGX, cela signifie nécessairement que les données transitant sur le bus périphérique devront a minima être chiffrées afin d'en garantir la confidentialité. Si l'on ignore la question du chiffrement du côté du périphérique, il reste à décider de l'entité qui implémentera ces fonctions de cryptographie sur la machine Intel.

La réponse immédiate la plus évidente est de laisser au pilote logiciel la responsabilité du chiffrement de la communication. Ce pilote, implémenté dans une enclave permettrait d'établir une communication sécurisée avec le périphérique. Le modèle d'attaquant alors envisagé serait identique à celui d'Intel SGX (en ignorant l'implémentation du périphérique même) et l'ensemble des propriétés évoquées précédemment (authentification, confidentialité, intégrité) pourrait être garanti par différents mécanismes cryptographiques.

Le désavantage de cette solution est qu'elle représente un coût important en performance. En effet, et si l'on ignore le coût dû au chiffrement logiciel (l'architecture Intel propose un jeu d'instruction AES implémenté en matériel), Intel SGX ne permet pas aux périphériques d'accéder directement (par *Direct Memory Access* ou *DMA*) à la mémoire d'une enclave. Transmettre les données chiffrées au périphérique nécessiterait donc soit que l'enclave les envoie directement au périphérique (en utilisant répétitivement des instruction I/O), soit qu'elle les écrive dans la mémoire non protégée où le périphérique pourrait y accéder directement. Cette dernière méthode aurait également un coût important en performance car elle générerait du trafic sur le bus mémoire.

3.3 Unifier chiffrement mémoire et périphérique

Afin de conserver un modèle d'attaquant permissif, tout en améliorant les performances vis-à-vis de la solution précédente, il ne semble y avoir d'autres choix que d'apporter des changements à l'architecture SGX. Nous entrons donc à présent dans des considérations théoriques qui nécessiteraient un prototypage afin de les valider en vue d'une potentielle intégration à l'architecture d'Intel SGX.

L'idée d'une implémentation matérielle d'un canal sécurisé entre une enclave et un périphérique s'inspire du modèle d'Intel SGX où le chiffrement des accès mémoire est transparent pour les applications. La première idée pour étendre ce modèle aux périphériques est de profiter du chiffrement effectué par le contrôleur mémoire : un périphérique cherchant à accéder à la mémoire d'une enclave pourrait aller directement *piocher* dans les données chiffrées stockées dans la mémoire externe. Il s'agirait pour cela d'autoriser les transferts DMA ciblant la mémoire des enclaves et de modifier le contrôleur mémoire afin de désactiver le déchiffrement pour les accès mémoire initiés par le périphérique.

Cette solution est cependant compliquée par des détails d'implémentation. Premièrement, cette solution nécessiterait que le contrôleur mémoire et le périphérique partagent la clé de chiffrement utilisée pour chiffrer les données en mémoire. Afin que le périphérique ne soit pas en mesure de déchiffrer la mémoire d'autres enclaves, il faudrait que le contrôleur mémoire gère plusieurs clés. Cette gestion demanderait d'ajouter beaucoup de logique au contrôleur mémoire qui, pour le moment, n'utilise qu'une clé et ne connaît le concept ni de page, ni d'enclave. La deuxième difficulté est que les données d'une enclave ne sont chiffrées que lorsqu'elles sont stockées dans la mémoire externe. Elles apparaissent en clair dans les caches. Or, les architectures Intel récentes autorisent les accès DMA à accéder aux données directement depuis les caches pour des questions de performance. Il faudrait donc modifier ce comportement pour les enclaves et accepter la baisse de performance associée, ou alors ajouter des mécanismes de contrôle d'accès aux caches qui seraient effectués en fonction des enclaves associées aux pages en cache. Finalement, afin d'empêcher les attaques par rejeu, il serait nécessaire d'ajouter un numéro de séquence ou mécanisme similaire (c'est-à-dire un champ variant d'une transaction à l'autre) aux transferts ciblant le périphérique. Ce numéro de séquence est nécessairement différent de celui utilisé pour les accès mémoire par Intel SGX puisque chaque transaction périphérique ne correspond pas à une transaction du processeur (et réciproquement). L'ajout de ce numéro de séquence demanderait d'implémenter une nouvelle couche de chiffrement qui aurait un impact en performance.

3.4 Chiffrement matériel

Finalement, la solution que nous développons dans la dernière section permet de ne pas affaiblir le modèle d'attaquant d'Intel SGX, d'améliorer les performances comparativement à un scénario de chiffrement logiciel et limite les modifications qu'il est nécessaire d'apporter à l'architecture SGX.

4. Notre proposition : utiliser une nouvelle approche logicielle/matérielle

Notre proposition pour une nouvelle architecture est représentée à la Figure 2. Les blocs en gris représentent les modifications matérielles à réaliser sur l'architecture Intel SGX au niveau du CPU et au niveau du moteur DMA. Nous détaillons par les suite les différentes modifications.

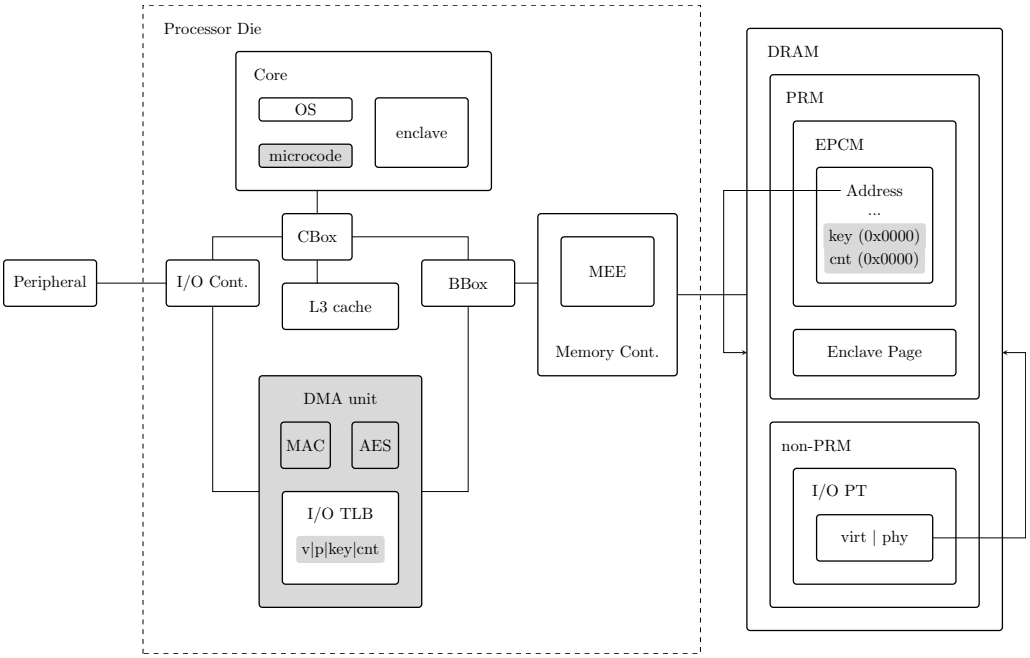

Fig. 2 : Notre proposition pour étendre Intel SGX

4.1 Modification au niveau du CPU

4.1.1. Ajout d'une clé et d'un compteur au niveau de l'EPCM

Nous réutilisons les structures déjà existantes de SGX (EPCM) afin d'ajouter une clé de chiffrement symétrique *key* pour communiquer avec le périphérique et un compteur *cnt* afin d'identifier de façon unique les messages échangés avec le périphérique. La génération et l'installation de la clé symétrique sont expliquées dans le paragraphe sur la gestion des clés.

4.1.2. Ajout d'une instruction

Cette nouvelle instruction (microcode du processeur) doit permettre de manipuler la clé *key* et le compteur *cnt*. Lorsque l'instruction est invoquée, le processeur vérifie que la page ciblée appartient à l'enclave exécutant cette instruction. Toutefois, autoriser le fait de positionner une clé laisse la porte ouverte à la réutilisation de la même clé pour des pages différentes, et donc ouvre la porte à des attaques par rejeu. Une solution serait que l'instruction ne permette pas de positionner une clé, mais de générer une nouvelle clé (unique). Cela aurait pour conséquence d'obliger à partager plusieurs clés avec un périphérique si l'échange de données entre périphérique et processeur utilise plusieurs pages mémoire.

4.2 Gestion des clés

La gestion des clés a pour objectif de partager une clé secrète entre un périphérique et une enclave. Le protocole est présentée à la Figure 3 où l'enclave est intitulée *Driver*. Ce protocole comporte de plus une entité externe (appelée *Verifier*) qui vérifie que le périphérique (*Peripheral*) et l'enclave sont authentiques. Le processus d'attestation SGX est modélisé sous la forme d'une seule entité appelée *QuotingEnclave* dont l'objectif est de fournir une signature de l'état de l'enclave.

Plus précisément, l'échange se fait ainsi :

1. L'entité externe de vérification *Verifier* envoie un nonce *n1* au *Driver*.
2. Le *Driver* envoie un nonce *n2* au *Peripheral*.
3. Le *Peripheral* répond alors au *Driver* sous la forme du nonce *n2*, d'un nonce *n3,* d'un identifiant *id,* et d'une signature de ces éléments réalisée avec la clé privée du *Peripheral*.
4. Le *Driver* crée un nouveau couple (clé privée, clé publique). Il envoie alors au *processus d'attestation* sa clé publique, *n1*, *id*, *n3*, et une signature de *id* et *n3* basée sur sa clé privée.

5. Ce message est expédié au *Verifier*.
6. Cette unité vérifie alors que ce message provient d'une enclave dûment autorisée et qui contient *n1*. Le *Verifier* récupère la clé publique du *Peripheral* en fonction de son *id*. Le *Verifier* envoie un message signé au *Driver* avec d'une part la clé publique du *Peripheral* et d'autre part *n3* et la clé publique du *Driver* signés avec la clé publique du *Verifier*.
7. Le *Driver* vérifie alors les signatures du dernier message reçu, ainsi que la signature du message numéro 3. En cas de succès, le *Driver* crée aléatoirement une clé symétrique secrète *sk*. Le *Driver* envoie un message comprenant : la signature réalisée par le *Verifier* de *n3* et de la clé publique du *Driver*, la clé publique du *Driver* et enfin la clé sk chiffrée avec la clé publique du P*eripheral*.
8. Le P*eripheral* vérifie d'une part la signature du *Driver* mais aussi celle réalisée par le *Verifier*, puis déchiffre *sk*. Les deux entités *Peripheral* et *Driver* possédant ainsi la même clé *sk*, ils peuvent échanger des secrets.

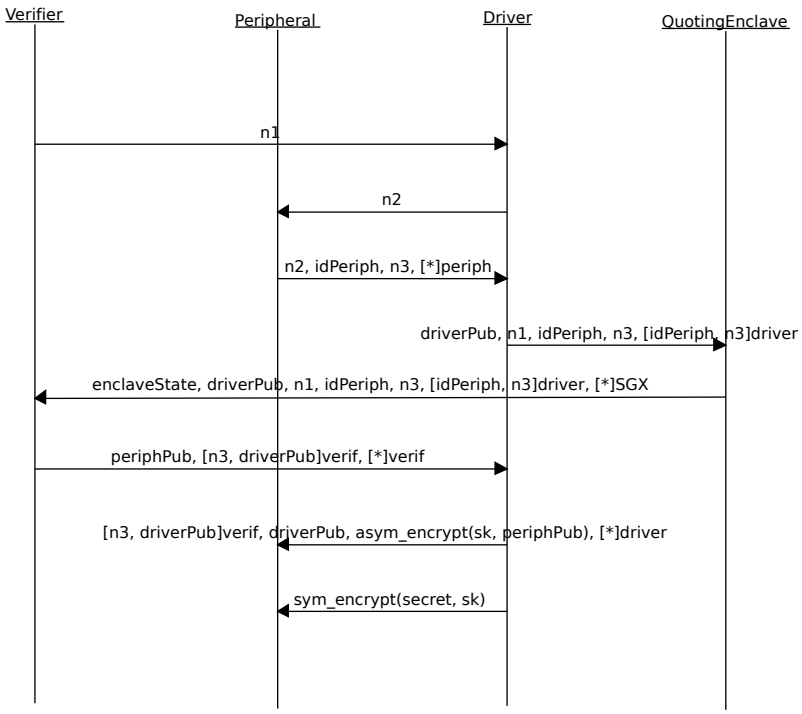

Fig. 3 : Partage d'une clé symétrique entre un pilote de périphérique (*Driver)* et un périphérique *(Peripheral)*

Afin de comprendre les intuitions derrière ce protocole relativement complexe, il faut partir des objectifs du *Driver* et du *Peripheral*. Il s'agit pour chacun d'eux d'accepter le partage d'une clé (*sk*) avec une autre entité approuvée par le *Verifier*, tout en ayant la certitude que les messages reçus des autres participants au protocole ont bien été générés en réponse à un message de l'instance courante du protocole. Ainsi, on note que chaque participant (*Verifier*, *Driver* et *Peripheral*) génère un nonce (respectivement *n1*, *n2* et *n3*) et n'accepte une signature que si elle contient le nonce généré (aux étapes 6, 7 et 8 respectivement). La signature vérifiée à l'étape 6 inclut le nonce *n1* généré à l'étape 1, etc.

Si on laisse de côté les nonce, le protocole est simplifié : à l'étape 3, le *Peripheral* fournit son id. A l'étape 4, le *Driver* associe un couple de clés à la requête. A l'étape 6 le *Verifier* vérifie que la requête émane bien d'une enclave autorisée et qu'elle cible un *Peripheral* également autorisé. Le *Verifier* crée également deux messages : le premier est destiné au *Driver* et atteste de la légitimité de la clé publique du *Peripheral* en la signant avec la clé du *Verifier* ; le second est son pendant adressé au *Peripheral* : cette fois-ci, c'est la clé publique du *Driver* qui est signée. Le premier message sera vérifié à l'étape 7 et le second à l'étape 8. Une fois que le *Peripheral* est convaincu de la validité de la clé publique du *Driver*, il peut accepter la clé symétrique *sk* générée à l'étape 7 et signée avec la clé asymétrique du *Driver*.

Le modèle abstrait le mécanisme d'isolation d'Intel SGX par une entité Driver dont les champs et comportements sont privés. Le mécanisme d'attestation est quand à lui modélisé par une autre entité QuotingEnclave qui atteste (signe) un certain état de l'enclave. Les Peripheral et Verifier sont modélisés également par des entités distinctes, de sorte qu'un attaquant peut à la fois espionner et manipuler la communication entre le Driver et le Peripheral et entre le Driver et le Verifier. Ce modèle est disponible dans le catalogue de modèles de l'outil TTool.

Notons que notre protocole permet de n'avoir aucun matériel cryptographique pré-requis dans le *Driver*. Aussi, le *Driver* ne sait pas avec quel périphérique il correspond puisque l'authentification se fait, comme sous SGX, avec une autorité externe de confiance (*Verifier*). De façon similaire, le périphérique ne sait pas avec quel *Driver* il échange des données, et en particulier si le *Driver* est réellement exécuté sur un système SGX : il se repose donc lui aussi sur son tiers de confiance. Enfin, notre approche permet au *Verifier* d'autoriser ou non des communications entre des *Drivers* et des *périphériques*, permettant ainsi de refuser des échanges non sécurisés.

Ce protocole de gestion de clés a été modélisé avec TTool [5] (notre modèle SGX est téléchargeable depuis l'outil) et vérifié formellement via le traducteur automatique de modèles TTool vers ProVerif. Sous le modèle d'attaquant expliqué auparavant, l'outil de vérification est en mesure de prouver la confidentialité d'un message chiffré avec la clé symétrique établie sk. Il faut noter que l'authenticité n'est pas vérifiée pour ce modèle. En effet, le *Driver* ne sait pas a priori avec quel *Peripheral* il va communiquer (et vice versa). Il ne peut donc mécaniquement pas l'authentifier. Pour obtenir cette propriété d'authenticité, il faudrait améliorer notre modèle afin que le Verifier n'accepte qu'un seul Driver et / ou un seul *Peripheral.*

4.3 Modification au niveau du DMA

Dans notre proposition, le DMA est en charge de réaliser les chiffrements et déchiffrements des transactions mémoire avec les enclaves. Deux modules cryptographiques doivent être ajoutés au moteur DMA : un module de chiffrement et déchiffrement, et un module de protection de l'intégrité (type *Message Authentication Code* ou *MAC*), à l'image de ceux présents dans le MEE. Aussi, les lignes de la TLB (Translation Look-aside Buffer) pour les entrées / sorties doivent être étendues pour contenir une clé et un compteur.

Nous allons à présent expliquer de façon détaillée comment un échange de données sécurisé peut être réalisé via un transfert DMA.

La première partie de la Figure 4 concerne la configuration au niveau du système préalablement à l'accès sécurisé à une page de l'enclave. L'enclave utilise la nouvelle instruction pour définir la clé correspondant à la page donnée. Cette instruction vérifie les droits de création, puis positionne la clé dans l'entrée de l'EPCM correspondant à la page, et réinitialise le compteur de cette page à 0. Ensuite, l'enclave demande une mise à jour des pages des I/Os au système d'exploitation.

La seconde partie du transfert décrit à la Figure 4 concerne la lecture d'une donnée par un périphérique. Celle-ci commence avec le message « read » envoyé par le périphérique.

1. Le périphérique fait la demande de lecture au moteur DMA
2. S'il ne la possède pas déjà, le moteur DMA résout l'adresse virtuelle requise par le périphérique en une adresse physique (avec l'aide du système d'exploitation) puis récupère l'entrée EPCM de la page correspondante. Le moteur DMA ajoute la correspondance entre adresse virtuelle et physique, la clé et le compteur dans sa TLB des entrées / sorties (cache des accès) : « saveInIOTLB »
3. Le moteur DMA réalise une lecture à l'adresse physique correspondante en s'adressant au contrôleur mémoire. La réponse est stockée par le DMA dans le cache L3 dont une partie est réservée dans les processeurs Intel pour les transferts DMA.
4. Le moteur DMA utilise la clé symétrique pour chiffrer le compteur et la donnée, et pour calculer le MAC du compteur et de l'adresse virtuelle. Le message contenant le chiffré et le MAC est envoyé au périphérique.

Dans le cas d'une écriture, le périphérique émet un message qui comporte un chiffré de la donnée et du compteur, et un MAC du compteur et de l'adresse virtuelle. Le DMA déchiffre la première partie, vérifie que le compteur est correct par rapport à la valeur présente en mémoire, puis vérifie le MAC. Le compteur est alors incrémenté. La donnée est envoyée à la Cbox qui écrit la donnée soit en mémoire cache, soit en mémoire physique, en fonction de la politique de gestion des caches.

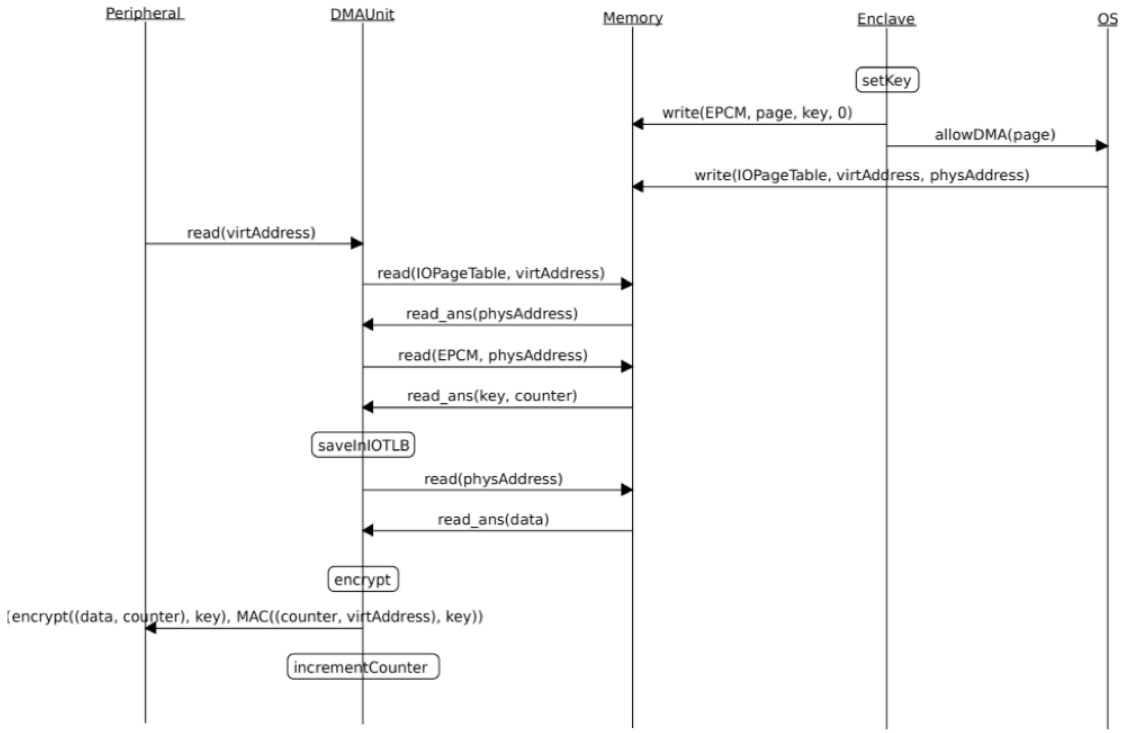

Fig. 4 : étapes d'un transfert DMA sécurisé

4.4 Impact sur les performances

Premièrement, l'avantage de notre approche est qu'elle est totalement transparente pour tout ce qui se trouve après le contrôleur d'entrées/sorties.

Lorsqu'un périphérique désire accéder pour la première fois à une page protégée, l'adresse virtuelle utilisée par le périphérique doit d'abord être traduite en une adresse physique par le moteur de gestion des I/Os du DMA. Ce processus est le même pour des périphérique non protégés. Par contre, une fois que l'adresse a été traduite, les accès à la clé et au compteur génèrent deux accès mémoires avec utilisation du MEE. De plus, les transferts DMAs ont besoin d'être déchiffrés par le MEE (si la donnée n'est pas en cache), puis chiffrés à nouveau à l'aide du moteur DMA, et enfin déchiffrés par le périphérique. Finalement, en supposant que les données à transférer ont une grande probabilité d'être en mémoire cache et que clé et compteur sont présents dans la TLB des entrées / sorties, la latence totale a été estimée à environ deux fois l'exécution d'un chiffrement à l'aide d'un MEE [3]. Les auteurs de [4] estiment que la pénalité moyenne engendrée par un MEE est de 5,5 %, soit finalement une pénalité de 11 %. Cet impact est à mettre en balance avec les garanties fortes de sécurité ainsi obtenues.

5. Conclusion

Si l'environnement d'exécution Intel SGX permet une isolation forte au sein d'un processeur Intel SGX, permettant en particulier de se protéger contre le système d'exploitation, cette isolation ne prend pas en compte des communications avec des périphériques, ce qui est pourtant important dans le cas de l'utilisation d'une plateforme qui n'est pas de confiance. Notre proposition repose sur une mise à jour matérielle plutôt légère et qui impacte peu les performances, tout en offrant des garanties de sécurité importantes. Les principales modifications concernent la création d'une clé symétrique, l'utilisation d'un compteur, et l'amélioration des moteurs DMAs avec des fonctions cryptographiques.

Références

[1] Intel SGX Explained (V. Costant et S. Devadas) : **https://eprint.iacr.org/2016/086.pdf**

[2] SgxPectre Attacks: Stealing Intel Secrets from SGX Enclaves via Speculative Execution (G. Chen, S.

Chen, Y. Xiao, Y. Zhang, Z. Lin, T. H. Lai) :**https://arxiv.org/abs/1802.09085**

[3] Environnement pour l'analyse de sécurité d'objets communicants (F. Lugou) : **http://www.theses.fr/2018AZUR4005**

[4] A Memory Encryption Engine Suitable for General Purpose Processors (S. Gueron) : **https://eprint.iacr.org/2016/204**

[5] TTool **:** **https://ttool.telecom-paristech.fr/**