Trusted Computing : Limitations actuelles et perspectives

Frédéric Guihéry, Goulven Guiheux, Frédéric Rémi
prenom.nom(@)amossys.fr

Laboratoire d'évaluation d'AMOSSYS
4 bis allée du Bâtiment
35000 Rennes
http://www.amossys.fr

Résumé Cet article à vocation prospective dresse un panorama des atouts et limites de l'informatique de confiance pour la réalisation d'une architecture de sécurité sur une plateforme PC. L'étude met en évidence deux problèmes de positionnement majeurs : une adéquation imprécise vis-à-vis des besoins des différents marchés ainsi que le manque de clarté sur le positionnement du TPM relativement aux technologies concurrentes. L'article montre que la couverture de nouveaux besoins liés à l'intégrité et la confidentialité d'exécution nécessiterait soit une adaptation du TPM (transformation vers un crypto-processeur, voire un processeur sécurisé), soit l'utilisation de technologies complémentaires telles que la virtualisation.

Mots-clés: Trusted Computing, TCG, TPM, DRTM, TXT, architectures sécurisées.

1 Introduction

Cet article étudie l'apport et les limites de l'informatique de confiance pour la réalisation d'une architecture de sécurité à partir d'une plateforme de type PC. Les mécanismes de sécurité sous-jacents sont considérés tant sur les aspects logiciels que sur les aspects matériels.

L'informatique de confiance, notamment au travers de la puce TPM, est présente dans l'industrie depuis environ 7 ans. Pour autant, les concepts de sécurité associés – tels que le principe de chaine de confiance, l'attestation d'intégrité ou encore le chiffrement conditionnel –, majoritairement poussés par le Trusted Computing Group (TCG), sont aujourd'hui peu implémentés par les industriels. Le composant TPM et sa pile logicielle demeurent les seules réalisations largement déployées sur les parcs informatiques grands publics

et professionnels. Mais très peu d'applications tirent réellement parti des services offerts par le TPM. En effet, seul le chiffrement conditionnel est mis en œuvre de façon industrielle (notamment via BitLocker de Microsoft), les autres services restant cantonnés au stade de prototype.

Ce constat peut s'expliquer par différents facteurs, déclinés cidessous par ordre d'importance :

- problème d'adéquation aux besoins des différents marchés ;
- soucis de positionnement par rapport aux approches concurrentes ;
- faible maturité de la technologie.

Une remise en cause est donc nécessaire et a justifié l'écriture de cet article, dont les objectifs sont multiples : en premier lieu, il s'agit d'offrir un panorama détaillé des limitations actuelles de l'informatique de confiance. En parallèle, nous présentons une analyse de besoins jusqu'à présent peu pris en compte par le TCG afin d'identifier de nouvelles orientations pour les mécanismes de sécurité. Les besoins considérés dans cet article sont à la fois relatifs au démarrage d'une plateforme dans un état de confiance (cas d'usage originel du TCG) et au maintien de cet état pendant le fonctionnement du système. Nous abordons également l'exécution confidentielle de code.

Cette analyse est complétée par une comparaison avec les technologies potentiellement concurrentes (crypto-processeurs, solutions logicielles...). A partir de ces informations, nous proposons de nouvelles perspectives pour l'informatique de confiance, à la fois sur le plan technique que sur le plan stratégique.

2 Objectifs de sécurité

Dans ce chapitre, nous proposons une présentation d'objectifs de sécurité propres à la protection de l'exécution de code sur une plateforme de type PC. Nous verrons par la suite de quelle manière ils sont pris en compte par les solutions et travaux de recherche actuels.

2.1 Assurer l'intégrité du système au démarrage

Tous les éléments de la chaine de démarrage sont concernés : BIOS, MBR, Boot Loader, hyperviseur, noyau, initrd, modules, fichiers de configuration et applications.

La menace concerne avant tout le piégeage des éléments du système de fichiers (noyau, modules, binaires, etc.), lors d'un accès illégitime à la plateforme (notamment au disque dur).

Plusieurs sous-objectifs, ordonnés par niveau de robustesse :
- intégrité de l'OS (noyau, modules, système de fichiers) ;
- intégrité de l'ensemble de la chaine de démarrage (BIOS, microcodes de périphériques, boot loader et OS) ;

2.2 Assurer l'intégrité du système pendant l'exécution

L'assurance d'intégrité porte sur tous les éléments logiciels du système (noyau et des modules, exécutables et bibliothèques, fichiers de données, etc.).

La menace est double : la perte d'intégrité à cause de maliciels (vers, virus, rootkits, portes dérobées, bombes logiques, etc.) ou à cause d'accès illégitimes.

Plusieurs sous-objectifs, ordonnés par niveau de robustesse :
- intégrité des applications au démarrage ;
- intégrité des applications pendant l'exécution ;
- intégrité du noyau pendant l'exécution.

2.3 Assurer la confidentialité d'exécution

Plusieurs besoins :
- la protection de la propriété intellectuelle (textes, images, vidéos, jeux-vidéo, implémentations d'algorithmes) des ayants droit et des fournisseurs de contenu.
- l'exécution cloisonnée de processus (notamment sur des serveurs mutualisés) ;
- l'exécution cloisonnée de processus sensibles (typiquement, des implémentations d'algorithmes cryptographiques) ;
- la protection contre l'espionnage des processus ou données des autres utilisateurs ;

– la protection contre la rétro-analyse des mécanismes protégeant la propriété intellectuelle d'ayants droit.

Plusieurs sous-objectifs, ordonnés par niveau de robustesse :

– confidentialité d'exécution vis-à-vis d'un autre utilisateur du système ;
– confidentialité d'exécution vis-à-vis de l'administrateur du système ;
– confidentialité d'exécution vis-à-vis de l'OS (noyau et modules) ;
– confidentialité d'exécution vis-à-vis d'une attaque matérielle (espionnage des communications sur les bus de données).

3 Présentation du Trusted Computing et de la puce TPM

Le TCG (Trusted Computing Group) est un consortium industriel dont la vocation principale est de définir des standards ouverts de sécurité pour les plateformes de type PC.

La clé de voute de l'architecture de sécurité est consituée par le composant cryptographique TPM (Trusted Platform Module) qui permet notamment d'offrir aux utilisateurs un composant physique de confiance sur le poste client. Ce composant met à disposition de l'utilisateur des fonctions cryptographiques de base (génération d'aléa, de clés, de signatures électroniques, de hachage), de stockage sécurisé dépendant de l'état de la plateforme et d'agrégation de mesures. La confiance affichée provient alors de la nature matérielle du composant. Le pilotage du composant TPM est quant à lui réalisé via une architecture logicielle dédiée, la TSS pour TCG Software Stack.

L'objectif du TCG est de fournir un service attestant de l'intégrité d'une plateforme reposant sur cette architecture matérielle et logicielle. Cette architecture permet de mesurer les informations jugées sensibles (fichiers exécutables, librairies, pilotes, noyau du système, clés d'activation, fichiers de configurations, données personnelles, ...). Ce service d'attestation est soit local, soit distant. Dans le second cas, les mesures effectuées sont communiquées via un protocole spécifique à un tiers de confiance qui valide ou non ces mesures.

En complément de ses travaux sur le TPM, le TCG promeut des architectures de sécurité fournissant des cas d'usage concrets :

- *Trusted Network Connect (TNC)*. L'objectif de cette spécification est d'améliorer la sécurisation d'une infrastructure réseau en conditionnant l'accès au réseau d'un poste client à la vérification de son intégrité.
- *Full Disk Encryption (FDE)*. L'objectif de cette spécification est de founir un chiffrement de disque autonome et non lié à une plateforme.

3.1 Le composant TPM et les services offerts

Le TPM (*Trusted Platform Module*) est un composant passif assimilable à une carte à puce dans le sens où il interagit avec sa pile utilisatrice selon un modèle challenge-réponse.

Le composant est de préférence soudé à la carte mère et communique avec celle-ci par le biais du bus LPC, géré par le *south bridge*. Le TPM est composé de plusieurs modules (voir figure 1) décrits fonctionnellement dans les paragraphes suivants.

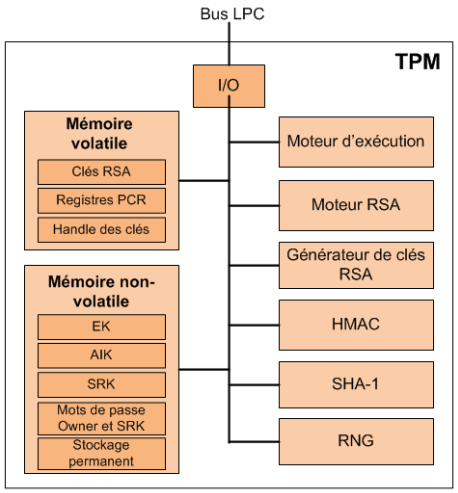

Figure 1. Composition d'une puce TPM

- **Mémoire non-volatile**. La mémoire non-volatile permet d'assurer le stockage permanent des données suivantes en mémoire :

  - **Endorsement Key (EK)**. Paire de clés asymétriques RSA générée par le fabriquant de la puce TPM pour son identification en phase de personnalisation du composant. Cette génération est normallement associée à une phase de certification de la partie publique de la clé EK. Ce certificat devient la carte d'identité du composant TPM ; il fournit la preuve que le TPM est authentique, c'est-à-dire que ce n'est pas une contrefaçon ou une version logicielle du composant matériel ;
  - **Storage Root Key (SRK)**. Paire de clés asymétriques RSA. Elle joue le rôle de clé racine dans un système de hiérarchie des clés classique. Seule sa partie privée est stockée dans une partie protégée de la mémoire non-volatile. La partie publique de la clé est utilisée pour le chiffrement de données ou des clés filles.
  - **Owner Authorization Data**. Un condensé de 160 bits créé et stocké en mémoire. Cette valeur constitue le mot de passe commun (également appelé secret d'autorisation) entre le composant et le propriétaire de la plateforme. Il permet d'assurer au propriétaire l'usage exclusif des capacités de sécurité du composant TPM.
- **Mémoire volatile**. Cette mémoire permet d'assurer le stockage des données temporaires.
- **Module Plateform Configuration Register**. Les mémoires PCR sont des registres de 160 bits permettant de stocker l'état d'une plateforme. Ils sont utilisés pour agréger les mesures d'intégrité entre elles. Le TCG spécifie un minimum de 16 registres dont les 8 premiers (0-7) sont réservés à un usage interne du TPM. Les autres registres, soit 8 au minimum (8-15), sont réservés aux applications (i.e. chargeur d'amorce, noyau, bibliothèques système, drivers, applications).
- **Module Attestation Identity Keys (AIKs)**. Ce module optionnel permet le stockage des clés AIK. Les clés AIK sont des bi-clés utilisées lors du protocole d'attestation de la plateforme.
- **Générateur d'aléa (RNG)**. Ce module implémente un générateur d'aléa physique. Cet aléa est utilisé pour la génération des

clés, des vecteurs d'initialisation et des challenges aléatoires pour les protocoles cryptographiques.

- **Module SHA-1**. Ce module implémente la fonction de hachage cryptographique SHA-1. Le TCG étant un centre de normalisation, ce module devrait évoluer dès la standardisation par le NIST d'un nouvel algorithme de hachage cryptographique en remplacement de SHA-1.
- **Module HMAC**. Ce module implémente la fonction de hachage à clé HMAC. La taille des clés est de 20 octets et la taille des blocs est de 64 octets. La fonction supporte le calcul du HMAC selon la RFC 2104.
- **Génération de clés RSA**. Le module de génération de clés permet de créer des clés symétriques et asymétriques. Le module de génération supporte des clés RSA jusqu'à 2048 bits (512 à 2048 bits).
- **Module RSA**. L'algorithme RSA est utilisé à la fois en mode signature et en mode chiffrement. Le TCG respecte le standard PKCS #1 qui fournit les détails des spécifications pour la signature, le chiffrement et le formatage des données RSA.

Les services offerts nativement par le composant sont les suivants :

- **Unicité de la plateforme**. Chaque TPM est associé à une clé RSA unique appelée Endorsement Key (EK). Cette clé est générée par le constructeur en phase de personnalisation du TPM. La partie publique de cette clé est certifiée par une AC associant à chaque TPM un certificat unique. Cette clé ne doit jamais être communiquée à l'extérieur du TPM ;
- **Stockage sécurisé**. Le TPM intègre un service de protection des données en confidentialité. Ce service repose sur une hiérarchie de clés. Chaque clé est protégée par une clé de niveau supérieur. La racine de cette hiérarchie est constituée par la clé SRK (Storage Root Key). Cette clé est stockée à l'intérieur du TPM, tandis que les autres clés sont stockées sur un support externe (disque dur par exemple). La clé SRK est générée par le TPM lors de la prise de possession du TPM par le propriétaire de la machine (commande TPM TakeOwnership).
- **Scellement de données**. Ce service est un mécanisme de stockage sécurisé dépendant de l'état de la plateforme. Le de-

scellement d'une donnée n'est possible que si l'état des registres PCR au moment du scellement est identique à l'état des registres PCR au moment du descellement. Ce mécanisme est utilisé pour le service d'attestation locale.

- **Opérations cryptographiques**. Le TPM met à disposition de l'utilisateur les fonctions cryptographiques suivantes :
  - Fonction de hachage SHA-1 ;
  - Génération de clés, chiffrement et signature RSA en 2048 bits conformément aux travaux du groupe de standardisation de l'IEEE P1363 ;
  - Génération d'aléa.
- **Extension des registres PCR**. Ce mécanisme est utilisé pour le service d'attestation locale et distante du poste client. Il permet d'agréger des mesures réalisées par des logiciels tiers dans les registres PCR du TPM. Le processus d'extension est une opération cryptographique simple consistant à concaténer la valeur courante du registre PCR avec la nouvelle mesure puis à hacher le tout :

$$PCR_{t+1}[i] = SHA1(PCR_t[i]||M) (1)$$

M désigne la mesure réalisée par une application logicielle. Selon le concept de mesure défini par le TCG, M = SHA-1(DATA). La variable i désigne l'index du registre concerné par l'extension. Cette opération offre plusieurs avantages :
  - Il est calculatoirement impossible de trouver des collisions sur la valeur des registres pour deux mesures différentes ;
  - La mesure n'est pas commutative ; i.e. deux mesures déséquencées produisent des résultats différents ;
  - L'opération permet de stocker un nombre illimité de mesures dans les registres PCR puisque le résultat est toujours une empreinte de 160 bits.

On notera par ailleurs que le composant TPM est, à l'instar d'une carte à puce, destiné à être certifié EAL4+ [10]. Nul doute que cette certification renforcera alors la confiance que l'on peut accorder aux services associés au TPM et notamment au principe d'attestation distante évoqué ci avant.

3.2 Chaines de confiance SRTM et DRTM

Une chaine de confiance est un service permettant le contrôle de l'intégrité du poste client. L'objectif de ce contrôle est de conditionner l'accès à certaines données, ressources ou applications. En pratique, l'intégrité de la plateforme est vérifiée par l'établissement d'une chaine de confiance liant tous les éléments critiques de la machine. Depuis le boot du poste client, chaque élément logiciel réalise une mesure d'intégrité de l'élément suivant avant de l'exécuter. En pratique, les éléments logiciels sont les suivants :

- le BIOS ;
- le gestionnaire de démarrage (par exemple, Grub) ;
- le système d'exploitation ;
- les applications.

La mesure d'une donnée ou d'une application est constituée par un motif d'intégrité SHA-1. Cette mesure est à l'initiative des applications. L'ensemble des mesures successives forment une chaine de confiance initiée par le CRTM (Core Root Trusted Measurement). Ce code est exécuté en premier sur le poste client au démarrage. L'objectif du CRTM est de réaliser une mesure de soi-même et de l'élément logiciel suivant. Concrètement, sur un poste client, le CRTM représente l'ensemble des premiers octets du BIOS qui vont s'auto-mesurer puis mesurer le reste du BIOS (voire figure 2). Le processus de mesure se déroule de la manière suivante : soient A et B deux entités telles que A exécute B :

- A mesure B (un exécutable ou un autre fichier). Le résultat est un condensé de B ;
- Ce condensé est écrit dans le fichier de log *Stored Measurement Log* (SML) stocké sur le disque ;
- A écrit le condensé de B dans un registre PCR via l'opération d'extension des registres PCR ;
- Le contrôle est donné à B.

La mesure d'intégrité fait intervenir deux éléments cruciaux :

- Le fichier SML. Il s'agit d'un fichier de journalisation des mesures géré par l'application initiatrice de la mesure. Ce fichier contient le triplet suivant :
  - Le registre PCR utilisé pour la mesure ;
  - Le condensé SHA-1 ;

  - Le nom de l'élément mesuré.
- Les registres PCR. Leur fonction est de garantir l'intégrité du fichier SML. Le processus de vérification de l'intégrité du fichier SML se déroule en deux étapes :
  - Rejeu logiciel du processus d'extension des registres sur la base des mesures du fichier SML ;
  - Comparaison du résultat avec l'état courant des registres PCR du TPM. L'intégrité est valide si les résultats sont identiques. Cette opération doit être initiée par une application tierce de confiance.

Afin de supprimer les éléments propres à la plateforme matérielle (BIOS, microcodes, etc.) dans une chaine de confiance, le processus de DRTM (*Dynamic Root of Trust Measurement*) a été spécifié. Son caractère dynamique vient du fait que le processus peut être initié lorsqu'un OS est en cours de fonctionnement. Le DRTM repose sur de nouvelles instructions processeur (*skinit* sous AMD SVM et GETSEC[SENTER] sous Intel TXT). Ces instructions réinitialisent l'état de la plateforme (passage en mono-processeur, désactivation des interruptions matérielles, mise à zéro des PCR 17 et 18 du TPM, etc.) et passent le contrôle à un *Secure Loader*, dont l'intégrité est automatiquement mesurée. Cette dernière étape est réalisée avec la fonction SHA-1 du TPM. Ensuite, ce *Secure Loader* peut démarrer un nouvel environnement d'exécution ou redonner la main à l'OS dont le fonctionnement a été mis en pause.

3.3 Technologie Intel TXT

La technologie Intel TXT (*Trusted eXecution Technology* est une implémentation du DRTM associée à de la protection mémoire IOMMU. Son objectif est de démarrer une plateforme dans un environnement d'exécution de confiance (c'est-à-dire dont le code est vérifié en intégrité) et protégé contre les accès externes (périphériques, autres domaines virtuels, etc.).

Les fonctions de sécurité offertes par Intel TXT font partie des extensions SMX (*Safer Mode Extensions*) d'Intel. SMX propose les services suivants :

- MLE : Measured Launched Environement – permet la mesure d'un environnement d'exécution lors d'un DRTM.

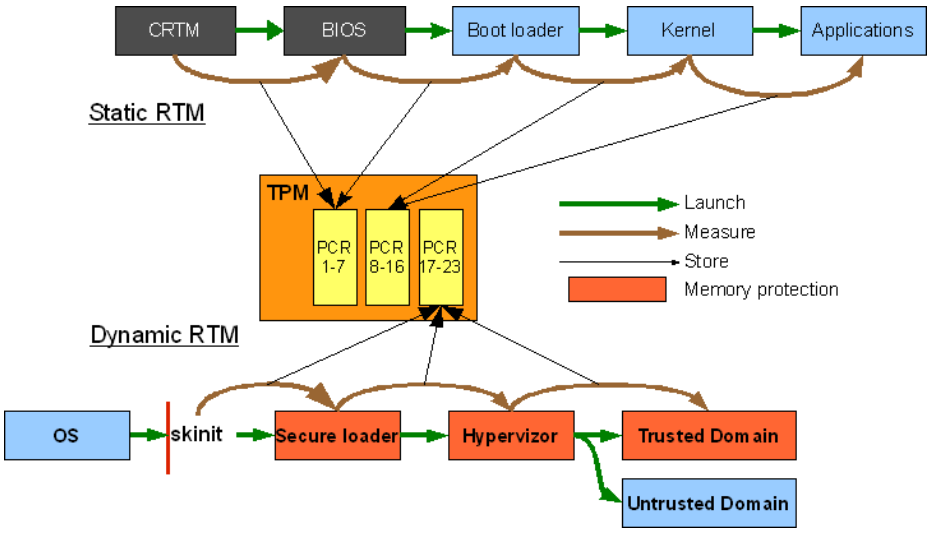

FIGURE 2. Chaines de confiance statique et dynamique

- LCP : Launch Control Policy – permet de définir une politique de sécurité indiquant le comportement attendu lors du processus de MLE (par exemple, continuer ou stopper l'exécution en cas d'intégrité non conforme).
- VERIFIED LAUNCH : permet l'application de la politique de sécurité définie (LCP) en fonction de l'environnement mesuré lors d'un MLE.
- PROTECTION IOMMU : mise en œuvre d'une MMU (*Memory Management Unit*) dédiée aux entrées/sorties afin de protéger l'accès DMA à certaines zones mémoires depuis des périphériques. Ces zones mémoires correspondent à des domaines au sens de la virtualisation.
- EFFACEMENT SÉCURISÉ : déclenchement de routines d'effacement sécurisé de la mémoire lors du basculement du processeur dans un mode spécifique (par exemple, une transition en mode hibernation S3) ou lors d'erreurs pendant le démarrage sécurisé. Ce service permet d'éviter de laisser des informations sensibles en mémoire.

Les étapes du processus de démarrage dans un environnement d'exécution de confiance, présentées sur la figure 3, sont les suivantes.

1. Mesure d'intégrité des éléments constitutifs du futur environnement d'exécution. Dans l'exemple, il s'agit d'un boot loader, d'un hyperviseur, d'un noyau Linux et de son *initrd*. A noter la présence du code SINIT AC, correspondant à un module fourni et signé par Intel, et dont la fonction est de vérifier l'état du système à l'initialisation d'un DRTM. Les mesures d'intégrité sont ensuite stockées dans une politique LCP.
2. Cette politique LCP est sauvegardée dans la mémoire non-volatile du TPM par le propriétaire de la plateforme (*owner*).
3. Lancement d'un DRTM avec l'instruction GETSEC[SENTER]). Le code SINIT AC, placé dans la mémoire interne du processeur, est alors exécuté. Il s'assure notamment qu'un seul processeur est en cours de fonctionnement et que les interruptions matérielles sont bien désactivées. Ces pré-requis sont nécessaire pour assurer la propriété d'atomicité d'exécution du démarrage sécurisé.
4. Le code SINIT AC calcule l'intégrité du prochain élément de la chaine de démarrage, le boot loader Trusted Boot.
5. Il compare ensuite la mesure d'intégrité avec celle de référence stockée dans le TPM, et récupère la politique à appliquer suivant le résulat de la comparaison.
6. En cas d'intégrité correcte, l'exécution est passée à Trusted Boot, qui réalise à son tour la mesure, la comparaison et l'application de la politique de sécurité sur les prochains éléments de la chaine de démarrage (opérations 7, 8 et 9 sur le schéma).

4 Problèmes de spécification et de conception

Du fait du caractère ouvert du processus de standardisation au sein du TCG, de nombreux acteurs du marché sont amenés à collaborer pour l'élaboration des différentes spécifications dans plusieurs groupes de travail. Cela implique une certaine inertie, avec le risque d'un manque de réactivité face à de nouveaux besoins, ou face à des attaques ou vulnérabilités identifiées en court de route. Le présent chapitre revient sur quelques erreurs de spécification et de conception propre à la technologie TPM. Certaines d'entres elles sont aujourd'hui corrigées.

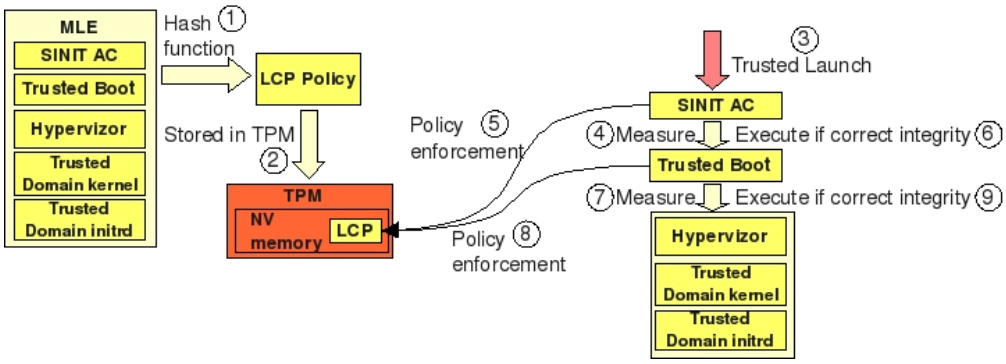

FIGURE 3. Mécanisme de vérification d'intégrité au démarrage avec Intel TXT

4.1 La cryptographie

Plusieurs problèmes relatifs à la prise en compte de la cryptographie par le composant TPM ont été identifiés. Tout d'abord, le TPM ne propose pas nativement à l'utilisateur d'algorithmie symétrique. En conséquence, pour chiffrer et déchiffrer efficacement de grandes quantités de données, il est nécessaire de faire appel à de la cryptographie symétrique réalisée de manière logicielle au sein du système d'exploitation. De fait, les attaques en mémoire visant à récupérer les clés de sessions, ou directement les données sensibles, sont toujours applicables. Les attaques de la littérature proposent différentes approches : analyse de la partition d'échange (*swap*) ou du fichier d'hibernation, analyse des *Core Dump*, attaque par démarrage à froid (*cold boot attack*) [12].

Par ailleurs, il n'est pas recommandé d'utiliser un unique algorithme pour mettre en œuvre plusieurs services cryptographiques – RSA pour la signature et le chiffrement, SHA-1 comme fonction de hachage, HMAC comme mécanisme d'authentification de messages.

Ce choix limité augmente le risque d'une perte confiance globale du TPM liée à la cryptanalyse d'un des mécanismes spécifiés pendant la durée de vie de la technologie TPM 1.2. Or, cette durée de vie est potentiellement importante pour un tel composant : de l'ordre d'une dizaine d'année. De plus, l'utilisation de SHA-1 n'est pas conforme au référentiel de l'ANSSI [1] pour une utilisation dans un produit visant une qualification standard. Ce qui est d'autant plus

pénalisant pour un produit visant une évaluation CC EAL4. Pour être en conformité avec le référentiel du certificateur, il faudrait au moins spécifier le mécanisme SHA-256. Le choix du RSA comme unique algorithme de chiffrement à clé publique et signature peut surprendre considérant qu'il est sujet à polémique, notamment sur la taille des modules ou encore l'introduction de portes dérobées dans la génération des clés [3]. Ces différents éléments montrent qu'il y a une certaine urgence à implémenter de nouveaux standards cryptographiques.

Certaines fonctionnalités liées aux mécanismes cryptographiques apparaissent en sus peu robustes. Par exemple, d'après les spécifications TPM, les clés privées RSA ne sont pas censées pouvoir sortir en clair du TPM. Or, du fait du mécanisme de migration, il est possible de récupérer une clé privée en réalisant la migration d'une clé RSA vers un faux TPM. Lors de la mise en œuvre d'un tel mécanisme dans une application, il faut donc faire attention aux types de clés créées – doivent-elles être migrables ou non ? –, et, dans le cas d'une migration de clé, s'assurer qu'elles soit migrées vers un TPM matériel authentique. La confiance dans la fonction de migration devrait donc reposer non seulement sur le mécanisme cryptographique intrinsèque mais également sur l'environnement.

Un risque au niveau du mécanisme de stockage sécurisé existe également dans certaines situations. Celui-ci repose sur une hiérarchie de clés, avec comme clé racine, la SRK (*Storage Root Key*). Cette dernière peut être configurée pour avoir un mot de passe publiquement connu (*well-known password*). C'est notamment pratique, voire nécessaire, lorsque plusieurs applications ou utilisateurs tirent parti du mécanisme de stockage sécurisé sur une même plateforme. Cependant, lors de la création d'une clé, la donnée d'authentification qui va être associée est transmise chiffrée au TPM. La clé de chiffrement utilisée dérive de la donnée d'authentification de la clé parente. Par conséquent, l'écoute des communications entre la TSS et le TPM permet de retrouver la donnée d'authentification d'une clé créée sous la SRK utilisant le mot de passe connu.

En outre, même lorsque la clé SRK est associée à un mot de passe secret, il reste théoriquement réalisable d'espionner et de déchiffrer les communications protégées entre la pile TSS et le composant TPM. De telles communications sont effectuées au sein des sessions

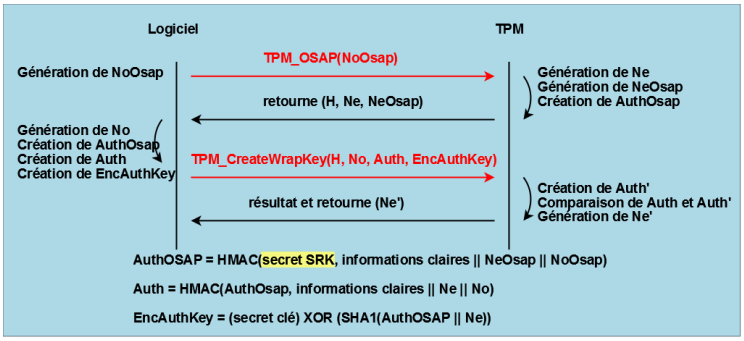

FIGURE 4. Création d'une clé sous la clé SRK

de type OSAP (*Object Specific Authorization Protocol*) ou OIAP (*Object Independent Authorization Protocol*). Or, ces protocoles ont fait l'objet d'attaques théoriques : par l'intermédiaire d'une attaque dictionnaire hors-ligne sur les communications entre le TPM et la pile logicielle, il est en effet possible de retrouver la donnée d'authentification de l'utilisateur si cette dernière est faible.

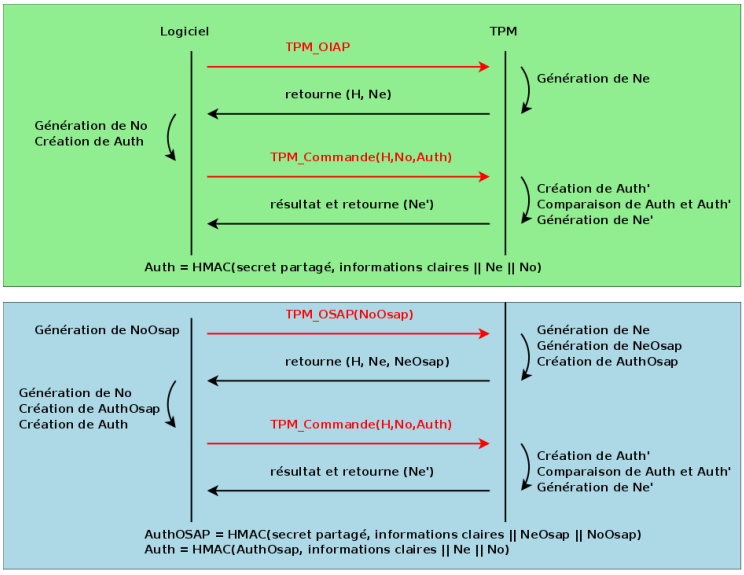

FIGURE 5. Protocole OIAP et OSAP

Néanmoins, la faisabilité d'une telle attaque dans le monde réel repose sur un pré-requis de taille : il faut pouvoir espionner les communications entre la pile TSS et le composant TPM, ce qui revient à être administrateur de la plateforme. Or, dans une telle situation, il est nettement plus simple d'espionner les communications au moment des appels à l'API de la TSS (technique d'*API hooking* par exemple). Des chercheurs de chez HP ont proposé un nouveau protocole d'autorisation [2] résistant à l'attaque par dictionnaire hors-ligne à ce type d'attaque et ont également prouvé formellement à l'aide de l'outil ProVerif. Il convient cependant d'identifer ce qui est concrêtement prouvé par cet outil.

Enfin, bien qu'il soit possible de protéger en confidentialité les communications entre la pile TSS et le composant TPM (lors d'une session chiffrée de type *Transport*), cela ne protège pas l'ensemble du chemin d'une communication entre l'application et le TPM. En effet, rien n'est proposé pour protéger les données transmises de l'application vers la bibliothèque TSP (couche supérieure de la pile TSS). Des attaques par API *hooking* lors d'appels de fonction de cette bibliothèque sont ainsi réalisables afin d'espionner les communications.

4.2 Au niveau du BIOS

Egalement dans les erreurs de conception, on peut citer les attaques par le BIOS lors des débuts du TPM 1.2. Le `S-CRTM` correspond au premier bout de code exécuté lors du démarrage du poste client. Le S-CRTM est situé dans les premiers octets du BIOS et constitue le premier maillon de la chaine de confiance. Il se mesure lui-même puis mesure le reste du BIOS avant de lui donner la main. L'attaque par le BIOS consiste à écraser le contenu du BIOS, dont le S-CRTM. L'attaquant peut ainsi instrumenter à sa guise le premier maillon de la chaine de confiance. Cette attaque exploite le fait qu'aucun mécanisme de protection du S-CRTM n'avait été spécifié. Depuis, ces attaques ont été prises en compte : les spécifications imposent au constructeur de contrôler la mise à jour du BIOS. En pratique, le S-CRTM est maintenant logé dans une zone qui ne peut-être mise à jour, tandis que le reste du BIOS peut l'être.

4.3 Au niveau de la pile TSS

La pile TSS souffre d'une complexité qui peut, soit repousser son usage par les développeurs, soit engendrer une mauvaise utilisation. De plus, la seule documentation officielle existante est le document de spécification TSS de 757 pages. Celui-ci cible à la fois le développpeur de la pile TSS et le développeur d'applications, ce qui peut apparaitre rebutant pour ce dernier. Chaque couche de la pile est détaillée de manière uniforme et avec de nombreux raccourcis. Il est ainsi souvent nécessaire de faire appel aux spécifications du composant TPM, qui sont souvent plus précises, afin de comprendre comment mettre en œuvre une fonctionnalité. Il manque donc un document auto-suffisant dédié aux développeurs d'applications, qui détaille uniquement la couche supérieure TSP.

En termes d'implémentation, la TSS libre d'IBM [15] (TrouSerS) n'est toujours pas conforme à l'ensemble des spécifications 1.2. En particulier, le protocole Direct Anonymous Attestation (DAA) n'est pas entièrement implémenté dans la dernière version en date (3.2-1). La mise en œuvre d'une attestation distante, avec le souhait d'assurer la propriété d'anonymat, est donc fonctionnellement réduite.

4.4 Un socle de confiance difficile à protéger

Une architecture de confiance repose sur un environnement de confiance (TCB ou *Trusted Computing Base*), lui même constitué d'une racine de confiance (TBB ou *Trusted Building Block*). La sécurité d'une telle architecture repose donc sur la protection du TCB dans son ensemble. Or, plus sa taille augmente, plus l'objectif de protection du TCB s'avère difficile. C'est notamment le cas pour tous les mécanismes de sécurité classiques (contrôle d'accès utilisateur, antivirus, IDS, pare-feux, etc.) qui ne fonctionnent qu'avec un TCB important : le noyau de l'OS et l'application de sécurité. Or, l'état de l'art montre la possibilité de nombreuses attaques sur le TCB (vulnérabilités dans le noyau et dans les modules, attaques hors-ligne, ect.).

En théorie, la technologie TXT doit notamment permettre de contrer les attaques hors-lignes (par exemple, modification d'intégrité du processus de démarrage). Néanmoins, plusieurs preuves de concept ont montré la difficulté à protéger le TCB, même lors de l'em-

ploi du DRTM avec Intel TXT. Au premier semestre 2009, des travaux [28,7] ont montré la possibilité d'attaques logicielles locales – reposant sur l'activation de routines de traitement ACPI ou SMI malicieuses – mettant en cause l'implémentation actuelle de la vérification locale d'intégrité (Intel TXT). Ces attaques supposent néanmoins qu'un attaquant possède les privilèges administrateur pour positioner au préalable les routines malicieuses dans la table ACPI du BIOS ou dans la mémoire SMM. Ces attaques, bien que reconnues et prochainement prises en compte par le TCG, montrent à quel point il est difficile d'établir une plateforme de confiance sur une architecture qui n'a pas été conçue dans ce sens initialement.

Plus récemment, en septembre 2009, l'implémentation d'une attaque logicielle locale [24] a mis en évidence la difficulté pour une plateforme de se prémunir contre la menace de piégeage logiciel temporaire. Rutkowska démontre en effet la faisabilité d'une attaque visant à piéger l'interface d'authentification de Truecrypt (afin de récupérer les éléments d'authentification). Bien qu'elle explique l'intérêt de la technologie TPM et du concept de chaine de confiance pour contrer les attaques par piégeage hors ligne, elle en précise également ses limites. Il est par exemple techniquement possible de piéger l'interface d'authentification de manière temporaire : une fois les données d'authentification récupérées, le maliciel peut alors s'auto-effacer, indiquer à l'utilisateur une erreur d'authentification et forcer un redémarrage afin de lancer de nouveau la procédure de vérification d'intégrité – qui sera réalisée, cette fois-ci, avec succès. En conséquence, le fameux dicton disant qu' « un système attaqué localement est un système compromis » s'avère toujours d'actualité avec l'informatique de confiance.

Egalement dans le domaine des attaques locales, l'accès physique par DMA semble être une voie privilégiée pour les attaquants, que ce soit par le port Firewire [5], que par le port PCI [4]. Néanmoins, depuis l'introduction de l'IOMMU [17] dans les chipsets NorthBridge, ces attaques ne sont plus applicables : il est en effet possible de spécifier les espaces d'adressage mémoire accessibles par les différents périphériques, et de les assigner à des domaines mémoire au sens de la virtualisation.

4.5 Problématique de gestion de parc

La gestion de parc constitue l'ensemble des procédures liées au cycle de vie des applications :

- déploiement / installation ;
- configuration ;
- mise à jour ;
- administration ;
- désinstallation.

La technologie liée au TPM complexifie considérablement l'utilisation et l'administration des applications utilisatrices. Cela contribue notamment au faible déploiement de la technologie. Le TPM présente deux problèmes intrinsèques à ses spécifications :

- le scellement (chiffrement conditionnel) ;
- la hiérarchie des clés du TPM (stockage sécurisé).

Nous rappelons que la mesure d'un élément logiciel fait l'objet d'une agrégation de la mesure dans un des registres PCR du TPM. De même, le scellement est conditionné à un ensemble (qui peut-être nul) de registres PCR.

L'utilisation du scellement impose une synergie de l'ensemble des applications réalisant les mesures. Par conséquent, le choix des registres PCR pour réaliser les mesures ou pour conditionner le scellement est crucial. En effet, un mauvais choix peut conduire à l'une des deux situations suivantes :

- L'impossibilité de desceller : par exemple, si l'utilisateur choisit de sceller un fichier en utilisant un registre PCR qui a agrégé la mesure du noyau, alors la mise à jour du noyau ne permettra plus de desceller le fichier.
- Rendre le scellement inutile : par exemple si l'utilisateur choisit de sceller un fichier en utilisant un registre PCR non utilisé dans le processus de mesure ou non pertinent vis à vis des mesures dont il a fait l'objet, alors le descellement du fichier sera possible dans un système compromis.

On s'aperçoit rapidement que l'utilisation du scellement est une tâche complexe. Elle demande à l'utilisateur une bonne connaissance du processus de mesure. Les registres PCR utilisés doivent laisser de la souplesse à la mise à jour du système. Ils doivent en même temps permettre de détecter une compromission.

Par conséquent, un produit logiciel utilisant le scellement doit pouvoir :

- Fonctionner en synergie avec une architecture logicielle de mesure et de vérification. Il est donc nécessaire de spécifier un schéma d'architecture logicielle et l'utilisation des registres PCR. Actuellement, seule l'architecture logicielle de mesure du démarrage est spécifiée par le TCG (mesures réalisées par le BIOS utilisant les registres PCR 0 à 7).
- Mettre à jour le scellement d'un objet par rapport aux éléments logiciels dont il dépend et qui sont mesurés. On constate que pour répondre à cette exigence, un outil de gestion de parc doit identifier toutes les dépendances logicielles et toutes les informations relatives aux mesures des éléments logiciels.

Ensuite, l'architecture de hiérarchie de clés du TPM constitue intrinsèquement un système complexe à gérer. En effet, l'utilisation d'une clé nécessite son chargement dans le TPM. Son chargement est conditionné au fait que l'utilisateur doit s'authentifier auprès de la clé parente. Par conséquent, l'utilisation d'une clé demande à l'utilisateur de s'authentifier plusieurs fois pour l'ensemble des clés parentes (Si elles ne sont pas chargées au préalable). De même, cette hiérarchie de clé n'est pas adaptée au contexte d'un système multi-utilisateur. Afin de répondre à cette problématique, il a été spécifié un mot de passe connu (*well-known password*). Ainsi chaque utilisateur pourrait débuter sa hiérarchie de clés sous une clé dont le mot de passe est connu par tout le monde. Comme évoqué précédemment, l'utilisation du mot de passe connu pose plusieurs problèmes de sécurité.

De plus, il est important de noter que la destruction d'une clé rend impossible l'utilisation des clés filles. Par conséquent, il est nécessaire d'utiliser des clés migrables (non utilisées pour le scellement) et de les migrer avant la destruction de la clé mère.

Enfin, les clés créées par le TPM ne sont utilisables que par ce dernier. Par conséquent, l'utilisation du TPM pour la protection des données nécessite la réalisation d'une procédure de recouvrement des clés. En effet, un dysfonctionnement matériel du TPM ou de la carte-mère (surlaquelle est généralement soudé le TPM) provoquera l'impossibilité de déchiffrer les données.

4.6 Futures évolutions du TPM

Le TCG a publiquement annoncé les nouveautés de la future génération de TPM [11]. L'une des principales évolution est relative à l'ajout de nouvelles suites d'algorithmes cryptographiques respectant l'état de l'art en cryptologie. Ces modifications seront complétés par une flexibilité de l'algorithmie puisque les futures spécifications prévoient la possibilité d'ajouter de nouveaux algorithmes sans modification préalable de l'interface.

Par ailleurs, la spécification des mécanismes de virtualisation du TPM reste actuellement problématique. Depuis la dernière spécification TPM, les besoins en virtualisation se sont développés et la spécification du TPM 1.2 n'est intrinsèquement pas conçue pour supporter la virtualisation. La nouvelle génération de TPM spécifiera donc des fonctionnalités facilitant leur intégration dans des architectures de virtualisation notamment par l'utilisation de plusieurs hiérachies de stockage. Il sera alors possible d'associer une hiérarchie de stockage par machine virtuelle. La gestion de ces hiérarchies étant déléguée au gestionnaire de machine virtuelle.

Afin d'améliorer les performances de l'utilisation des hiérarchies de clés, le TCG prévoit deux aménagements majeurs s'appuyant sur la cryptographie symétrique. Le premier aménagement est la protection des clés asymétriques avec la cryptographie symétrique. Le deuxième aménagement concerne la protection des données. Les futures spécifications prévoient l'utilisation de clés symétriques de session pour protéger les données. Cette clé de session est ensuite protégée avec la cryptographie asymétrique.

Pour se prémunir contre les attaques sur l'authentification du TPM (cf. chapitre précédent), une alternative plus robuste aux protocoles OSAP et OIAP sera proposée. Elle reposera sur une graine secrète permettant de se prémunir des attaques hors ligne par dictionnaire.

Enfin, le TCG prévoit de simplifier l'administration du TPM en proposant notamment un simple mode "marche"/"arrêt" à la place des modes *enable* et *activate* du TPM. De même, les futures spécifications distingueront les fonctions dédiées à la protection des données et les fonctions dédiées à l'identité.

5 Intégrité du système au démarrage

La suite de l'étude est relative au positionnement de l'informatique de confiance en général, et du composant TPM en particulier. Il s'agit de voir comment répondre aux besoins de vérification locale d'intégrité et d'exécution confidentielle de code, en construisant une architecture sécurisée à base de TPM. Les trois thèmes suivants sont développés dans cet article : intégrité du système au démarrage, intégrité du système pendant son fonctionnement et exécution confidentielle de code.

5.1 Solutions existantes

Trusted Grub Depuis 2005, IBM a développé une architecture complête de mesure d'intégrité [26] se basant sur le composant TPM. Cette architecture se compose d'une part, d'un boot loader, et d'autre part, de modules noyaux (IMA et EVM) qui sont abordés dans le prochain chapitre.

Trusted Grub [14] est un boot Loader intégrant des fonctionnalités de mesure d'intégrité de type SHA-1. Il utilise la puce TPM pour stocker, dans les registres PCR, les mesures d'intégrité des éléments de la chaine de démarrage (noyaux, modules, initrd et fichiers de configuration). Ce projet libre permet ainsi la continuité de la chaine de confiance statique (SRTM), initiée par le BIOS et le CRTM.

Cependant, la version actuelle, à la date de l'écriture de l'article, ne réalise pas correctement l'extension des registres PCR, ce qui est pourtant nécessaire à l'établissement d'une chaine de confiance réellement efficace. Quelques modifications dans le code source sont ainsi nécessaires pour obtenir une chaine de confiance opérationnelle.

OSLO OSLO est un boot loader développé par Bernhard Kauer [20] en 2007. Il s'agit de la première implémentation publique de la technologie DRTM. OSLO fonctionne sur une architecture de type AMD SVM et tire parti de son jeu d'instruction SKINIT pour initier une chaine de confiance dynamique. Cette preuve de concept montre qu'il est possible d'obtenir une mesure de l'intégrité d'un OS, indépendamment de la plateforme matérielle – ou, plus exactement du BIOS et des microcodes. Cette dépendance est en effet délicate à prendre en charge sur les parcs informatiques.

Trusted Boot Trusted Boot est un boot loader développé par Intel [18] depuis 2007, et qui a été intégré officiellement au noyau Linux en décembre 2009 dans sa version 2.6.32. Trusted Boot est la première implémentation publique de la technologie Intel TXT [16]. Il supporte le lancement de Linux et Xen au sein d'une chaine de confiance dynamique. Plus précisément, il est désormais possible de démarrer Linux ou Xen avec une vérification de son intégrité par rapport à un politique de référence. L'OS est protégé contre le piégeage logiciel des éléments de la chaine de démarrage, de même que contre les attaques matérielles de type DMA (protection IOMMU).

5.2 Travaux relatifs

Il existe des chargeurs sécurisés alternatifs, qui ne reposent pas sur la technologie TPM. Nous proposons d'en décrire deux dont le fonctionnement nous apparait pertinent.

Sboot Sboot est une extension du chargeur de démarrage Grub. Cette extension a été développée par Mandriva lors du projet EConfidential [22]. Sboot permet de sécuriser l'accès aux données d'un poste client. Il offre les deux fonctions de sécurité suivantes :

- **Le mécanisme de vérification d'intégrité** : Ce mécanisme a pour objectif de vérifier l'identité et le contenu de plusieurs composants du poste client. La vérification d'identité porte sur le processeur, les cartes réseaux, les disques durs et les cartes PCI. La vérification du contenu porte sur le BIOS, le répertoire `/boot`, le MBR, le noyau et certains fichiers de configuration sensibles (`/etc/security/*`, `etc/sysctl.conf`...).
- **Le mécanisme d'authentification de l'utilisateur** : Une fois la vérification d'intégrité réalisée, ce mécanisme se charge d'authentifier l'utilisateur et de libérer une clé de chiffrement permettant l'accès à une partition chiffrée.

La clé de chiffrement de la partition de données est stockée sur une clé USB. Cette clé est protégée par un login et un mot de passe.

Chromium OS Verified Boot Le projet de système d'exploitation développé par Google [9] repose sur une version allégée et renforcée de Linux. La sécurité est présente à tous les niveaux : réduction

du nombre d'applications – en pratique, l'utilisateur est cloisonné au navigateur Chromium –, isolation des processus, durcissement des processus et du noyau – mécanisme de canari pour protéger la pile, randomisation des espaces d'adressage mémoire, utilisation du bit NX pour protéger les pages mémoire non exécutables, et bien d'autres fonctionnalités.

En complément des mécanimes protégeant le noyau et les applications, une chaine de confiance est initiée par le chargeur puis propagée jusqu'au lancement du noyau afin de vérifier l'intégrité de chaque élément de la procédure de démarrage. Ce mécanisme de mesure d'intégrité reprend le modèle de la chaine de confiance statique spécifié par le TCG. Une différence tout de même : aucun support TPM n'est requis. La racine de la chaine est uniquement constituée d'un *firmware* stocké dans une région d'une mémoire EEPROM protégée en écriture.

La procédure *Verified Boot* de Chromium ne vérifie que l'intégrité des éléments logiciels propres à l'OS. Aucune mesure de l'environnement matériel n'est réalisée. En particulier, les microdes des cartes PCI et du processeur ne sont pas pris en compte, contrairement à la chaine de confiance SRTM spécifiée par le TCG. De fait, il est théoriquement possible d'insérer un microcode piégé sans modifier le résultat de la vérification d'intégrité.

5.3 Couverture des objectifs de sécurité

En bilan, le tableau 1 présente la couverture des objectifs de sécurité, définis au début de l'article, par les solutions issues de l'informatique de confiance et par les solutions alternatives.

6 Intégrité du système pendant l'exécution

6.1 Solutions existantes

IMA IMA [13] est un module noyau permettant au système d'exploitation de mesurer l'intégrité des binaires chargés en mémoire : pilotes, bibliothèques et exécutables. IMA est développé par le groupe de recherche d'IBM sur le TCG. Il a été intégrée au noyau Linux dans sa version 2.6.30. Il est complémentaire au projet Trusted Grub, en

TABLE 1. Bilan des solutions assurant l'intégrité au démarrage

Objectif	Solutions "Trusted Computing"			Solutions alternatives		
	Nom	Intérêt	Limites	Nom	Intérêt	Limites
Intégrité du démarrage de l'OS	Oslo	Supporte le DRTM	Pas de vérification d'intégrité	Chromium	Indépendant du TPM	Robustesse liée au contrôle d'accès au firmware
	Trusted Boot	Applique une politique dépendante de l'intégrité	Lourdeur de la gestion de la politique			
Intégrité de la chaine complète de démarrage	Trusted Grub	Première véritable implémentation du TC dans un boot loader	Pas de vérification d'intégrité. Non intégré à Grub	Sboot	Indépendant du TPM	Piégeage logiciel possible

ce sens où il complète la chaine de confiance initiée par le boot loader. Le fonctionnement d'IMA est transparent pour l'utilisateur. Lors du chargement du binaire en mémoire, IMA le mesure. S'il s'agit de la première exécution du binaire depuis le démarrage du système, IMA stocke le résultat de la mesure dans un registre PCR, de même que dans un fichier de référence, appelé SML (*Stored Measurement Log*). Si le binaire a déjà été utilisé, IMA compare le motif d'intégrité avec celui présent dans le fichier de référence. Quelque soit le résultat de ce test, IMA n'empêche pas l'utilisation des binaires. En revanche, il indique le nombre de changements d'intégrité pour chaque binaire

IMA n'est qu'une brique logicielle qui trouve son utilité qu'avec une application tierce interprétant les résultats et agissant en conséquence. Il peut s'agir, par exemple, de bloquer l'exécution du binaire, d'émettre une alerte ou encore d'attester de l'état de la plateforme à un serveur distant.

EVM Le module Linux EVM (*Extended Verification Module*, développé par IBM, permet d'appliquer une politique de sécurité en fonction de l'intégrité des binaires exécutables. A chaque binaire est associée une mesure d'intégrité de référence et une politique précisant si le binaire est autorisé ou non à être exécuté en cas de mauvaise intégrité. Ces attributs de sécurité sont stockés dans les méta-données du fichier. L'intégrité de la mesure de référence est assurée par la

```
10 03ee0cce68447c947f8e8eb37881c27570347fa4 boot_aggregate
10 ea8239dfed9dd11bd538f9c3234e0d7b71672fff /bin/sh
10 ebb4f3db0b83c1e717e3d05f702e4608a9c2ea08 /lib/ld-linux.so.2
10 671aba5cb6df951463e57c963e8c327fc6cfb5ab /lib/libcrypt.so.1
10 445babe91e586090cb7f9782b44ba115ceec6b7f /lib/libm.so.6
10 411ab19e995e06b1d7378e1daea9926ccba1ea20 /lib/libc.so.6
10 68b297cd8fe07a3e54e6ce9d2e66afa799e472a3 /sbin/depmod
10 407285ba377ea035f2ff6d61c47ead8befa7cd5f /sbin/modprobe
10 ee3295941fd593c95c03268ff961036d248dd8c0 thermal_sys
10 432f769f5864adc1bc5daf17fbdf2e8c99e7ffd4 fan
10 ccfaf0b02d678c9b3f6eba54968e832505a50793 processor
10 033d934e266c6aff719278a28d6dd7e74fe70c6c thermal
10 b3f9dc0a1cc001f88d43609c7dea156d280deca8 /sbin/udevd
10 91700dcba332de54094beef00fea0e11dd414b9f /lib/libselinux.so.1
10 8abd11a221cd68b6f544e8a4b74a59dfbffd839c /lib/libdl.so.2
10 cea55a2390cfc6c4947e980be3aa457cea1ec335 /lib/libsepol.so.1
10 78750ed652666c01f4e7de51768b673e217c536b /sbin/udevtrigger
10 07bcb87e7acca43b0f3f0ba67f480402519f87b3 /sbin/udevsettle
10 08e158dc445c50aa32b1e1adbed2a3f13828ee2d sd_mod
10 1cc363d95a64c3ebf26127355d894724b0415a53 /bin/true
10 cdce54d429fe002b7377da94c74c6aa1014f6f4f /lib/klibc-rOj3PRLKBA9FcF5ZuoqKQLmOWcA.so
10 ecbc9f0925e1f7ec370fec9082ab30d4111f9df2 usbcore
10 8d4105ed57deb1a417805a88a1a5c3df332273ba ieee1394
10 8631dc01e9e12aa1af8be015874d2cf53cd8475c ehci_hcd
10 ed1937e5f4bf1e3c07949df24981d538bce24618 /lib/udev/usb_id
10 5546a5dcad4b9f77ab05d310daa3ae1a3d503e69 ide_core
10 8c818fb30300cf0ea4c278a52c3c671f50f2a529 /lib/udev/scsi_id
10 df2c5070d5d3322009086cd3d464d30473b0666e libphy
10 3988ef17d2e9264a671d0c84135a7dda852cd977 /lib/udev/edd_id
10 f774ca3c6a5a55ace882946548d0bf6f6bf03e4a ohci1394
10 d7190161192e5ce5ea32405fe5abdcf81c4221f0 /lib/udev/vol_id
10 e76488eaae2df61955da3bb3e8f087212a99307c piix
10 fc89d5eb576dcad375163f710807eb1e6f4ebbe3 /lib/libvolume_id.so.0
10 d34755fcb7c6e18480ef90c2eff55c52d7f7e0a9 uhci_hcd
10 ff82002ad570af494266f7b16ba4a9d5080cefc2 tg3
10 0d339ef60e3d2bd4d9ec11f38972ff5eac9e4eb3 ata_generic
10 32b6258f0e46ef4669d3e1573096244acb078181 usbhid
```

Figure 6. Fichier SML d'IMA

fonction HMAC-SHA1 du TPM. EVM offre ainsi un service de sécurité qui complète l'architecture de sécurité d'IBM (Trused Grub et IMA).

SecVisor Le projet SecVisor est développé depuis 2007 par le laboratoire Cylab de l'université Carnegie Mellon à Pittsburgh. SecVisor est un hyperviseur qui assure l'intégrité du noyau d'un OS pendant son exécution [27]. Les auteurs proposent quatre propriétés d'intégrité du noyau (ou code légitime) à assurer :

- P1 : lors de l'entrée dans le mode noyau, le registre contenant le pointeur d'instruction (IP) du processeur doit pointer vers une adresse mémoire correspondant à du code légitime ;

- P2 : pendant toute la durée d'exécution en mode noyau, le registre IP doit toujours pointer vers du code légitime ;
- P3 : lors de la sortie du mode noyau, le niveau de privilège du processeur (registre CPL) doit être positionné en mode utilisateur ;
- P4 : il ne doit pas être possible de modifier l'espace mémoire du noyau contenant le code légitime.

Bien qu'une solution uniquement à base d'hyperviseur soit suffisante pour assurer l'intégrité des propriétés P1 et P3 du noyau, la technologie SVM – avec son mécanisme IOMMU – offre la possibilité de protéger la zone de code légitime (propriétés P2 et P4) conjointement avec la MMU du processeur. Cette MMU permet de spécifier les pages mémoires accessibles en lecture et/ou écriture par le processeur (lors d'une demande réalisée par une application ou le noyau du système invité). L'IOMMU bloque l'accès en lecture depuis les périphériques lors de transferts DMA. Ce dernier mécanisme, appelé DEV (*Device Exclusion Vector*), repose sur un vecteur spécifiant, pour chaque page de la mémoire physique, les droits d'accès du périphérique.

Pour être efficace, le contrôle d'accès mémoire porte non seulement sur l'image en mémoire du noyau invité (le code légitime), mais également sur le code de l'hyperviseur SecVisor et le vecteur DEV de l'IOMMU. En outre, l'hyperviseur SecVisor se positionne comme une racine d'une chaine de confiance dynamique (DRTM), ce qui lui permet de fonctionner indépendamment des éléments qui lui précèdent (BIOS, boot loader, OS, etc.).

A noter qu'en 2008, le code de SecVisor a été vérifié formellement [8]. Ceci a été réalisable du fait de la taille minimaliste de l'hyperviseur (de 3500 à 5000 lignes de code en C et en assembleur).

6.2 Couverture des objectifs de sécurité

En bilan, le tableau 6.2 présente la couverture des objectifs de sécurité, défini au début de l'article, par les solutions de l'informatique de confiance et par les solutions alternatives.

1. http://www.ghs.com/products/rtos/integrity_virtualization.html
2. http://www.vmware.com/technical-resources/security/vmsafe/security_technology.html

Objectif	Solutions "Trusted Computing"			Solutions alternatives		
	Nom	Intérêt	Limites	Nom	Intérêt	Limites
Intégrité des applications au démarrage	IMA	Mesure tous les binaires exécutables chargés en mémoire	Uniquement de la mesure d'intégrité	HIDS pour le contrôle d'intégrité (Tripwire, Samhain)	Fonctionne sur tous les types de fichiers	Pas de protection évoluée sur les mesures de référence
	EVM	Signature des mesures de référence à l'aide du TPM				
Intégrité des applications pendant l'exécution				PaX (restriction MPROTECT)	Protège contre la modification du code exécutable	Ne protège pas contre une modification directe et externe d'une application
Intégrité du noyau pendant l'exécution	SecVisor	Gartantit l'intégrité du code noyau	Ne garantit pas l'intégrité du flot de contrôle et des données noyau	Hytux [21] (hyperviseur protégeant le noyau contre les actions malveillantes)	Vise à protéger le code et les structures du noyau contre les rootkits	Difficulté pour identifier un ensemble exhaustif de contraintes noyau
				INTEGRITY Hyperviseur industriel certifié EAL6+[1]	Cloisonnement virtuel des applications sensibles	Pas de vérification d'intégrité
				Framework de sécurité VMSafe pour VMWare[2]	Permet le contrôle des environnement virtuels afin de lutter contre les codes malveillants	Non libre

Table 2. Bilan des solutions assurant l'intégrité d'exécution

La vérification locale d'intégrité repose sur un socle de confiance (CRTM - Core Root of Trust for Measurement), soit intégré au BIOS de la carte mère et initié de manière statique (S-CRTM), soit présent en mémoire cache du processeur et initié de manière dynamique (D-CRTM). Chaque approche possède ses propres avantages et inconvénients. Le SRTM intègre, lors de l'étape de mesure d'intégrité, des éléments matériels (BIOS de la carte mère, microcodes des cartes PCI et du processeur, etc.) à la chaine de confiance, ce qui a pour effet de lier les résultats obtenus, et les services qui s'en servent (dont la vérification d'intégrité), à la plateforme matérielle. Au contraire, le DRTM permet d'établir une chaine de confiance indépendante de la plateforme matérielle. Il est donc important de prendre en considération ces effets lors de l'utilisation du mécanisme de mesure d'intégrité pour les applications haut-niveaux (chiffrement conditionnel, vérification d'intégrité, etc.).

En ce qui concerne le modèle de l'attestation d'intégrité, il nécessite trois acteurs : l'élément souhaitant prouver son intégrité, l'élément vérifiant la validité de l'intégrité et l'élément a qui on atteste la validité de l'intégrité. Dans le cadre de l'attestation distante, les deux derniers rôles peuvent être joués par la même entité, sans que cela ne remette en cause la pertinence de l'attestation. En revanche, dans le cadre de l'attestation locale, les trois rôles doivent être joués par une même plateforme. Ceci apparait incompatible avec les principes de sécurité et de séparation des rôles sous-jacent au modèle d'attestation.

Le prochain chapitre présenter quelques réflexions sur l'amélioration du processus de vérification local d'intégrité, et évoquera quelques pistes pour s'orienter vers de l'attestation locale – principe essentiel pour assurer la confiance de l'utilisateur envers sa plateforme.

6.3 Analyse et perspectives

A l'heure actuelle, la mise en place d'un service d'attestation distante global (par exemple, à travers Internet) apparait illusoire car il repose sur plusieurs pre-requis difficile à mettre en œuvre : PKI à grande échelle, base de donnée complète des éléments à mesurer et maintien de cette base dans le temps. Les perspectives s'orientent

donc soit vers un service d'attestation distante sur un réseau fermé et maitrisé [1] soit vers une architecture de vérification locale d'intégrité.

L'objectif de cette dernière serait alors de garantir à l'utilisateur que son poste client est intègre et non-piégé. Néanmoins, la spécification d'une telle architecture nécessite de répondre aux problématiques suivantes :

1. Comment assurer le lancement du processus de vérification ?
2. Comment assurer l'intégrité du processus de vérification locale d'intégrité ?
3. Comment assurer une communication de confiance des résultats à l'utilisateur ?

Comment assurer le lancement du processus de vérification ? La résolution de cette problématique nécessite une spécification préalable du processus de vérification. Plusieurs choix sont possibles :

- *Initiation du processus par l'utilisateur* ;
- *Initiation automatique du processus.* L'initiation est effectuée, soit à intervalle de temps régulier, soit à la demande du système pour la réalisation de certaines opérations ciblées (par exemple pour la demande du mot de passe de l'utilisateur).

Dans le premier cas de figure, une demande de vérification provenant de l'utilisateur nécessite l'affichage d'un résultat de la vérification. Ce cas est donc équivalent à la problématique de la communication de confiance des résultats à l'utilisateur puisque sa résolution implique alors que l'utilisateur est en mesure de déterminer si le processus de vérification a bien été effectué.

Dans le second cas, la réalisation d'un lancement automatisé nécessite la définition du mode de communication des résultats de la vérification à l'utilisateur. En effet, il n'est pas envisageable de saturer le système par la génération intempestive de messages indiquant à l'utilisateur que son système est intègre. Par conséquent on peut distinguer deux types de processus automatisés :

1. **Processus automatisé communiquant systématiquement les résultats de vérifications.** Ce type de processus convient

1. Typiquement sur un réseau d'entreprise, comme évoqué avec TNC

dans les situations où le système a besoin de réaliser de manière ponctuelle une opération sensible à la demande de l'utilisateur (demande du mot de passe de l'utilisateur). Le système peut alors assurer l'utilisateur que l'environnement est de confiance. Ainsi, ce cas de figure rejoint celui où l'utilisateur demande une vérification. Par conséquent, ce cas est équivalent à la problématique de la communication de confiance des résultats à l'utilisateur.

2. **Processus automatisé communiquant seulement les résultats d'une perte d'intégrité du système.** Ce type de processus est relatif à une vérification planifiée automatiquement à intervalle de temps régulier sans intervention de l'utilisateur. La spécification d'un tel processus est complexe, puisque l'utilisateur ne pourra être prévenu de la désactivation du mécanisme. Il convient, de plus, de sécuriser le mécanisme de lancement du processus de vérification. Plusieurs approches sont possibles et chacune d'entre elles offrent leurs avantages, leurs inconvénients et leurs hypothèses de fonctionnement :
   - **Utilisation des interruptions matérielles.** Une première approche serait de déclencher le processus de vérification par le biais d'une interruption matérielle récurrente : *timer* de la carte mère par exemple. Toutefois, il devient nécessaire de protéger la table des vecteurs d'interruption et le traitant de l'interruption. Sur un système d'exploitation moderne, cela est complexe à mettre en œuvre si la menace sur la compromission en mode noyau est prise en compte. Etant donné que la menace provient du noyau, il est possible de l'éluder si l'on considère que le noyau est de confiance (utilisation d'une politique de sécurité, ...).
   - **Utilisation des interruptions matérielles dont les traitants sont difficilement accessibles.** L'idée est sensiblement similaire à la première approche. Dans ce cas de figure, on envisage l'utilisation d'interruption matérielle dont les traitants ne sont pas accessibles par le système d'exploitation. Cela permet d'éluder la désactivation du processus de vérification par le biais de la compromission du noyau. De telles interruptions matérielles existent : interruptions SMI utilisées notamment pour la gestion de l'alimentation du poste client.

Les traitants de ces interruptions sont situés dans une zone de mémoire appelée SMRAM. Les traitants de ces interruptions sont fixés au démarrage par le BIOS. Dès que les traitants sont positionnés, le BIOS verrouille la SMRAM à l'aide d'un mécanisme de contrôle d'accès. Ce mécanisme empêche toute modification de la SMRAM par une application s'exécutant sur le processeur. Seuls les traitants des interruptions SMI (System Management Interruption) peuvent modifier la SMRAM. La génération d'une interruption SMI provoque le basculement du processeur en mode SMM (System Management Mode) afin d'exécuter le traitant d'interruption adéquat. Le mode SMM est un mode privilégié des processeurs X86 à partir duquel il est possible d'accéder à l'ensemble de la mémoire physique du poste client. L'utilisation des interruptions SMI et de la SMRAM pour loger le processus de vérification est très intéressant car il est difficilement attaquable par un *rootkit* depuis le système d'exploitation. Toutefois, cette solution présente plusieurs inconvénients :

- En mode SMM, le processeur accède physiquement à la mémoire et n'utilise pas le mécanisme de mémoire virtuelle. Par conséquent, le processus de vérification doit adapter sa vision de la mémoire à celle du système d'exploitation qui utilise la mémoire virtuelle.
- La publication [7] montre qu'il est possible de modifier la SMRAM si cette dernière est mise dans le cache du processeur. En effet, le mécanisme de contrôle d'accès est situé au niveau du chipset de la carte mère et ne concerne que la mémoire. Par conséquent, si on spécifie que la SMRAM peut-être mise dans le cache du processeur, il existera une deuxième version de la SMRAM qui ne sera pas protégée et donc modifiable par un attaquant.

– **Utilisation d'un hyperviseur :** La virtualisation peut largement contribuer à répondre à cette problématique. En effet, dans la mesure où un système virtualisé ne peut *a priori* pas accéder à l'hyperviseur, il est donc intéressant d'y loger le processus de vérification d'intégrité. Toutefois, il sera nécessaire d'avoir un processus de vérification adapté au système d'exploitation virtualisé (Linux, Windows...).

Comment assurer l'intégrité du processus de vérification locale d'intégrité ? Cette problématique peut-être résolue en grande partie relativement aux choix évoqués précédemment. En effet, le fait de rendre inaccessible au système d'exploitation le processus de vérification (utilisation d'un hyperviseur ou des interruptions SMI) garanti *de facto* son intégrité.

Toutefois, s'il n'est pas possible d'empêcher le système d'exploitation d'accéder au processus de vérification, il est possible d'utiliser le D-CRTM pour lancer un environnement de confiance pour le processus de vérification. Lors du lancement du D-CRTM, tous les processeurs à l'exception d'un seul sont stoppés. Le dernier processeur possède l'exclusivité et peut exécuter le code de confiance. Cette propriété est intéressante et peut-être utilisée pour attester l'intégrité du processus de vérification et son bonne exécution.

Remarque : Par rapport à l'utilisation du mode SMM du processeur évoquée dans la partie précédente, le D-CRTM n'est pas utilisable dans ce mode de fonctionnement. En effet la commande SKINIT (GETSEC[SENTER] pour les processeurs Intel) ne peut-être exécutée qu'en mode réel ou protégé.

Comment assurer une communication de confiance des résultats à l'utilisateur ? Cette problématique est cruciale car sa solution a pour objectif de garantir à l'utilisateur que les résultats de la vérification sont bien ceux qu'ils voient. En effet, les attaques de type usurpation d'interfaces d'authentification peuvent être utilisées pour masquer les résultats. Ce type d'attaque est intéressant car elles ne sont pas nécessairement couteuses à mettre en œuvre et peuvent très efficace face pour tromper les utilisateurs.

La problématique de la communication sécurisée avec l'utilisateur reste, à ce jour, totalement ouverte. Des efforts de l'industrie sont en cours, en particulier de la part d'Intel avec leur projet *Intel Trusted I/O*, visant à sécuriser les interfaces d'entrées/sorties avec l'utilisateur. Cependant, aucune spécification, ni implémentation probante n'ont été rendues publiques jusqu'à présent.

7 Confidentialité d'exécution

7.1 Solutions existantes

Flicker La preuve de concept Flicker, réalisée par le laboratoire Cylab de Carnegie Mellon, démontre la possibilité d'exécuter une portion de code de manière isolée du système d'exploitation sous-jacent. Grace au mécanisme de DRTM, le Secure Loader Flicker peut être initié lorsqu'un OS est en cours de fonctionnement. L'instruction *skinit* (sur les architectures AMD SVM) permet de mettre au pause l'OS, d'exécuter une portion de code sensible – appelé PAL (*Pieces of Application Logic*) par les auteurs –, puis de réactiver l'OS. A noter que le projet a récemment été porté sur la technologie Intel TXT.

TrustVisor Le projet TrustVisor [23] est également développé par le laboratoire Cylab de Carnegie Mellon. Son objectif est d'assurer l'intégrité et la condidentialité des données et des applications utilisateur sensibles vis-à-vis de l'OS sous-jacent ; cet OS n'étant pas considéré de confiance dans le modèle de TrustVisor.

Cette preuve de concept repose sur un hyperviseur minimaliste tirant parti de la technologie AMD SVM pour exécuter du code (PAL) de manière isolée de l'OS. Lors du lancement d'un PAL, un hypercall est envoyé vers l'hyperviseur afin de passer en mode d'exécution sécurisée. La mémoire contenant le PAL est alors isolée du reste de l'OS en la positionnant dans un domaine virtuel distinct. Ce domaine est protégé, à l'aide de l'IOMMU d'une part, contre les accès DMA, et d'autre part, contre les accès du domaine contenant l'OS.

Le DRTM est mis en œuvre pour s'assurer de l'intégrité de l'hyperviseur lors de son démarrage. En outre, une instance logicielle du TPM est utilisée afin d'accéler les changements de contextes lors du (dé)chargement du PAL (déchiffrement réalisé avec les primitives cryptographiques du TPM). Il s'agissait en effet d'une limitation importante dans le projet Flicker lors du changement de contexte entre le Secure Loader et l'OS. Le fait d'émuler un TPM diminue le niveau de robustesse de cette architecture, mais la rend plus performante.

P-MAPS P-MAPS est un projet de Intel [25], qui repose sur la technologie TXT et la virtualisation matérielle pour protéger en confidentialité l'exécution d'une application vis-à-vis d'un OS. La cible de ce projet concerne les mobiles et les postes client. Bien qu'il n'existe aucune implémentation publique à l'heure actuelle, les informations disponibles montrent une similarité avec le projet TrustVisor. P-MAPS profite, en plus, des services de sécurité supplémentaires de la technologie Intel TXT.

P-MAPS repose sur un hyperviseur dont le lancement, avec un DRTM, est réalisé par un chargeur fonctionnant sous l'OS non de confiance. Lorsqu'une application sensible souhaite être exécutée de manière protégée vis-à-vis de l'OS, elle s'enregistre auprès du chargeur. Dès lors, l'accès à ses propres pages mémoire dans le domaine virtuel de l'OS provoque une faute de page. L'hyperviseur P-MAPS prend alors la main et crée un domaine virtuel dédié à l'application. Cette dernière reprend ensuite le contrôle et s'exécute de manière isolée.

P-MAPS permet d'assurer plusieurs propriétés de sécurité :

- restriction d'accès en écriture sur les pages de code/données de l'application sensible de la part de l'OS ;
- restriction d'accès en lecture sur les pages de code/données de l'application sensible de la part de l'OS ;
- partage de pages mémoire entre l'OS et l'application sensibles (lorsque cette dernière fonctionne en mode protégé) ;
- restriction sur les adresses des points d'entrée autorisés pour les fonctions de l'application sensible.

De plus, si l'application souhaite manipuler des données confidentielles, elle a la possibilité d'utiliser les primitives de scellement (chiffrement conditionné) du TPM pour protéger ces données contre une attaque en analyse de la mémoire.

L'hyperviseur P-MAPS est capable de gérer plus instances d'applications sensibles en parallèle. Une limitation importante concerne le temps nécessaire à la réalisation du DRTM. Le temps de chargement de l'hyperviseur P-MAPS est estimé à 300 msec, dont l'essentiel provient des intéractions avec le composant TPM.

7.2 Bilan, analyse et perspectives

Toutes les solutions présentées ci-dessus permettent au minimum d'assurer la confidentialité d'exécution vis-à-vis du système d'exploitation et vis-à-vis de certaines attaques matérielles (accès DMA). Le processus de DRTM associé aux mécanismes de démarrage sécurisé et de protection IOMMU offrent un niveau de robustesse satisfaisant, au prix d'un impact identifiable sur les performances.

La confidentialité d'exécution peut être efficace dès lors que l'environnement de confiance, le TCB, a une taille limitée. La preuve de concept Flicker montre qu'il est déjà possible d'exécuter du code dans un environnement cloisonné et indépendant du noyau de l'OS. Ou en partie seulement... En effet, le lancement du *Secure Loader* nécessite l'exécution d'une instruction spécifique, typiquement à partir du noyau d'un OS – instruction GETSEC[SENTER] sous Intel ou SKINIT sous AMD. La compromission du noyau peut donc engendrer, dans le pire des cas, un déni de service empêchant l'exécution du *Secure Loader*.

Il est donc essentiel de pouvoir empêcher un tel déni de service (c'est-à-dire empêcher un attaquant de bloquer l'exécution de l'instruction lançant le *Secure Loader*). Sur ce thème, plusieurs approches ont été discutées dans le chapitre précédent.

Un autre problème se pose : l'exécution d'un code sécurisé à l'aide de la technologie Intel TXT monopolise le processeur. Il y a donc un besoin de pouvoir avoir une exécution, sinon alternée, au mieux parallèle entre l'OS et le code sécurisé. C'est précisément ce qui est mis en œuvre au sein de la console PS3. En effet, cette architecture dédie spécifiquement un cœur du processeur Cell à un processus sécurisé. La multiplication des cœurs sur les processeurs actuels devrait donc théoriquement permettre de réaliser un tel mécanisme sur les plateformes PC sans impacter de manière importante l'architecture et le fonctionnement de l'OS.

Enfin, une approche alternative pour permettre l'exécution sécurisée est de profiter du mécanisme de cloisonnement offert par les technologies de virtualisation. Une telle approche aurait pour avantage d'être applicable sur les architectures matérielles actuelles (avec ou sans support de la virtualisation matérielle).

TrustVisor permet la vérifiation d'intégrité et la protection en condidentialité d'un code pendant son exécution. Cette preuve de concept tire partie de la technologie Intel TXT et de la virtualisation afin de mettre en œuvre un hyperviseur de sécurité.

L'approche de TrustVisor, bien que satisfaisante d'un point de vue performance, pose quelques soucis de robustesse. En effet, l'exposition aux vulnérabilités est théoriquement plus importante, du fait d'une plus grande surface de code. C'est typiquement le cas avec des hyperviseurs comme Xen, VMware ou même KVM. De plus, l'hyperviseur est quelque fois associé à un domaine de gestion (par exemple, Xen et son Dom0). L'exposition aux vulnérabilités est alors augmentée, sans parler des risques liés aux consoles d'administration des machines virtuelles : la faiblesse du modèle est ainsi en partie déportée sur la robustesse du contrôle d'accès à ces domaines de gestion. En outre, dans la plupart des cas, l'hyperviseur peut être légitimement contrôlé par l'administrateur de la plateforme. De fait, les besoins liés à la protection de la propriété intellectuelle ne sont plus applicables.

En comparaison avec les solutions alternatives, il convient de citer les processeurs sécurisés présents dans certaines architectures du marché (Xbox 360 et PS3), de même que certains travaux académiques (Aegis, Hide, TrustZone, Dallas et CryptoPage [6]). Ces réalisations démontrent la faisabilité d'un exécution confidentielle de code reposant sur un cloisonnement uniquement matériel. Mais ceci se fait au prix d'un impact important sur l'architecture système et sur le cout de production. Citons également les co-processeurs cryptographiques tels que les cartes PCI actuellement distribuées par IBM. A titre illustratif, la carte cryptographique IBM 4764 est proposée à 8,600 $, qu'il convient de mettre en perspective face aux quelques dollars pour le cout d'acquisition d'un composant TPM.

De fait, l'usage d'un processeur sécurisé apparait uniquement applicable pour des environnements fermés (console de jeux, matériel embarqué, téléphone mobile,...). En effet, ces environnements facilitent particulièrement la gestion transparente et fermée des clés cryptographiques et des licences. Mais ceci n'est pas transposable aux plateformes de type PC qui sont, par essence, ouvertes.

8 Conclusion

Malgré la standardisation ISO des spécifications TPM [19], les concepts de sécurité promus par le TCG sont aujourd'hui peu implémentés par les industriels. Le composant TPM et sa pile logicielle demeurent les seules réalisations largement déployées sur les parcs informatiques grands publics et professionnels. Mais très peu d'applications industrielles tirent réellement parti des services offerts par le TPM.

Un souci important concerne les applications utilisant le composant TPM et leur manque de maturité industrielle. Elles souffrent régulièrement de la présence d'erreurs d'implémentation (bogues et vulnérabilités logiciels). Cela est en partie dû à la jeunesse de la technologie et, dans certains cas, au manque de maitrise des spécifications de la part des développeurs. On peut prendre pour exemple le cas de l'implémentation de l'informatique de confiance sous Linux. En effet, de nombreux logiciels utilisant le TPM sont encore incomplets ou au stade de prototype.

Cette étude met également en évidence un positionnement imprécis par rapport aux approches concurrentes. Un des freins actuels à une utilisation plus massive du TPM vient du fait que les solutions développées rendent un service déjà proposé par des applications concurrentes. Or, tout le monde sait qu'il est difficile de changer les habitudes des utilisateurs et des développeurs.

Mais il est avant tout nécessaire d'obtenir une meilleure couverture des besoins du marché. En particulier, le besoin de vérification d'intégrité d'une plateforme pendant l'exécution doit être renforcé. Un tel mécanisme permettrait alors de fournir une souche de confiance aux suites de sécurité logicielle pour accroitre la sécurité d'un poste client. Il va de même pour le besoin en confidentialité d'exécution, plus adapté aux contextes de défense et aux environnements partagés (serveurs avec machines virtuelles).

La relative atonie du marché de l'informatique de confiance impose une remise en cause, qui doit aboutir à une clarification du positionnement du composant TPM vis à vis des technologies alternatives. Historiquement, la conception du TPM a été guidée par le choix d'un compromis entre le faible cout unitaire du composant et son caractère matériel qui renforce nativement la sécurité de l'ar-

chitecture du poste client. Les nouveaux besoins exhibés dans cette article nécessiteraient une adaptation des spécifications du TPM ou l'utilisation de technologies complémentaires. A titre illustratif, dans le domaine du grand public et des entreprises, une architecture reposant uniquement sur un TPM sera certes à bas cout mais ne permettra pas de répondre à certains besoins liés à l'exécution sécurisée de code. Une telle architecture devrait reposer à la fois sur le TPM et sur des mécanismes de virtualisation. En revanche, dans le domaine de la défense, où le cout des composants cryptographiques n'est pas forcément un critère prioritaire, il est préférable de s'orienter vers un composant de type co-processeur cryptographique ou processeur sécurisé pour mettre en œuvre ces services. Au final, quelque soit la cible, nous retrouvons un schéma identique opposant domaine de confiance et domaine non de confiance, que celui-ci soit réalisé logiciellement ou matériellement.

La prise en compte de nouveaux besoins nécessite probablement l'implication de nouveaux acteurs. En faisant l'analogie avec le processus de spécification de la puce TPM, une telle réorganisation a eu lieu lorsque le consortium TCPA s'est transformé en TCG en 2003. De nouveaux acteurs sont apparus et le positionnement du groupe de travail s'est ainsi modifié. En particulier, le mécanisme d'attestation distante correspondait, au moment de sa spécification, a un besoin fort de l'industrie pour renforcer la gestion des droits numériques (DRM). Ce besoin s'est peu à peu estompé au profit de nouveaux tels que la protection matérielle des données – avec le chiffrement disque natif – et le lancement sécurisé d'environnement d'exécution – technologie Intel TXT.

Références

1. ANSSI. *Règles et recommandations concernant le choix et le dimensionnement des mécanismes cryptographiques de niveau de robustesse standard*. 2009. http://www.ssi.gouv.fr/site_documents/politiqueproduit/Mecanismes_cryptographique_v1_10_standard.pdf.
2. Liqun Chen and Mark Ryan. Attack, solution and verification for shared authorisation data in tcg tpm. In *FAST*, 2009. http://www.cs.bham.ac.uk/~mdr/research/papers/pdf/09-FAST.pdf.
3. C. Crépeau and A. Slakmon. Simple backdoors to rsa key generation. In *3rd RSA Cryptographer track conference*, 2003.

4. Christophe Devine and Guillaume Vissian. Compromission physique par le bus pci. In *SSTIC 2009*, 2009.
5. Maximillian Dornseif. 0wned by an ipod. In *PacSec*, 2004. http://md.hudora.de/presentations/firewire/PacSec2004.pdf.
6. Guillaume Duc and Ronan Keryell. Un panorama des architectures informatiques sécurisées et de confiance. In *C&ESAR*, 2008. http://enstb.org/~keryell/publications/conf/2008/CESAR/article.pdf.
7. Loïc Duflot and Olivier Levillain. Acpi et routine de traitement de la smi : des limites à l'informatique de confiance ? In *SSTIC*, 2009. http://www.sstic.org/media/SSTIC2009/SSTIC-actes/ACPI_et_routine_de_traitement_de_la_SMI_des_limite/SSTIC2009-Article-ACPI_et_routine_de_traitement_de_la_SMI_des_limites_a_l_informatique_de_confiance-duflot.pdf.
8. Jason Franklin, Arvind Seshadri, Ning Qu, Sagar Chaki, and Anupam Datta. Attacking, repairing and verifying secvisor : A retrospective on the security of a hypervisor. Technical report, 2008. http://www.cylab.cmu.edu/files/pdfs/tech_reports/cmucylab08008.pdf.
9. Google. Verified boot – the chromium projects. Technical report, 2009. http://sites.google.com/a/chromium.org/dev/chromium-os/chromiumos-design-docs/verified-boot.
10. Trusted Computing Group. Common criteria protection profile – pc client specific trusted platform module family 1.2 ; level 2. Technical report, 2008. http://www.trustedcomputinggroup.org/files/resource_files/B5F98F50-1D09-3519-ADBB86FAF51D5490/pp0030a.pdf.
11. Trusted Computing Group. Design and implementation of a tcg-based integrity measurement architecture. Technical report, 2009. http://www.trustedcomputinggroup.org/resources/summary_of_features_under_consideration_for_the_next_generation_of_tpm.
12. J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember : Cold boot attacks on encryption keys. In *USENIX Security Symposium*, 2008. http://citp.princeton.edu/pub/coldboot.pdf.
13. IBM. Integrity measurement architecture. Technical report, 2008. http://domino.research.ibm.com/comm/research_people.nsf/pages/sailer.ima.html.
14. IBM. Trusted grub. Technical report, 2009. trousers.sourceforge.net/grub.html.
15. IBM. Tss 1.2 features being implemented in trousers. Technical report, 2009. http://trousers.sourceforge.net/trousers12support.html.
16. Intel. Intel® trusted execution technology. Technical report, 2008. http://download.intel.com/technology/security/downloads/315168.pdf.
17. Intel. Intel® virtualization technology for directed i/o. Technical report, 2008. http://download.intel.com/technology/computing/vptech/Intel(r)_VT_for_Direct_IO.pdf.
18. Intel. Trusted boot. Technical report, 2009. http://sourceforge.net/projects/tboot/.
19. ISO. Iso/iec 11889-1 :2009 – trusted platform module – part 1 : Overview. Technical report, 2009. http://www.iso.org/iso/catalogue_detail.htm?csnumber=50970.

20. Bernhard Kauer. Oslo : Improving the security of trusted computing. In *USENIX Security Symposium*, 2007. http://os.inf.tu-dresden.de/papers_ps/kauer07-oslo.pdf.
21. Eric Lacombe, Vincent Nicomette, and Yves Deswarte. Une approche de virtualisation assistée par le matériel pour protéger l'espace noyau d'actions malveillantes. Technical report, 2009. http://www.sstic.org/media/SSTIC2009/SSTIC-actes/Une_approche_de_virtualisation_assistee_par_le_mat/SSTIC2009-Article-Une_approche_de_virtualisation_assistee_par_le_materiel_pour_proteger_l_espace_noyau-lacombe.pdf.
22. Mandriva. Sboot technical description. Technical report, 2008. http://sboot.linbox.org/wiki/Documentation.
23. Jonathan M. McCune, Ning Qu, Yanlin Li, Anupam Datta, Virgil D. Gligor, and Adrian Perrig. Efficient tcb reduction and attestation. Technical report, 2009. http://www.cylab.cmu.edu/files/pdfs/tech_reports/CMUCylab09003.pdf.
24. Joanna Rutkowska and Alexander Tereshkin. Evil maid goes after truecrypt ! Technical report, Invisible Things Lab, 2009. http://theinvisiblethings.blogspot.com/2009/10/evil-maid-goes-after-truecrypt.html.
25. Ravi Sahita, Ulhas Warrier, and Prashant Dewan. Protecting critical applications on mobile platforms. Technical report, 2009. http://www.intel.com/technology/itj/2009/v13i2/ITJ9.2.2-Launch.htm.
26. Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn. Design and implementation of a tcg-based integrity measurement architecture. In *USENIX Security Symposium*, 2004. http://www.usenix.org/events/sec04/tech/full_papers/sailer/sailer.pdf.
27. Arvind Seshadri, Mark Luk, Ning Qu, and Adrian Perrig. Secvisor : A tiny hypervisor to provide lifetime kernel code integrity for commodity oses. In *SOSP*, 2007. http://www.cs.cmu.edu/~arvinds/pubs/secvisor.pdf.
28. Rafal Wojtczuk and Joanna Rutkowska. Attacking intel® trusted execution technology. In *Black Hat DC*, 2009. http://invisiblethingslab.com/resources/bh09dc/Attacking%20Intel%20TXT%20-%20paper.pdf.