RenPar'20 / SympA'14 / CFSE 8
Saint-Malo, France, du 10 au 13 mai 2011

CertiCloud : une plate-forme Cloud IaaS sécurisée

Benoît Bertholon[1], Sébastien Varrette[2] and Pascal Bouvry[2]
[1] Interdisciplinary Centre for Security Reliability and Trust
[2] Computer Science and Communication (CSC) Research Unit
Université du Luxembourg 16, rue Richard Coudenhove-Kalergi, L-1359 Luxembourg (Luxembourg)
Emails : {Firstname.Name@uni.lu}

Résumé

La sécurité des Clouds est un aspect essentiel qui n'est pas forcément abordé selon le point de vue de l'utilisateur. En particulier, sur une plate-forme de type Infrastructure-as-a-Service (IaaS), il est actuellement impossible pour un utilisateur de certifier de manière fiable et sécurisée que l'environnement qu'il a déployé (typiquement sous forme d'une machine virtuelle) est toujours dans un état qu'il juge intègre et opérationnel. Cet article s'attelle à cette tâche en proposant CertiCloud, une plate-forme Cloud de type IaaS qui exploite les concepts développés dans le cadre du Trusted Computing Group (TCG) mais aussi les éléments matériels que sont les Trusted Platform Module (TPM) pour offrir à l'utilisateur un environnement sécurisé et sécurisant. Ces deux aspects sont garantis par les deux protocoles `TCRR` (TPM-based Certification of a Remote Resource) et `VerifyMyVM` qui sont à la base de CertiCloud. Quand le premier permet de certifier l'intégrité d'une machine distante et d'échanger une clef de chiffrement symétrique, le second permet à l'utilisateur de s'assurer dynamiquement et à la demande de l'intégrité de sa machine virtuelle exécutée sur les ressources de CertiCloud. Ces deux protocoles étant les briques de base de notre plate-forme, une attention toute particulière a été apportée à leurs élaborations. À cet effet, ils ont été validés avec succès par AVISPA [1] et Scyther [9], deux outils de référence dans le domaine de la vérification automatique des protocoles de sécurité (cette analyse est présentée dans cet article). Ensuite, la plate-forme CertiCloud est détaillée : outre les protocoles `TCRR` et `VerifyMyVM`, elle propose le stockage sécurisé des environnements utilisateurs et leurs exécutions à travers un framework de virtualisation reprenant l'hyperviseur Xen. Quand les ressources physiques sont certifiées par `TCRR`, l'utilisateur peut utiliser à la demande le protocole `VerifyMyVM` pour s'assurer de l'intégrité de son environnement déployé. Un prototype de CertiCloud a été réalisé et nous présentons les premiers résultats expérimentaux qui démontrent de la faisabilité et du faible surcoût de notre approche sur des scénarios classiquement rencontrés sur les infrastructures Cloud de type IaaS.

1. Introduction

Le Cloud Computing (CC) conceptualise un service de calcul dans lequel l'utilisateur ne possède aucune partie de l'infrastructure : il ne paie que pour le temps de calcul utilisé. Ce concept reprend celui des grilles de calculs [12] – largement étudiées dans le monde académique depuis de nombreuses années sans toutefois susciter un intérêt suffisant des principaux acteurs industriels et privés. À l'inverse des grilles, le paradigme du Cloud est rapidement devenu un *leitmotiv*, à la base de la plupart des *business model* qui touchent de près ou de loin à l'informatique. Les raisons de ce succès sont multiples (et dépassent le cadre de cet article) mais la principale est liée au fait que ce concept rationalise et limite les coûts liés au calcul ou au stockage des données à une époque où ces éléments sont devenus stratégiques tout en imposant des investissements lourds pour qui veut une infrastructure opérationnelle à la pointe de la technologie. Dans la terminologie CC on distingue trois types de services de CC [26] :

1. Software-as-a-Service (SaaS). Ce service concerne l'utilisation de logiciels qui s'exécutent sur le Cloud. Le fournisseur de service possède les logiciels, de même que les licences de sorte que l'utilisateur n'est pas obligé de s'acquitter du prix de la licence pour utiliser le logiciel.
2. Platform-as-a-Service (PaaS), où l'utilisateur a la possibilité d'exécuter son programme sur les ressources du Cloud, en utilisant les langages et outils supportés par le fournisseur de services.
3. Infrastructure-as-a-Service (IaaS), qui autorise le déploiement et l'exécution d'un environnement entièrement contrôlé par l'utilisateur (une machine virtuelle en général) sur les ressources du Cloud.

Nous nous intéressons ici à la sécurité des Clouds de type IaaS. Si cette thématique reste prépondérante dans la littérature relative au CC, on trouve peu de travaux qui l'abordent en adoptant le point de vue de

l'utilisateur au sens suivant : en tant qu'utilisateur d'une plate-forme Cloud de type IaaS, (1) comment puis-je être assuré que mon environnement reste confidentiel une fois enregistré sur les ressources de stockage de masse offerts par mon fournisseur de services ? (2) Ai-je un moyen fiable et sécurisé de vérifier que les ressources du Cloud ne sont pas corrompues ? (3) Une fois mon environnement déployé, puis-je certifier son intégrité (si possible à la demande) *i.e.* détecter s'il a subi une modification non désirée telle l'installation d'un *rootkit* ? Les contributions de cet article traitent ces questions en proposant :

- le protocole `TCRR` (TPM-based Certification of a Remote Resource) qui exploite un élément matériel appelé Trusted Platform Module (TPM) émanant d'une spécification internationale élaborée par les principaux acteurs de l'informatique moderne (HP, IBM, Intel, Microsoft, AMD etc.). `TCRR` a pour objectif de certifier l'intégrité d'une machine distante (disposant d'un TPM) et d'échanger avec elle une clef de chiffrement symétrique (contrôlée par l'utilisateur) qui pourra être employée à diverses fins cryptographiques comme le chiffrement de données ou de communications réseaux.
- CertiCloud, une plate-forme Cloud de type IaaS qui offre des mécanismes permettant de répondre aux trois questions ci-dessous. CertiCloud se base évidemment sur le protocole `TCRR` mais aussi sur l'installation d'un framework de virtualisation reprenant l'hyperviseur Xen sur les ressources du Cloud. Dans ce contexte, les environnements utilisateurs (qui prennent la forme d'une Virtual Machine (VM) Xen) sont déployés et associés au démarrage à un TPM logiciel (ce choix sera discuté au §5). Le protocole `VerifyMyVM` est alors proposé (impliquant notamment le TPM logiciel) pour permettre à l'utilisateur de vérifier à la demande l'intégrité de sa VM.

Les protocoles `TCRR` et `VerifyMyVM` ainsi que l'architecture CertiCloud ont été développés et imaginés dans le cadre de cet article.

Beaucoup trop de travaux introduisent des protocoles de communication et valident leur sécurité sur présentation de quelques scénarios déclarés sûrs. L'existence d'outils de vérification automatique de protocoles permet en revanche d'augmenter les capacités de détection de problèmes de sécurité ne serait-ce que par la complexité et le nombre de scénarios d'attaques envisagés. Les outils de référence dans ce domaine ce nomment AVISPA [1], Scyther [9], Proverif [7] ou encore Casper/FDR [15]. Nous verrons que les protocoles proposés dans ce document ont été analysés avec succès par AVISPA et Scyther. Cela assure qu'aucune attaque envisagée par ces outils n'est possible contre `TCRR` et `VerifyMyVM`.

Cet article est organisé de la manière suivante. Après un bref rappel de l'état de l'art au §2, la section 3 présente brièvement les TPM et introduit les hypothèses sur les acteurs impliqués dans CertiCloud. Le protocole `TCRR` ainsi que sa validation est ensuite présenté (au §4). Fort de cette première contribution, la section 5 décrit CertiCloud, notre proposition de plate-forme IaaS offrant à l'utilisateur un environnement sécurisé et sécurisant. Le protocole `VerifyMyVM` est exposé à cette occasion, de même que le résultat de son analyse par AVISPA et Scyther. Un premier prototype de CertiCloud a été implémenté et les résultats expérimentaux sont particulièrement encourageant, sont présentés au §6. Enfin, la section 7 conclut cet article tout en offrant quelques perspectives.

2. État de l'art

Jusqu'à la généralisation des TPMs, vérifier l'intégrité d'un système était limité à des solutions logicielles. En pratique, on vérifie généralement que le système qui s'exécute n'a pas été modifié, en utilisant par exemple des méthodes de détection de malwares [8] ou des techniques basées sur de la vérification de sommes de contrôle. Ce dernier concept, utilisé notamment dans Tripwire [14], évalue périodiquement les checksums sur les fichiers et dossiers les plus importants du système afin de les comparer avec des valeurs de références. Nous verrons que CertiCloud reprend ce concept, adapté toutefois aux fonctionnalités offertes par la présence des TPMs. Il existe également d'autres techniques, décrites en particulier dans [6], pour s'assurer que le flot de contrôle d'un programme exécuté est correct. Cela est typiquement réalisé à l'aide de mécanismes de protection de la pile, ou en chiffrant les pointeurs de fonction de retour. Néanmoins, ces solutions ne s'appliquent pas à un environnement distribué et ne sont pas fiables contre un attaquant qui aurait un accès privilégié (*i.e.* `root`) au système.

Plus généralement, les solutions purement logicielles ne permettent pas de faire entièrement confiance à un système distant, il est donc indispensable de se munir de composant matériels tels que les TPMs. Ce terme désigne à la fois une spécification de crypto-processeur et son implémentation sous forme d'un chipset disponible de plus en plus couramment sur les cartes mères récentes. Nous y reviendrons plus

longuement au §3.1. L'utilisation des TPMs tend à se développer, par exemple Microsoft Bitlocker [17] y a recourt pour effectuer un chiffrement total de disques durs, notamment pour les ordinateurs portables. Ce composant est aussi émulé dans l'hyperviseur Xen [5] pour permettre à une VM de l'utiliser. Dans tous les cas, les TPMs offrent des perspectives inédites quand à la vérification de la sécurité d'un système (local ou distant). L'idée d'utiliser ses capacités pour sécuriser les services de Cloud n'est pas nouvelle et a été évoquée par le TCG dans [3], sans pour autant fournir beaucoup de détails. Également utilisé dans [10], le TPM et sa version émulée logiciellement ont été utilisés à travers Xen pour sécuriser les différentes VMs exécutées et les isoler les unes des autres. Néanmoins, cette utilisation se limite à une utilisation locale. Dans un contexte plus proche de celui abordé dans cet article, une infrastructure appelée Trusted Cloud Computing Platform (TCCP) a été proposée dans [20]. Similaire à Tera [23], elle permet au fournisseur de services IaaS de garantir une exécution confidentielle de la VM d'un l'utilisateur. Même si ces travaux n'ont pas les mêmes objectifs que CertiCloud, ils illustrent bien la volonté d'exploiter les TPMs et ses capacités de certification à distance. Cependant, comme la plupart des propositions dans ce domaine, les protocoles décrits dans les études évoquées précédemment ne sont ni validés ni analysés, de sorte qu'il est délicat d'y apporter un quelconque crédit en terme de sécurité. À l'inverse, notre approche est similaire à celle utilisée par Munoz & al. dans [18] qui se base sur les TPMs dans un contexte multi-agents. le protocole associé a été validé par un outil de vérification automatique (AVISPA). Le protocole TCRR présenté dans cet article étend celui développé dans SecMilia et l'adapte au contexte Cloud.

3. Contexte et hypothèses

3.1. Trusted Platform Module (TPM)

Un TPM [13] peut être vu comme un crypto-processeur autonome présent sur la plupart des cartes mères vendue à l'heure actuelle. Ses spécifications [25] émanent du Trusted Computing Group (TCG), un consortium impliquant les principaux acteurs de l'informatique moderne (HP, IBM, Intel, Microsoft, AMD etc.) de sorte que les TPMs ont pour vocation de devenir le standard intégré *de facto* dans tous les ordinateurs du futur. Le concept moteur élaboré par ce consortium sous la dénomination Trusted Computing (TC) est l'établissement d'une chaîne de confiance dans les couches logicielles s'exécutant sur une machine : une exécution n'est autorisée que si les logiciels antérieurs (BIOS, OS etc.) ont été certifiés. En pratique, une puce TPM est donc un coprocesseur cryptographique composé de deux types de mémoire : l'une permanente, l'autre volatile. La puce est physiquement protégée par un bouclier résistant aux interférences et dispose de plusieurs mécanismes de protection envers les attaques physiques similaires à ce qui se fait dans le domaine des cartes à puce. Le *coprocesseur* permet de réaliser de façon autonome un certain nombre de primitives cryptographiques (chiffrement, déchiffrement, hachage, signature etc.). Par exemple, la spécification TPM impose l'implémentation de l'algorithme RSA (avec une taille de clef de 2048 bits) ou encore de la fonction de hachage SHA-1. La *mémoire volatile* est principalement utilisée pour le stockage de sommes de contrôle dans des registres spéciaux appelés Platform Configuration Register (PCR). Il y en a au moins 16, chacun servant à sauvegarder la mesure d'intégrité d'un élément logiciel particulier (BIOS, MBR, *boot-loader*, OS, etc...) reflétant ainsi l'état du système à un instant donné. La mémoire *non-volatile* est utilisée pour stocker de manière persistante plusieurs données sensibles, principalement des paires de clefs : Endorsement Key (EK) (générée à la fabrication de la puce), Storage Root Key (SRK) et Attestation Identity Key (AIK) (générées à la prise de possession de la TPM – étape intervenant à la livraison de la ressource de calcul équipée de la TPM). Parmi les fonctions incluses dans le TPM et détaillées dans [24], les protocoles définis dans cet article utilisent les procédures suivantes :

- `TPM_UnBind(msg,'Key')` : déchiffre un message `msg` chiffré avec la clef publique `Key.pub` ;
- `TPM_Extend(valeur,i)` : modifie la valeur du registre `PCR[i]` à l'aide d'une fonction de hachage selon le pseudo-code suivant : $\mathrm{PCR}_{\mathtt{i}}^{new} = \mathrm{H}(\mathrm{PCR}_{\mathtt{i}}^{old} \| \mathtt{valeur})$ et $\mathrm{PCR}_{\mathtt{i}}^{init} = 0$
- `TPM_Quote('AIK',nonce,i1,...,ik)` : renvoie les valeurs des registres PCRs aux index $\mathtt{i}_1,\ldots,\mathtt{i}_k$, signées avec la clef `AIK` *i.e.* $\mathrm{SIGN}_{\mathrm{AIK}}(\mathrm{H}(\mathrm{PCR}_{\mathtt{i}_1},\ldots,\mathrm{PCR}_{\mathtt{i}_k}),\mathtt{nonce})$. Cela permet de récupérer la configuration d'un TPM authentifié par sa clef `AIK`.

3.2. Modélisation de la plate-forme IaaS et des différents acteurs

Nos travaux se placent dans le contexte d'un Cloud de type IaaS où interviennent les acteurs suivants :

- **le fournisseur de services Cloud**, qui gère la plate-forme IaaS accessible à travers une frontale

d'accès et dont les ressources physiques sont configurées pour déployer des Virtual Machine (VM) ;
– **l'utilisateur** qui souhaite déployer son environnement ;
– l'**autorité de certification (Certificate Authority (CA))** à qui se fient à la fois l'utilisateur et le fournisseur de service. Cette entité est le tiers de confiance qui garantit les certificats électroniques (et donc les clefs publiques) intervenant dans les communications entre l'utilisateur et le Cloud.

Les hypothèses suivantes sont supposées dans la suite :
– (H1) Chaque ressource physique du Cloud dispose d'un TPM "*possédé*" (au sens décrit précédemment) par le fournisseur de services ;
– (H2) Les manipulations physiques mais légitimes du TPM (telle que la remise à zéro des PCRs), ne sont pas protégées dans nos protocoles. Il est donc supposé que le fournisseur de services est le seul habilité à avoir un accès physique aux ressources du Cloud (et donc aux TPMs associés) ;
– (H3) L'utilisateur fait confiance au fournisseur pour l'installation, la maintenance et la protection des ressources physiques du Cloud.

On peut noter aussi qu'aucune hypothèse ne concerne la sécurité de la frontale d'accès car les protocoles développés dans cet article ont été définis en supposant que le médium de communication entre l'utilisateur et une ressource (dont la frontale est un élément) n'est pas sûr et sujet à des attaques de type *man-in-the-middle*.

3.3. Analyse automatique des protocoles de sécurité

Le développement de protocoles de communications est à la base du travail présenté dans ce papier, et ce type de protocole est sujet à des erreurs difficilement détectables du fait des combinaisons possibles dans l'ordre des messages et dans le nombre de sessions simultanées possibles. S'il fut un temps où seule l'analyse "manuelle" et le génie humain permettaient de détecter les failles de conception dans un protocole ou de prouver la sécurité d'un protocole, l'expérience montre que ce n'est plus considéré comme suffisant aujourd'hui. L'exemple le plus frappant dans ce cadre est celui du protocole Needham-Schroeder [19]. Alors qu'il avait été *prouvé* sûr à sa conception, il a été victime d'une attaque découverte par Lowe [16] en utilisant un outil de vérification automatique (Casper/FDR[15]), et cela à cause d'une mauvaise modélisation de l'attaquant dans la preuve initiale. Pour résumer, il est **impératif** actuellement de valider ses protocoles de sécurité à travers un outil de vérification automatique. La description de ces outils et de leurs contextes d'action dépasse le cadre de cet article. Néanmoins, nous avons tenu à valider nos protocoles par les logiciels de vérification de protocole les plus utilisés dans le monde industriel : AVISPA [1] et Scyther [9].

Note : par souci de place, nous n'avons pas mis dans cet article les listings utilisés pour la modélisation et la validation des protocoles proposés. Ils sont évidemment à la disposition du lecteur sur simple demande.

4. Le protocole `TCRR` (TPM-based Certification of a Remote Resource)

L'un des objectifs de ces travaux est de certifier à l'aide d'un protocole de communication réseau et de composants matériels qu'une machine distante est intègre, c'est à dire que les programmes s'exécutant depuis son démarrage (BIOS, MBR, Boot loader, Kernel, OS ...) n'ont pas été corrompus. Le protocole `TCRR` (TPM-based Certification of a Remote Resource) décrit dans cette section rempli cette fonction mais il permet également à l'utilisateur et à la machine distante certifiée de s'échanger une clef de chiffrement symétrique. Ce protocole met en scène trois entités, l'*utilisateur* U, la *machine distante* M et son TPM. Les messages échangés dans `TCRR` sont détaillés dans la figure 1. Nous supposons (1) que les acteurs possèdent déjà les certificats des autres parties impliquées dans le protocole, signés par le CA ; (2) que l'utilisateur dispose des valeurs des PCRs dites "de référence" *i.e.* correspondant à un état de la machine distance jugé normal. Les deux phases du protocole `TCRR` sont décrites ci-dessous.

Vérification de l'intégrité de la machine de calcul (msg 1-4). L'utilisateur U envoie un nonce $\mathfrak{n}_1$ à M et l'identité de M le tout signé avec sa clef privée. La signature attestant de l'identité de l'utilisateur, la machine distante envoie au TPM le nonce $\mathfrak{n}_1$, un nouveau nonce $\mathfrak{n}_2$ qu'elle a généré ainsi que son identité. Ces trois éléments servent de challenge à la fonction `TPM_Quote` qui répond par les valeurs des PCRs signées avec sa clef AIK comme expliqué dans §3.1. Puis M transmet à U une version signée avec sa clef du message dernièrement reçu, de la valeur de $\mathfrak{n}_2$ et de son identité. À l'issue de cette phase, les propriétés suivantes sont acquises :

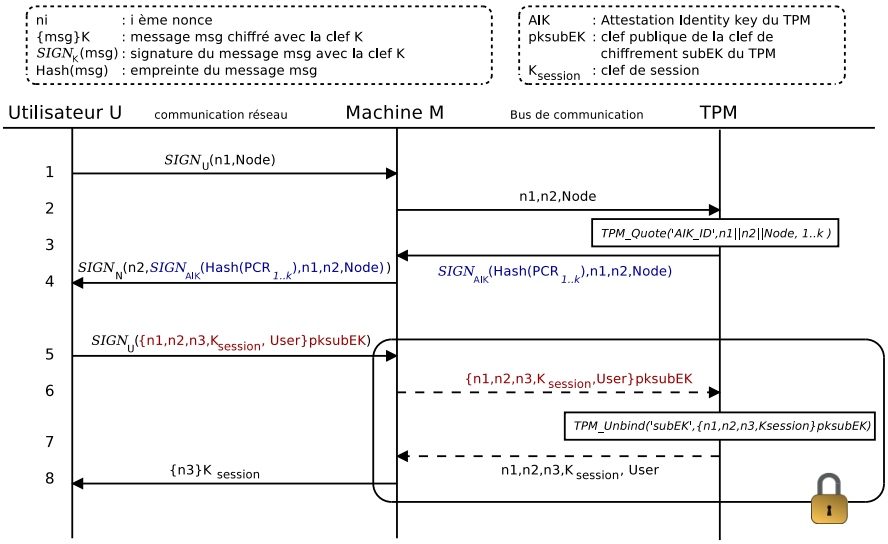

FIGURE 1 – Vue générale du protocole `TCRR` (TPM-based Certification of a Remote Resource).

– U peut vérifier la validité de toutes les signatures et des valeurs des nonces, ce qui authentifie M et son TPM associé. Cela garantit l'intégrité des messages échangés. U peut également être sûr que les valeurs des PCRs reflètent l'état de M car le TPM à été challengé avec un nonce unique généré par U.
– Les valeurs des PCRs reçues peuvent être comparées à celles de références.

À l'issue du 4^e message, M est dans un état sûr (validé par les valeurs PCRs), U peut donc faire confiance à M. Il est supposé qu'à partir de ce point, **la machine distante M et le TPM sont une seule et même entité**.

Échange de la clef de session (msg 5-8). U utilise la clef publique `subEK` du TPM pour chiffrer un message contenant les nonces échangés précédemment (n_1 et n_2), un nouveau nonce n_3 et une clef privée $K_{session}$. Ce message est d'abord signé avec la clef privée de U, puis envoyé à M. M vérifie la signature, utilise le TPM pour déchiffrer le message, et renvoie à U le nonce n_3 chiffré avec la clef de session.

Validation du protocole `TCRR`

Le protocole `TCRR` a été analysé et vérifié en utilisant à la fois AVISPA [1] et Scyther [9] (cf §3.3). Le modèle d'attaquant utilisé dans les deux outils est celui de Dolev-Yao [11] qui suppose que l'attaquant a accès à *tous* les messages qui sont transmis sur le réseau ; il peut donc les lire et les modifier. Dans ce contexte, nous avons sollicité la vérification des contraintes suivantes :
– le protocole fini de s'exécuter en même temps pour les acteurs sélectionnés, sans attaque du type 'rejeux' ou 'fabrication de messages'. Cela correspond aux objectifs '`authentication_on valid`' (resp. '`Niagree`' et '`Nisynch`') dans AVISPA (resp. Scyther).
– seuls les acteurs éligibles connaissent la clef $K_{session}$, qui doit rester confidentielle.

Les listings 1 et 2 montrent le résultat de ces vérifications. On voit qu'aucune attaque n'a été trouvée sur le protocole par les deux outils pour les objectifs spécifiés. Les codes de vérification sont disponibles sur le site de CERTICLOUD [2].

5. CertiCloud : une nouvelle infrastructure pour la sécurité des plates-formes IaaS

Basé sur le protocole `TCRR` décrit au §4, nous présentons dans cette section une nouvelle infrastructure de protection pour les plates-formes Cloud de type IaaS. Comme souvent dans ces cas là, le client déploie sa machine virtuelle sur les ressources du Cloud. Dans CERTICLOUD cette tâche est déléguée au *manageur de VM* du Cloud, dont le système d'exploitation se résume aux logiciels nécessaires à la virtualisation de systèmes (comme par exemple Xen [5]) et à un ensemble d'émulateurs de TPM. On considère que l'utilisateur U détient une image de sa machine virtuelle VMI_U qu'il souhaite exécuter de manière sécurisée sur les ressources du Cloud. U calcule alors les sommes de contrôle (checksums) $\{h_0^{ref}, \ldots, h_k^{ref}\}$ de cet

```
$> ./avispa verification_avispa.hlpsl --ofmc
% OFMC
% Version of 2006/02/13
SUMMARY
  SAFE
DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
PROTOCOL
  /home/benoit/git/tcrr/avispa/results_verif.if
GOAL
  as_specified
BACKEND
  OFMC
COMMENTS
STATISTICS
  parseTime: 0.00s
  searchTime: 7.93s
  visitedNodes: 6696 nodes
  depth: 17 plies
```

```
$> ./avispa verification_vm_avispa.hlpsl --ofmc
% OFMC
% Version of 2006/02/13
SUMMARY
  SAFE
DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
PROTOCOL
  /home/benoit/git/verifymyVM/avispa/results_verif.if
GOAL
  as_specified
BACKEND
  OFMC
COMMENTS
STATISTICS
  parseTime: 0.00s
  searchTime: 0.07s
  visitedNodes: 158 nodes
  depth: 6 plies
```

Listing 1 – Validation du protocole `TCRR` (à gauche) et `VerifyMyVM` (à droite) avec AVISPA.

```
$> time ./scyther.py --max-runs=20 --all-attacks
 verification_scyther.spdl
Verification results:
claim id [tpmp,u1], Niagree : No attacks.
claim id [tpmp,u2], Secret ksession : No attacks.
claim id [tpmp,u3], Nisynch : No attacks.
claim id [tpmp,n3], Nisynch : No attacks.
claim id [tpmp,n1], Niagree : No attacks.
claim id [tpmp,n2], Secret ksession : No attacks.

real    0m6.029s
user    0m5.950s
sys     0m0.060s
```

```
$ time ./scyther.py verification_vm.spdl
Verification results:
claim id [verifvm,u1], Niagree : No attacks.
claim id [verifvm,u3], Nisynch : No attacks.
claim id [verifvm,n3], Nisynch : No attacks.
claim id [verifvm,n1], Niagree : No attacks.



real    0m0.055s
user    0m0.034s
sys     0m0.019s
```

Listing 2 – Validation du protocole `TCRR` (à gauche) et `VerifyMyVM` (à droite) avec Scyther

environnement, chacune correspondant aux éléments mentionnés dans le tableau 1. Elles forment ce que CertiCloud considère comme le "certificat de $\mathcal{VMI}_U$". Pour déployer son système de manière sécurisée,

Nom	Description	Éléments haché
h_0^{ref}	VM kernel	`vmlinuz-x.x.x.xxx`
h_1^{ref}	Init. ramdisk	`initrd.img-x.x.x.xxx`
$h_2^{ref} \cdots h_8^{ref}$	Dossiers principaux du système de la VM	`/bin /etc /sbin /lib /usr/bin /usr/lib /usr/sbin`

Table 1 – Définition des checksums de référence utilisés dans CertiCloud.

les étapes suivantes sont proposées (voir figure 2) : (1) U génère une clef privée K_U qu'il utilise pour chiffrer $\mathcal{VMI}_U$; (2) l'image $\mathcal{VMI}_U$ est envoyée sur un serveur de stockage dans le Cloud, en utilisant un protocole de transmission tel que `scp` ou `sftp`. L'image est ainsi stockée chiffrée sur les ressources de stockage de masse du fournisseur de service ; (3) maintenant U veut déployer son image. Il vérifie le manageur de VM et lui transmet la clef de session K_U en utilisant le protocole `TCRR` ; (4) le manageur de VM récupère et déchiffre l'image $\mathcal{VMI}_U$ à l'aide de la clef K_U et la stocke dans la mémoire locale de la ressource physique appelée à déployer $\mathcal{VMI}_U$; (5) le manageur de VM exécute alors l'émulateur de TPM qui sera associé à la machine virtuelle de l'utilisateur. Il y a donc autant d'émulateurs que de machines virtuelles qui s'exécute sur la machine ; (6) le manageur de VM initialise les PCRs de l'émulateur de TPM qui se comportent comme un TPM physique, reflétant l'état original de la machine virtuelle ; (7) la machine virtuelle peut être lancée ; (8) à la demande, l'utilisateur peut effectuer une vérification de l'état de sa machine virtuelle en utilisant le protocole `VerifyMyVM` décrit dans §5.1.

Dans CertiCloud, chaque VM à déployer se voit donc associée un TPM émulé logiciellement [22]. Ce choix a été fait pour les raisons suivantes : (1) le développement d'une librairie cryptographique est coûteux en temps, et une nouvelle implémentation entraîne de nombreuse failles de sécurité. En utiliser une déjà implémentée et testée est donc plus sûr ; (2) le TPM physique de la machine ne peut être réutilisé car le nombre de registres PCRs est limité et servent pour la plupart à certifier la ressource physique ; Comme plusieurs machines virtuelles doivent pouvoir s'exécuter sur la machine distante, associer un TPM (certes émulé logiciellement) à chaque VM assure un traitement dédié et garantit le passage à l'échelle

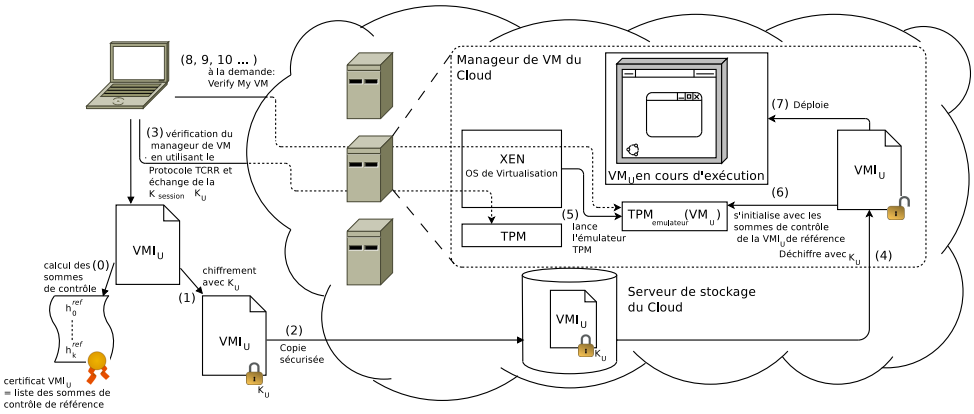

FIGURE 2 – Vue d'ensemble du framework CERTICLOUD.

quant au nombre de VM. Il est important de souligner que le code du TPM émulé est, au même titre que tous les logiciels s'exécutant sur les ressources du Cloud, certifié à travers le protocole `TCRR`. Évidemment, la VM et son TPM émulé sont deux processus dissociés contrôlés par le fournisseur de services de sorte que la corruption de la VM ne permet pas de mystifier le TPM.

5.1. Le protocole `VerifyMyVM`

Une fois l'intégrité du manageur de VM vérifié via le protocole `TCRR`, la VM déployée, un autre mécanisme est proposé pour permettre à l'utilisateur de détecter, dynamiquement et à la demande, toutes modifications sur la configuration ou sur les exécutables de sa VM. Le protocole `VerifyMyVM` reprend l'approche développée au sein du projet Tripwire[14], à savoir la comparaison des mesures d'intégrité du système à une base de référence éventuellement mise à jour. Ces haches ne sont évidemment pas calculés directement à l'intérieur de la machine virtuelle mais de façon externe par le manager de VM, hors de portée d'un attaquant ayant pris le contrôle de la VM.

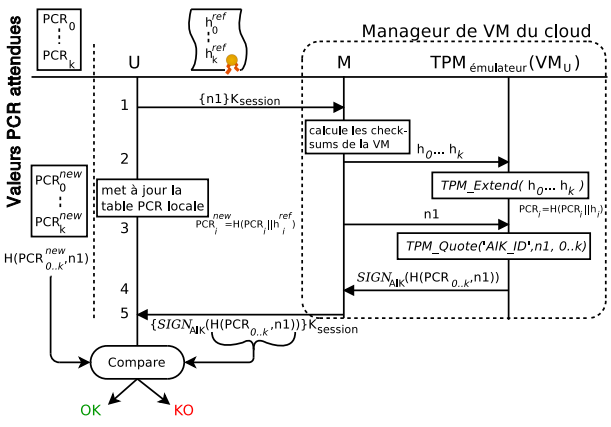

FIGURE 3 – Vue d'ensemble du protocole `VerifyMyVM`.

Le protocole `VerifyMyVM` est détaillé dans la figure 3 et est composé de 5 messages : (1) U demande au manageur de VM de vérifier sa machine virtuelle en exécution VM_U : il envoie un nonce n_1 chiffré avec la clef de session $K_{session}$ du protocole `TCRR`. M monte alors l'image de la machine virtuelle en lecteur seule et calcule les sommes de contrôle $\{h_0, \ldots, h_k\}$; (2) les sommes de contrôle sont utilisées par le $TPM_{emu}(VM_U)$ pour mettre à jour les valeurs de ses PCRS en calculant `TPM_Extend(`h_i`,i)`.En parallèle U met également à jour sa propre table de références PCRs de la façon suivante : $PCR_i = H(PCR_i \| h_i^{ref})$ (3) M challenge $TPM_{emu}(VM_U)$ avec le nonce n_1 ; (4) le $TPM_{emu}(VM_U)$ exécute un `TPM_quote` pour générer la signature des PCRs en fonction de n_1 avec la clef AIK ; (5) M transmet à U cette signature chiffrée avec la clef de session $K_{session}$. À l'aide de la clef de session $K_{session}$, l'utilisateur peut déchiffrer le dernier message, vérifier que la signature correspond bien au TPM et au nonce n_1, et comparer les valeurs des PCRs reçus avec celles évaluées localement. Si elles correspondent, le système est intègre. Comme pour `TCRR`, le protocole `VerifyMyVM` a été analysé et vérifié en utilisant à la fois AVISPA [1] et Scyther [9] (cf §3.3) contre le premier objectif mentionné au §4. Les listings 1 et 2 montrent le résultat de ces vérifications. On voit qu'aucune attaque n'a été trouvée sur le protocole pour les objectifs spécifiés.

6. Implémentation et validation expérimentale

Pour valider la faisabilité de cette infrastructure, un prototype de CertiCloud a été implémenté en utilisant comme manageur de VM la machine suivante :
– Processeur : Intel Pentium 4 (3 Ghz), 2 GB DDR RAM ; OS : Debian lenny, kernel 2.6.26-2-xen-686
– Virtualisation : Xen 3.2.1 ; ETHZ TPM émulateur (version 1.2.0.7 des spécifications)
– STMicroelectronics TPM (version 1.2.2.5 des specs) ; TPM/J application Java, API 0.3.0 (alpha) [21].
Nous avons considéré deux profils d'environnements classiquement déployés par les utilisateurs de services de types IaaS. Le premier de petite taille (336Mo), contient une distribution Debian Lenny configurée en mode LAMP (Linux/Apache/MySQL/PHP). Le second correspond plutôt à un environnement de bureau et contient une distribution Ubuntu Hardy de taille supérieure (2.3 Go). Le critère choisi pour l'évaluation des performances des protocoles est le temps des vérifications sur la plateforme mentionnée précédemment. Ces vérifications ont été effectuées au moins 100 fois pour avoir des résultats statistiquement significatifs. Les performances des protocoles `TCRR` et `VerifyMyVM` sont présentées dans la figure 4. À chaque fois, une distinction est faite entre le temps total nécessaire pour finir le protocole et les opérations les plus lentes qui le composent *i.e.* celles qui font intervenir une fonction TPM (rappelons que les capacités de calcul des TPMs restent largement limitées par rapport aux processeurs classiques).

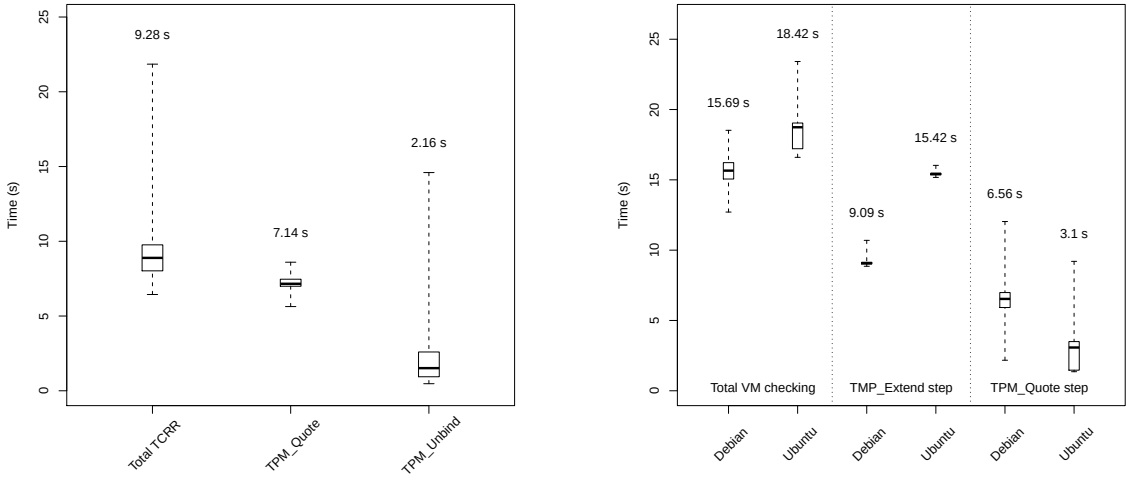

Figure 4 – Évaluation des performances des protocoles `TCRR` (g.) et `VerifyMyVM` (d.) dans CertiCloud

Type de VM :	Debian	Ubuntu
Taille (utilisée/totale)	336MB/2.1GB	2.3GB/21GB
Temps de déchiffrement	1m46.041s	16m12.503s
Temps de détection min	14.36989s	16.60140s
Temps de détection moyen	33.1792s	18.63032s
Temps de détection max	58.21868s	38.52942s

Figure 5 – Temps de déchiffrement moyen d'une image de VM et temps de détection après une modification non-autorisée avec CertiCloud.

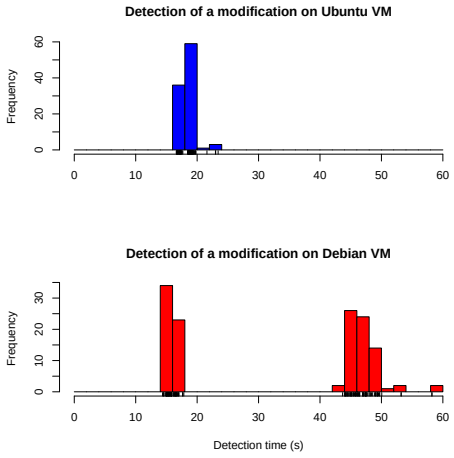

Figure 6 – Temps de détection après une modification non-autorisée avec CertiCloud : analyse en fréquence.

La mesure en temps de l'étape 4 du processus de CertiCloud, qui correspond au déchiffrement de l'image de la VM effectué par le manageur de VM, est représentée dans le tableau 5. Quand bien même cette étape est coûteuse, elle n'intervient finalement que lors du chargement de l'environnement de l'utilisateur et elle reste indispensable pour assurer la confidentialité de la machine virtuelle de l'utilisateur lorsque celle-ci est placée sur une zone de stockage de masse accessible à tous les utilisateurs du service Cloud. Le manageur de VM peut cependant effectuer ce déchiffrement, à la volée lors de la récupération du fichier sur le serveur de stockage, et étant donné que la vitesse de déchiffrement est plus grande que celle de transfert de fichier sur le réseau, le temps de déchiffrement n'implique donc quasiment aucune perte de temps.
La dernière expérience effectuée est le temps de détection d'une corruption (comme par exemple l'installation de rootkit) au temps t_0 en supposant que l'utilisateur demande des vérifications de sa machine virtuelle au manageur de VM au temps $t_0 + \epsilon$, On mesure alors le temps mis jusqu'à la découverte de la modification. En pratique cette modification est faite en altérant un fichier système à distance (en utilisant `scp`) avec le compte `root`. L'erreur insérée est forcement détecté par le protocole du fait de l'architecture et du protocole utilisé, en revanche le temps de détection varie en fonction de la taille du système ainsi que de sa politique d'écriture sur le disque. Ces temps de détection sont représentés dans les figures 6 et 5. On remarque que la politique d'écriture sur le disque favorise le système d'exploitation Ubuntu : bien que plus volumineux, le processus de détection sur cet environnement demeure en moyenne plus rapide. En effet, pour éviter des accès disques trop fréquents, Debian (dans sa configuration de base) n'écrit pas forcément directement les modifications sur le disque et garde celles-ci en cache. La détection est donc plus lente dans ce cas puisqu'elle est assujettie au vidage de ce cache.
Néanmoins, cette expérience montre que malgré le temps de déchiffrement de l'image de la VM, le coût induit par les protocoles est relativement faible (~ 10s pour `TCRR`, et moins de 20s pour `VerifyMyVM`). La détection de modifications à l'aide de `VerifyMyVM` nécessite entre 20s et 60s ce qui est très raisonnable du point de vue utilisateur (surtout dans un contexte distribué tel que le Cloud), d'autant plus que ce protocole s'effectue à la volée sans nécessiter de mettre en pause la VM vérifiée pour être effectué.

7. Conclusion

En dépit de l'euphorie suscitée depuis l'émergence du concept de Cloud Computing (CC), de nombreuses problématiques de sécurité restent à résoudre et font l'objet de nombreux travaux. Dans ce contexte, cet article apporte une contribution aux problèmes d'intégrité (et dans une moindre mesure de confidentialité) sur les plates-formes Cloud de type IaaS au travers d'une nouvelle infrastructure appelée CertiCloud. Jusqu'à récemment, seule l'approche logicielle proposait de vérifier l'intégrité d'un système, de telle manière qu'il n'était pas possible de faire totalement confiance à une ressource distante. Avec l'avènement des spécifications émises par le Trusted Computing Group (TCG) concernant les TPMs ainsi que leurs intégration à bas coût dans la plupart des cartes mères récentes, on peut sérieusement envisager la certification fiable d'une ressource distante. En effet, ce coprocesseur cryptographique, protégé physiquement contre les intrusions, donne accès aux principales primitives cryptographiques et permet de vérifier l'intégrité de la machine associée. Dans cette optique, le protocole `TCRR` (TPM-based Certification of a Remote Resource) a été introduit. Il permet également d'échanger une clef de chiffrement symétrique entre l'utilisateur et la ressource distante, clef utilisable par la suite à diverses fins cryptographiques. CertiCloud repose sur `TCRR` et propose une approche progressive pour sécuriser les machines virtuelles des utilisateurs, depuis leurs créations sur la machine de l'utilisateur jusqu'à leurs déploiements sur les ressources du Cloud en passant par leurs stockages sécurisés sur le Cloud. CertiCloud impose aux ressources appelées à exécuter les VMs utilisateurs de disposer d'un TPM (physique) et d'un framework de virtualisation basé sur l'hyperviseur Xen, le tout certifié par `TCRR`. À chaque VM déployée est associée un TPM émulé logiciellement, lui aussi certifié par `TCRR`. Enfin, le protocole `VerifyMyVM` est proposé. Il exploite les éléments précédents pour permettre à l'utilisateur de vérifier dynamiquement et à la demande l'intégrité de sa VM *i.e.* détecter une modification non désirée du système telle que l'installation d'un *rootkit*. Les deux protocoles ont été validés avec succès par AVISPA [1] et Scyther [9], deux outils de référence dans le domaine de la vérification automatique des protocoles de sécurité. Par ailleurs, une validation expérimentale (impliquant notamment l'implémentation des protocoles de CertiCloud) a été réalisée. Les premiers résultats expérimentaux démontrent de la faisabilité et du faible surcoût de notre approche sur

des scénarios classiquement rencontrés sur les infrastructures Cloud de type IaaS.
Les perspectives envisagées à l'issue de ces travaux sont nombreuses. Cela inclut une analyse plus poussée du passage à l'échelle de CertiCloud, ainsi que l'intégration dans un middleware tel que OpenNebula [4].
Remerciement : les auteurs tiennent à remercier Ton Van Deursen quant à son aide dans l'utilisation des outils AVISPA et Scyther. Ce projet est financé par le Fond National de la Recherche (FNR) du Luxembourg.

Bibliographie

1. Avispa. `http://avispa-project.org/`.
2. CertiCloud. `http://certicloud.gforge.uni.lu`.
3. Cloud Computing and Security – A Natural Match. Technical report, Trusted Computing Group.
4. Opennebula. [online] see `http://www.opennebula.org/`.
5. Xen hypervisor, a powerful open source virtualization software. `http://www.xen.org/`.
6. Martín Abadi, Mihai Budiu, Úlfar Erlingsson, and Jay Ligatti. Control-flow integrity principles, implementations, and applications. *ACM Trans. Inf. Syst. Secur.*, 13(1) :1–40, 2009.
7. Bruno Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, pages 82–96, Cape Breton, Nova Scotia, Canada, June 2001. IEEE Computer Society.
8. M. Christodorescu, S. Jha, S. A. Seshia, D. Song, and R. E. Bryant. Semantics-aware malware detection. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy (Oakland 2005)*, pages 32–46, Oakland, CA, USA, May 2005.
9. C.J.F. Cremers. *Scyther, Semantics & Verification of Security Protocols.* PhD thesis, Eindhoven Uni.
10. Chris I. Dalton, David Plaquin, Wolfgang Weidner, Dirk Kuhlmann, Boris Balacheff, and Richard Brown. Trusted virtual platforms : a key enabler for converged client devices. *SIGOPS Oper. Syst. Rev.*, 43(1) :36–43, 2009.
11. Danny Dolev and Andrew C. Yao. On the security of public key protocols. In *22nd Annual Symposium on Foundations of Computer Science*, pages 350–357, 1981.
12. I. Foster and C. Kesselman. *The Grid : Blueprint for a new Computing Infrastructure.* Morgan Kaufman Publishers, 1998.
13. International Organization for Standardization, Geneva, Switzerland. *ISO/IEC 11889-1 : Information technology – Trusted Platform Module – Part 1 : Overview*, 2009.
14. Gene H. Kim and Eugene H. Spafford. The design and implementation of tripwire : a file system integrity checker. In *Proc. of the 2nd ACM Conference on Computer and communications security (CCS'94)*, pages 18–29. ACM, 1994.
15. G. Lowe. Casper : A compiler for the analysis of security protocols.
16. Gavin Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR. In *TACAS 96*, pages 147–166. Springer-Verlag, 1996.
17. Microsoft. BitLocker Drive Encryption. `http://technet.microsoft.com/en-us/library/cc732774.aspx`.
18. Antonio Muñoz, Antonio Maña, and Daniel Serrano. Protecting agents from malicious hosts using tpm. *IJCSA*, 6(5) :30–58, 2009.
19. Roger Needham and Michael Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12), December 1978.
20. Nuno Santos, Krishna P. Gummadi, and Rodrigo Rodrigues. Towards trusted cloud computing. In *Workshop on Hot Topic in Cloud Computing (HotCloud'09)*, June 2009.
21. Luis Sarmenta. TPM/J Java-based API for the Trusted Platform Module. `http://projects.csail.mit.edu/tc/tpmj/`.
22. Mario Strasser. Software-based TPM Emulator. `http://tpm-emulator.berlios.de`.
23. J. Chow M. Rosenblum T. Garfinkel, B. Pfaff and D. Boneh. Terra : A virtual machine-based platform for trusted computing. In *Proc. of SOSP'03*, 2003.
24. TCG. TPM Main Part 3 Commands. Technical report, TCG, 2006.
25. TCG. TCG Specification Architecture Overview – Revision 1.4. Technical report, TCG, 2007.
26. John Viega. Cloud computing and the common man. *Computer*, 42 :106–108, 2009.