Comparaison fonctionnelle et expérimentale d'une PKI sur mobile et de Kerberos sur mobile. Mise en oeuvre et comparaison rigoureuse des résultats expérimentaux. Automatisation du déploiement.

Jean-Philippe Blaise, Willian Jouot

Master SSIC, Metz University,
Ile du Saulcy, 57045 Metz, France
jeanphilippe.blaise@umail.univ-metz.fr
william.jouot@umail.univ-metz.fr

Résumé De nos jours, les téléphones mobiles sont partout, et la population de smartphone est en plein essor. Ces types d'appareils peuvent faire de plus en plus de choses, et ainsi égaler nos ordinateurs. Ainsi en leur fournissant une multitude de moyens de communication, nous devons nous poser la question de la sécurité des données, telle que nous la connaissons pour nos ordinateurs. Les smartphones, étant de plus en plus puissant, peuvent parfaitement gérer un ou plusieurs protocole d'authentification. De ce fait, nous allons développer une solution de PKI et une solution Kerberos pour smartphones utilisants le celebre Operating System de Google, Android. Dans cet article, nous allons comparer quelques solutions déjà existantes, puis proposer notre méthode.

Mots-clés : PKI, Kerberos, Android, mobile, mise en oeuvre, déploiement automatique

1 Problématique

A l'heure actuelle, les téléphones ainsi que les smartphones se démocratisent de plus en plus, et il y a donc de plus en plus de données qui transitent via ces appareils, que cela soit via le wifi, bluetooth, ou directement le réseau téléphonique. On commence ainsi à faire des achats directement depuis son téléphone, où encore envoyer des informations personnels, comme des mots de passe, ou des documents que l'on veut garder secret. Malheureusement, toutes ces données transitent plus ou moins en clair dans l'air, à la portée de n'importe qui. Il faut donc sécuriser les données. Quelles solutions existent ? Des systèmes comme PKI ont déjà fait leurs preuves sur nos ordinateurs, mais existe-t-il des solutions concrètes pour nos mobiles ?

2 Travaux existants dans la littérature scientifique

2.1 PKI pour sites marchands

Il existe de nombreux travaux dans le domaine, notamment pour la PKI. Le premier article étudié[1], est très intéressant. Datant de 2007, cet article se pose une excellente

question : comment sécuriser des données transitant d'un téléphone mobile à un site marchand ? Ainsi on peut trouver une solution assez intéressante, la création des clés d'authentification utilisées se fait en local, c'est à dire que le téléphone est capable de générer ses propres clés. C'est un bon point. Les auteurs utilisent la Crypto librairie[2] pour la signature de données transmises. Cependant cette solution connait quelques faiblesses, que nous détaillerons plus tard.

2.2 Système de paiement sécurisé

Le second article[3] nous propose un autre système de paiement sécurisé pour mobile via une PKI. Pour leurs tests, le centre d'enregistrement finlandais s'est chargé de leur fournir la PKI.

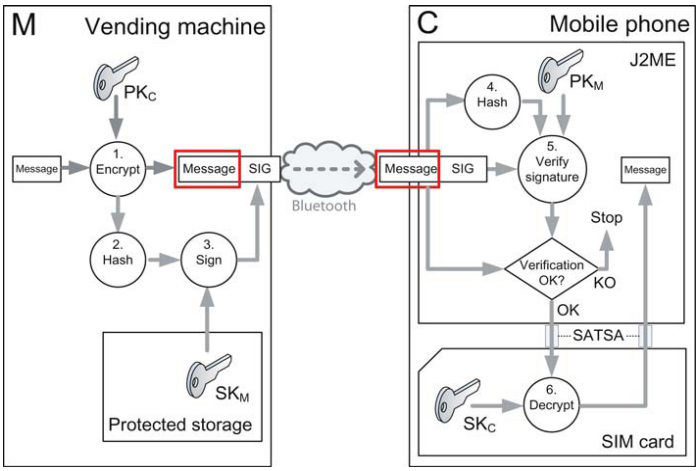

FIGURE 1. Exemple d'échange de messages securisés

Pour faire simple, les clés privées sont stockées dans la carte SIM. Etant infalsifiable, celles-ci ne peuvent donc pas être compromises. Chaque carte SIM se voit attribuer un certificat unique, délivré par le centre d'enregistrement finlandais. Cette carte SIM est capable de signer les données, mais le cryptage/décryptage se fait sur le téléphone. Ce système est déjà utilisé dans certaines cartes à puce où la dîte puce contient elle-même ses propres clés et est capable d'effectuer quelques opérations tel que signer des données. Une machine en ligne (banques, vente en ligne, etc.) crypte son message, et le signe. Le message est ensuite envoyé au téléphone mobile. Celui-ci vérifie la signature, et dans le cas où elle est correcte, le téléphone consulte sa clé privée, stockée dans la carte SIM, et décrypte le message.

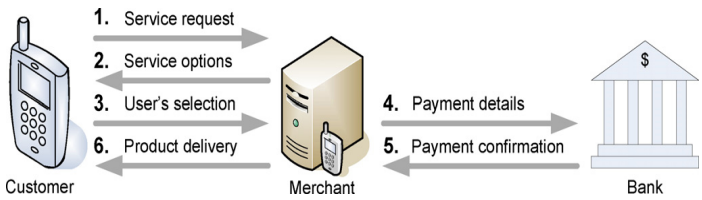

FIGURE 2. Modèle de paiement virtuel

Cette figure schématise le fonctionnement du paiement via téléphone mobile. 6 étapes sont nécessaires.

2.3 PKI pour (très vieux) mobiles

Le troisième article sur une PKI pour mobile [4]. Leur travail se reporte à d'anciens téléphones, avec très peu de mémoire vive et même un processeur très faible. Ainsi la génération de clés RSA leur est impossible. Ils décidèrent d'utiliser un autre algorithme de signature, plus adapté à leur type de téléphone, le ECDSA (Elliptic Curve Digital Signature Algorithm), avec des clés de 163 bits, équivalent à une clé RSA de 1024 bits, en terme de sécurité. De ce fait, les certificats ainsi générés sont eux aussi, plus petits.

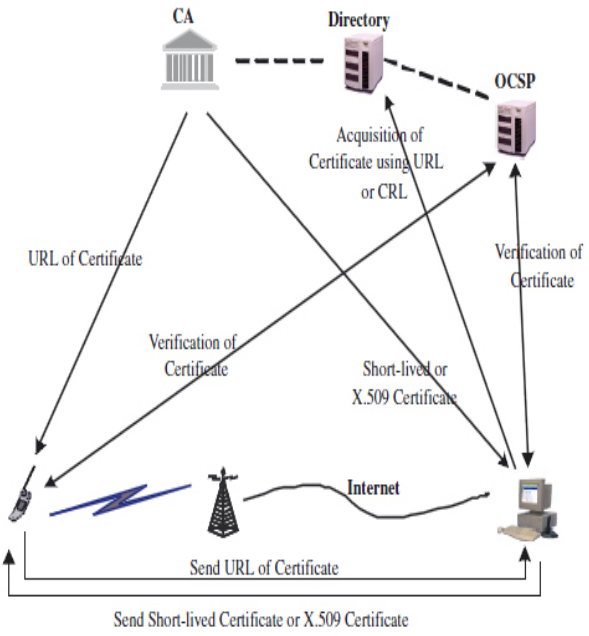

FIGURE 3. Schéma de leur système de PKI pour mobile

2.4 Kerberos et la non répudiation

Pour le premier article, offrant une solution Kerberos [5], met en avant la non-répudiation, en utilisant une méthode de hashage fonctionnant plus rapidement que les algorithmes de signatures numériques. Leur procédé fonctionne exactement comme le protocole Kerberos que nous utilisons sur nos ordinateurs. Le téléphone mobile demande à s'authentifier auprès du centre d'authentification. Puis après quelques opérations, on fournit au téléphone un ticket, lui permettant d'accéder au service demandé au tout debut. Cependant, il est possible de réutiliser un ticket pour accéder au même service, à plusieurs reprises. Au niveau de la sécurité, il n'est pas possible de créer un ticket, car la signature de l'authorité de certification n'est pas falsifiable. Un attaquant ne peux pas utiliser ou modifier un ticket émis. Les clés symétriques, utilisées lors de l'authentification sont regénérées à chaque session.

2.5 Kerberos et ses tickets transportables

Le second article offre lui une solution permettant à un téléphone ayant en sa possession un ticket d'itinérance, d'avoir accès aux services auquels il veut avoir accès, sans communiquer avec le serveur d'authentification à chaque fois.

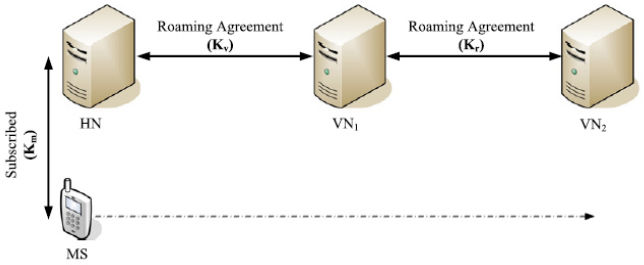

FIGURE 4. Représentation de l'itinérance

Pour ce faire, le téléphone s'authentifie auprès d'un réseau "visiteur", qui lui même authentifie le téléphone auprès de son home réseau.

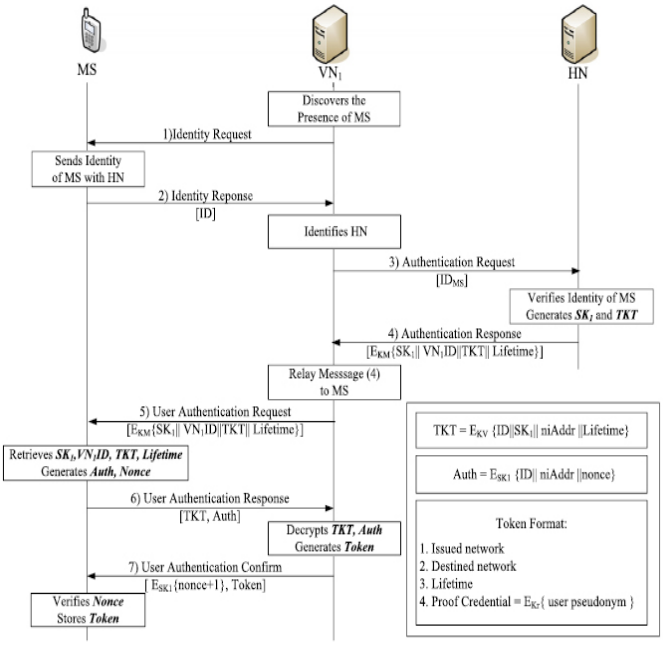

FIGURE 5. Authentification du téléphone auprès du réseau "visiteur"

3 Premières critiques des travaux existants

Premier constat, aucune solution de PKI ne supporte l'échange de données de téléphone à téléphone, seuls les échanges de données entre le téléphone et un serveur (banque, site de vente en ligne, etc.) sont gérés. D'un autre coté, les 3 solutions de PKI présentées, sont relativement complètes. Les systèmes de signatures sont réfléchis, les clés privées sont innaccessibles, même en cas de vol, puisque les clés sont stockées dans la carte SIM, qui est elle même protégée par un code PIN. De leur coté, les solutions Kerberos, présentées plus tôt ont eux aussi ce soucis de ne pas gérer les échanges entre deux téléphones.

Evidemment, chacunes des techniques PKI ou Kerberos présentent des désavantages qui sont propres à leur fonctionnement. Par exemple, lors d'un système Kerberos, les mots de passe utilisateurs sont enregistrés sur le serveur de manière non hashé, car le serveur a besoin du mot de passe en clair pour crypter la réponse à envoyer au client. De plus, Kerberos, de part son fonctionnement en serveur, peut poser problème si pour une raison ou une autre le serveur d'authentification n'est pas accessible. Dans le cas par exemple où le serveur a planté.

4 Objectifs et perspectives du projet de synthèse

Notre objectif était d'implémenter le déploiement automatique d'un système de certification pour mobile. Pour cela, nous devions implémenter une solution de PKI pour

mobile, et une solution Kerberos pour mobile. Ainsi, nous pouvons comparer de manière fonctionnelle les deux systèmes et choisir le plus efficace. Dans le cas de Kerberos, le but est de créer un petit client, capable de se connecter à un centre d'authentification Kerberos pour ensuite avoir un ticket pour un service demandé.

FIGURE 6. Nos début d'implementation

Nous avions commencé à implémenter une solution de PKI. Seules les méthodes de génération de clés, et de signatures étaient présentes. Le but de cette première implémentation était de pouvoir manipuler et vérifier le fonctionnement des fonctions de bases pour les clés publiques et privés sur des smartphones de type Android.

Coté Kerberos, nous avions réussi à créer un serveur Kerberos, basé uniquement sur des utilisateurs crées manuellement, et à créer un client Java capable de se connecter sur ce serveur pour récupérer, après quelques échanges, un ticket de service. Selon nos espérances, la prochaine étape était de, premièrement, convertir le projet pour qu'il fonctionne sur Android, et ensuite de créer un service, compatible Kerberos, comme par exemple une session SSH, pour que le téléphone puisse se connecter dessus.

Malheureusement, nous n'avons pas pu réaliser cette partie du projet. Nous détaillerons tout cela, un peu plus tard.

5 Critères de comparaison

Notre sujet était d'exporter deux méthodes d'authentification pour machines traditionnelles sur des téléphones mobiles et de les comparer.

Nous avons décidé de tester et comparer ses deux protocoles sur quelques aspects :

– Implémentation
– Rapidité d'execution
– Interaction avec l'humain
– Sécurité du protocole
– Solidité du protocole
– Scalabilité

6 Problèmes rencontrés

6.1 PKI

Nous n'avons pas rencontré de rééls problèmes pour le développement de notre client PKI. Les classes Java de bases et celle d'Android nous ont permis de réaliser cette partie sans soucis. La documentation ainsi que les diverses ressources sur Internet tel que les forums nous ont permis de surpasser chaque problème avec seulement un peu de patience. En véritable problème majeur, nous avons, environ à la moitié du developpement, dû passer d'une architecture ou nous gérions tous les messages en type string (chaînes de caractères), à une architecture qui utilise uniquement des tableaux de bytes, car c'est le type de variable qu'utilise les clés pour effectuer toutes les opérations. Sinon, des problèmes se posaient, notamment lors de la transmission entre le client et le serveur. Passer à des tableaux de bytes à faciliter grandement le problème.

6.2 Kerberos

Comme nous l'avions mentionné dans notre premier rapport, nous sommes tombé sur un problème de taille, pour Kerberos. La création d'un serveur Kerberos n'était pas une mince affaire. Nombreux sont les tutoriaux sur Internet, mais aucun ne disait la même chose. Ainsi nous avons eu quelques soucis pour tout mettre en place côté serveur. Un serveur Kerberos ne s'installe pas comme la plupart des logiciels serveurs tels que Apache ou SSH où il suffit simplement d'éditer un fichier de configuration. Kerberos est en fait une combinaison de plusieurs processus, qui peuvent être installé sur des machines différentes, et la configuration de cet ensemble est d'une compléxité remarquable, même sur une machine unique. Des notions et un vocabulaire spécifique est requis tel que la notion de "Royaume", qui désigne en fait le serveur qui contient les informations sur les utilisateurs. La gestion des utilisateurs est également assez complexe. Chaque service final (comme par exemple un serveur SSH fonctionnant avec Kerberos) doit également avoir un compte utilisateur sur le royaume en question. Le service est identifié par son nom, ainsi que par le nom du serveur sur lequel il fonctionne. Coté utilisateurs normaux, chaque utilisateur appartient à un groupe spécifique, groupe pouvant être défini avec des droits différents tel que la durée de vie des tickets ou de la robustesse des mots de passe.

Mais cela n'a pas été notre plus grand problème. Malheureusement, nous n'avons pas pu réalisé ce que nous avions prévu, à savoir convertir notre client Java classique pour un client Android. En effet, l'API Java inclue dans Android ne contient pas certaines classes qui sont obligatoires pour pouvoir gérer des identifications avec Kerberos. La grande majorité de nos fonctions ne fonctionnait plus.Nous avons tout d'abord

tenter de contourner le problème en recopiant simplement les classes manquantes à la main pour les réinclure dans notre projet. Malheureusement, certaines de ces classes, notamment les classes des packages org.ietf, n'ont pas de code source attaché et seul les version .class compilées sont présentes. La deuxième tentative a été de recréer un jar contenant toutes les classes manquantes que l'on a extraite des paquets de l'API Java. Nous avons donc inclus un fichier .jar à notre projet contenant toutes ses classes. Malheureusement, le compilateur Java détecte que l'on tente d'inclure des classes appartenant à des domaines java.* qui sont censées déjà être incluses, et refuse ainsi la compilation, même avec l'API Java d'Android.

Nous aurions pu contourner le problème en réécrivant totalement les classes Kerberos que nous devions utiliser, ce qui aurait été un travail considérable. Nous aurions également pû tenter d'écrire l'application en C++ grâce au SDK natif d'android, en incluant une librairie gérant Kerberos, mais cette solution aurait également demandé énormément de temps. Une dernière solution aurait été de décompiler les classes dont on ne pouvait pas obtenir le code source, pour pouvoir ensuite les réimplémenter dans notre projet comme nous avions tenter au début. Outre l'aspect légal qui est discutable, cela nous aurait également demandé un peu de temps, mais nous avons à ce stade préféré nous reconcentrer sur l'implémentation de PKI.

Enfin, nous avons remarqué que, contrairement à PKI, il y a très peu de documentation sur Internet pour implémenter Kerberos, notamment en Java où de tels exemples sont presque introuvable, et ne sont que très peu voir pas documentés.

7 Implémentation

7.1 PKI

Nous avons developper une application de PKI pour android, ainsi qu'un serveur Java sur lequel notre application se connecte. Notre application android est complète, c'est à dire qu'elle est capable de générer ses propres clés, les sauvegarder ou bien charger des clés deja stockées. Elle peut également envoyer sa clé publique à notre serveur. Elle peut récupérer la clé publique du serveur depuis un autre serveur tier. Et bien sur, elle peut signer et crypter un message. De son côté, le serveur écoute sur un port spécifique, et attend des connexions. Il peut ensuite réceptionner une clé publique, déchiffrer et vérifier la signature du message envoyé via notre application et le message en lui même. Le serveur peut également envoyer un message crypté avec la clé publique du téléphone, message que le téléphone déchiffera correctement pour retrouver le message d'origine.

De manière detaillée, notre serveur contient uniquement deux classes : Server.java et Client.java. Server.java est la classe principale et lance le serveur, charge sa clé publique et privée, et commence à écouter sur le port 1023. Elle contient également quelques fonctions permettant des opérations à base de tableaux de bytes sur des fichiers locaux. Lorsqu'un client se connecte, cela crée une instance de Client, qui est un sous-processus, et qui est spécifique pour ce client. Il écoute le message du client et y répond. Cette classe contient toutes les fonctions nécessaires pour crypter, décrypter, signer et vérifier la signature.

Notre application Android contient quelques classes. La classe principale gère l'interface, tel que le bouton et la zone de texte, ainsi que des fonctions d'accès au fichiers du téléphones, notamment pour sauvegarder les clés que l'on générera par la suite. L'application est dotée d'un menu qui permet de choisir l'action que l'on souhaite faire. Chaque action effectuée est chronométrée à la millisecondes et l'application affiche ainsi le temps nécessaire pour l'opération. On peut donc générer des clés, qui seront ici des clés 1024 bits RSA. Les clés générés seront automatiquement sauvegarder dans des fichiers locaux pour pouvoir ensuite les charger plus rapidement lors d'une utilisation ultérieure de l'application. On peut donc également charger les clés depuis le téléphone. Le menu nous permet ensuite de receptionner la clé publique du serveur, qui sera nécessaire pour toutes communications avec lui car tous les messages seront cryptés avec sa clé. La clé est récupérée d'un serveur distant Apache qui sert le fichier tel quel. La clé publique du serveur est une clé 4096 bits RSA au format DER. L'application peut également envoyer sa clé publique au serveur, qui en aura besoin pour crypter et vérifier la signature des messages de notre téléphone. Le paquet envoyé contient un byte à 0x01, pour indiquer l'envoi d'une clé publique. Le reste du paquet contient la clé publique au format DER sous forme de tableau de bytes. Le tout est crypté avec la clé publique du serveur et est envoyé. Le serveur répond simplement par 0x01 en cas de succès et 0x02 en cas d'erreur.

L'application peut ensuite s'authentifier auprès du serveur, ce qui est l'opération principale. Le téléphone va générer un message, avec comme premier byte 0x02 (pour indiquer une demande de connexion), suivi du message "CHALLENGE" sous forme de tableau de byte, suivi de la signature de CHALLENGE avec sa clé privée. Le tout est crypté, toujours avec la clé publique du serveur, et est envoyé. Le serveur répond par 0x02 en cas d'erreur, et en cas de succès, renvoi un paquet "AUTH" signé avec sa clé privée. Signature qui sera vérifié par le téléphone grâce à sa clé publique.

Enfin, l'application peut aussi envoyer un simple paquet 0x03, toujours crypté avec la clé publique du serveur, auquel cas le serveur répondera avec un paquet "MESSAGE_SECRET" qui sera crypté avec la clé publique du téléphone. Le téléphone utilisera ainsi sa clé privée pour le déchiffrer et ainsi retrouver le message original.

Nous avons ainsi une application capable de crypter, décrypter, signer, et vérifier la signature. Et échanger des messages avec un serveur capable de faire de même. Le fait que le téléphone envoi un message signé et chiffré permet d'être sûr que seul le véritable serveur sera capable de lire le message, et le serveur lui aura la certitude, grâce à la signature, que c'est bien le véritable client qui a envoyé le message.

7.2 Kerberos

Pour notre partie Kerberos, comme mentionné plus haut, l'implémentation n'a pas été bien loin, à cause de méthodes nécessaires à l'utilisation de Kerberos qui n'étaient pas utilisables avec le SDK d'Android.

8 Comparaison des deux systèmes d'authentification

8.1 Pour les PC

Il existe quelques comparaisons pour ces deux systèmes que sont la PKI et Kerberos pour les PC. Notamment une comparaison par des tchèques [6]. Pour eux, ils différencient les systèmes cryptographiques en eux même. Pour Kerberos sysmétrique avec des tickets, pour la PKI asymétrique avec des certificats. Au final les concepts sont similaires. Kerberos a absolument besoin d'avoir son serveur d'authentification (KDC) en ligne pour fonctionner, la PKI peut se permettre une non disponibilité de sa CA (certification authority) pendant un instant. Kerberos fonctionne avec des mots de passe, alors que PKI utilise des clés privées. De ce fait la gestion n'est pas identique. La gestion des clés peut être source de problèmes, ainsi que la révocation de ces clés, impossible pour Kerberos. En terme de déploiement, c'est un peu plus compliqué pour Kerberos, qui doit enregistrer chaque utilisateur, et gérer les tickets qui ont une durée de vie limitée et qui doivent donc être recrées. Dans la PKI, les clés n'ont pas besoin d'être regénérées, une fois les clés distribuées, il n'y plus de manipulation à faire. PKI a un net avantage en terme de cryptage d'e-mail, qui est commun, et facile à mettre en oeuvre avec Enigmail pour Thunderbird par exemple. D'après eux, la PKI est plus facile à gérer et à mettre en oeuvre, ils la recommandent donc.

Dans une seconde comparaison d'un docteur suisse [7], est synthétisé avec un tableau qui est très explicite.

	Kerberos-based	PKI-based
Security	+	+
Non-repudiation	--	++
Trust requirements	-	+
Complexity	-	o
Scalability	--	+
Interoperability	--	-
Application modifications	--	-
Vendor support	o	+
Future perspectives	-	+

FIGURE 7. Comparaison de Kerberos et de la PKI

Dans ces deux cas, on voit nettement que la PKI est plus intéressante à utiliser. Va-t-on avoir les mêmes résultats pour notre cas, sur mobile ?

8.2 Pour notre problème

Côté PKI, nous avons de très bons résultats. Nos tests ont été réalisés sur l'émulateur android fourni avec l'API et sur une tablette tactile fonctionnant sous android. Nous avons réalisé une série de test, notamment la génération des clés publiques et privée.

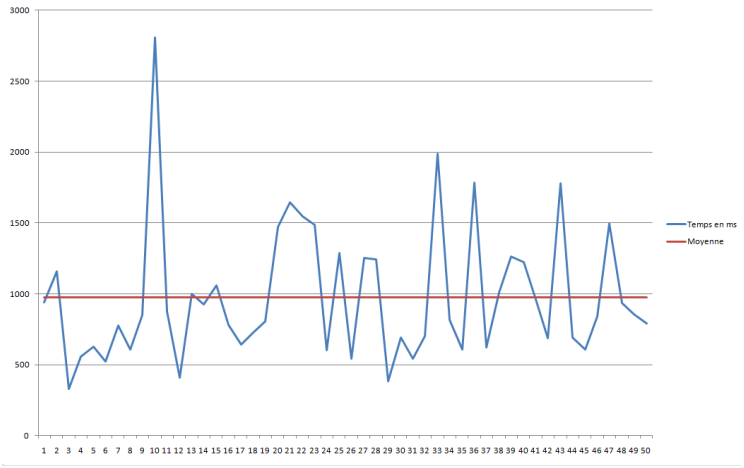

FIGURE 8. Génération des clés sur 50 essais

Cette figure montre sur 50 tentatives, le temps mis pour la génération de clés. En moyenne on s'aperçoit que l'application met un peu moins d'une seconde pour générer ses clés publique et privée.

Nous avons réalisé des tests sur le chiffrement d'un message "CHALLENGE", pour faire des statistiques en terme de performances de temps. Lors de ces tests, nous avons mesuré, le temps de signature du message avec la clé privée de notre application Android, le temps de chiffrement du message avec la clé publique du serveur, et le temps de transmission du message signé et chiffré au serveur.

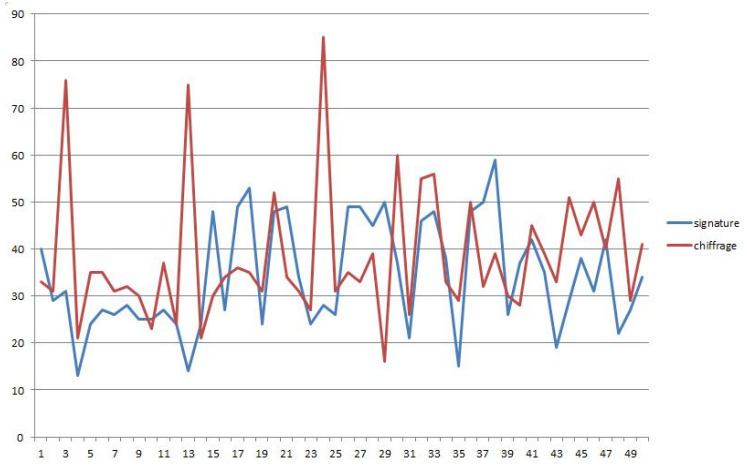

FIGURE 9. 50 essais de signature et de chiffrement du message "CHALLENGE"

Nos tests d'envoi du message au serveur se sont revelés plus excellent, d'une extrême rapidité.

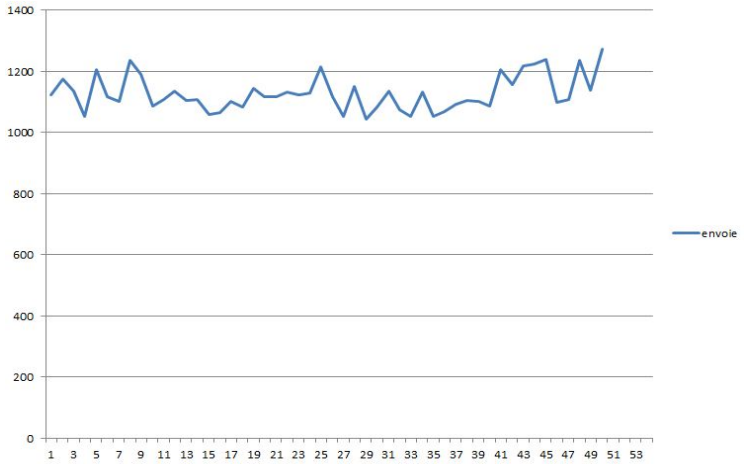

FIGURE 10. 50 essais d'envoi de message

Comme mentionné plus haut, nous n'avons pas pu réaliser de client Kerberos. Ainsi la comparaison va être un peu plus difficile. Cependant sur certain critère, nous pouvons tout de même répondre.

Récapitulons selon nos critères définis plus haut.

L'implémentation a été bien plus aisée pour la PKI. On peut se demander pourquoi. La raison est simple, c'est un système d'authentification très utilisé. Ainsi, nous avons pu trouver énormément de tutoriaux, de guides, et surtout les bonnes méthodes dans l'API pour développer un client PKI. Pour Kerberos, de nombreux guides sont disponibles sur Internet, mais très peu sont assez détaillé. Et comme Kerberos n'est pas vraiment utilisé pour le grand public, les méthodes utiles à son développement ne sont pas incluses dans l'API de Google, et surtout, cette API n'accepte pas l'API de Kerberos.

La rapidité d'exécution. D'après nos tests et nos graphiques précédents, on s'aperçoit que la PKI est vraiment très rapide. La raison étant que la PKI n'a besoin que d'un serveur, il n'y a donc pas beaucoup de connexion à réaliser contraiement à Kerberos, qui requiert deux serveurs, et implique de ce fait, bien plus d'échanges.

L'intéraction avec l'homme est tout aussi compliquée pour les deux protocoles. En effet, la PKI fonctionne avec un système de clés publiques et privées. L'utilisateur doit donc gérer, et protéger sa clé privée. Pour Kerberos, le système utilise des tickets, l'utilisateur n'utilise qu'un mot de passe, ou une clé symétrique pour communiquer avec le premier serveur, et réceptionner son ticket de service. Dans tous les cas, l'utilisateur doit garder quelque chose de secret, ce qui n'est pas forcement évident. Mais il est tout de même plus facile pour un utilisateur commum de mémoriser un mot de passe, plutôt que de gérer des clés qu'il doit transporter avec lui, ce qui peut être source de perte ou de vol, contrairement au mot de passe, qui lui, est censé rester dans la tête.

La sécurité du protocole repose d'un côté sur la clé privée, qui reste privée, et d'un autre côté sur un mot de passe ou une clé sysmétrique qui doit aussi rester secret. Pour la PKI le système est très sûr. Le seul moyen de "craquer" la sécurité serait la force brute, ou le vol de la clé privée. Pour Kerberos, l'attaquant peut avoir deux possibilités, soit attaquer le serveur, et lui voler la clé symétrique ou le mot de passe de l'utilisateur, ou voler directement l'utilisateur.

Notons que le plus simple pour voler la clé privée de la PKI ou le mot de passe de Kerberos et tout simplement de voler le téléphone. Il est donc recommandé de protéger ces données en les cryptant dans le téléphone.

La solidité des protocoles repose sur les points vus précédement. Il faut protéger les données privées.

Pour la scalabilité, on peut se douter que Kerberos est une fois de plus, plus compliqué que la PKI. En effet, le serveur de la PKI n'a besoin que de connaitre la clé publique du client, et inversement. Ainsi pour n utilisateurs, nous aurions n+1 clés publiques et n+1 clés privées. Pour Kerberos, il a les mots de passe, soit pour n utilisateurs, n mots de passe, mais en plus, il y a les tickets de services, qui sont différents pour chaque utilisateur et pour chaque service. De plus, ces tickets ont une durée de vie limitée, il

faut donc pour un même service et un même utilisateur, plusieurs tickets. Ce n'est pas des plus simple à gérer.

9 Conclusion

Pour conclure, bien que notre projet n'est pas pu être réalisé entièrement, nous pouvons tout de même voir que la PKI est très difficilement battable sur les critères de comparaison que nous avions décidé d'utiliser. Ainsi, bien que chacune des deux méthodes possèdes ses avantages et ses inconvénients, la méthode PKI reste très sûre tout en restant simple à déployer.

L'utilisation d'une authentification via une PKI nous semble relativement aisée pour une personne quelconque.

Références

1. M. Assora, J. Kadirire, and A. Shirvani, "Using wpki for security of web transaction," *Lecture Notes in Computer Science, 2007*, vol. 4655, pp. 11–20, 2007.
2. *WMLScript Crypto API Library : WAP-161-WMLScriptCrypto-20010620*.
3. M. Hassinen, K. Hyppönen, and E. Trichina, "Utilizing national public-key infrastructure in mobile payment systems," *Electronic Commerce Research and Applications*, vol. 7, pp. 214–231, 2008.
4. Y. Lee, J. Lee, and J. Song, "Design and implementation of wireless pki technology suitable for mobile phone in mobile-commerce," *Computer Communications*, vol. 30, pp. 893–903, 2006.
5. Y. Lei, A. Quintero, and S. Pierre, "Mobile services access and payment through reusable tickets," *Computer Communications*, vol. 32, pp. 602–610, 2009.
6. D. Kouril, L. Matyska, and M. Prochazka, "Kerberos and pki cooperation," 2006.
7. R. Oppliger, "Authentication and authorization infrastructures : Kerberos vs. pki,"