UNIVERSITÉ DE SHERBROOKE
Faculté de génie
Département de génie électrique et de génie informatique

SÉCURITÉ D'UNE APPLICATION DE COMMUNICATION MULTIMÉDIA SOUS PROTOCOLE IP DANS UN CONTEXTE MÉDICAL

Mémoire de maîtrise
Spécialité : génie électrique

Normand BÉDARD

Jury : Frédéric MAILHOT (co-directeur)
Alain HOULE (co-directeur)
Guy LÉPINE
Jean ROUAT

Sherbrooke (Québec) Canada MAI 2010

Your file Votre référence
ISBN: 978-0-494-65589-4
Our file Notre référence
ISBN: 978-0-494-65589-4

RÉSUMÉ

L'application MédicIP est un logiciel de télémédecine permettant à des spécialistes de la santé d'entrer en communication lors de situations d'urgence. Ce prototype, développé depuis 2005, permet des communications audio ainsi que le transfert d'électrocardiogrammes en temps réel.

Le scénario typique visé par ce projet était de permettre à une équipe ambulancière qui récupère un blessé sur la route, ou quelqu'un victime d'un malaise cardiaque, d'entrer en contact avec les hôpitaux les plus près afin de déterminer lequel est le plus apte à recevoir adéquatement ce patient. Cette approche permettrait d'améliorer la préparation, la qualité et la rapidité des opérations médicales à l'hôpital lorsque le patient se présente.

Le présent projet, SécureMédic, se veut un moyen d'aborder le problème de la sécurité entourant ce prototype, étant donné son utilisation dans un contexte médical. Une analyse de MédicIP a permis d'identifier quatre failles de sécurité critiques reliées aux authentifications usagers, aux établissements des conférences, aux transferts des données audio ainsi qu'aux transferts des électrocardiogrammes.

La contribution majeure de ce projet a été la création d'une infrastructure dédiée au processus d'authentification des usagers. Le système développé permet deux types d'authentification, fournissant ainsi un excellent niveau de robustesse. De plus, le serveur principal développé dans cette infrastructure intègre des mesures de protection permettant de minimiser les impacts de certains types d'attaque.

Le projet SécureMédic a permis de démontrer la faisabilité de la sécurisation d'une application de télémédecine en utilisant les techniques de protection et les standards actuels de l'industrie. Les résultats de tests comparatifs ont cependant permis de constater que des impacts reliés à la performance ont été engendrés par l'ajout des mesures de sécurité, dus principalement aux ressources requises pour le chiffrement et le déchiffrement des données dans un environnement multimédia temps réel.

Bien qu'il soit encore trop tôt pour envisager le déploiement du système actuel dans le milieu de la santé, les projets MédicIP et SécureMédic sont un pas dans la bonne direction. Le prototype actuel répond aux exigences techniques voulues, répond à un besoin bien réel et est parmi les premières solutions concrètes à démontrer la faisabilité d'un tel système. D'ici quelques années, l'apparition de solutions similaires est assurée.

Mots-clés : télémédecine, infrastructure à clés publiques, PKI, conférence multimédia sécurisée, SIP, TLS, SRTP.

REMERCIEMENTS

Je voudrais tout d'abord remercier mes directeurs de recherche Alain, Frédéric et Guy. Votre présence, vos conseils et votre support ont grandement contribué au succès de mes études graduées à la maîtrise. Vos exigences m'ont permis de me dépasser et de tirer le meilleur de moi-même. Ces deux ans de travail avec vous fûrent très agréables, enrichissants et significatifs dans mon parcours scolaire.

Je voudrais aussi te remercier Amélie. Ta compréhension, ton soutien moral et tes encouragements ont été très importants pendant ma maîtrise, mais plus particulièrement les semaines précédant ma conférence, que je considère le plus grand défi que j'ai relevé jusqu'à présent.

Je termine en remerciant mes parents, Monique et Robert. Même si vous n'avez pas été impliqués au quotidien durant ma maîtrise, c'est grâce à vos contributions sous toutes leurs formes si j'ai pu me rendre aujourd'hui jusqu'ici.

Merci à vous tous.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES ACRONYMES

Acronyme	Définition
AES	Advanced Encryption Standard
API	Application Programming Interface
CA	Certificate Authority
CSR	Certificate Signing Request
DLL	Dynamic-Link Library
DoS	Denial of Service
ePCR	Electronic Patient Care Record
FTP	File Transfer Protocol
HIPAA	Health Insurance Portability and Acountability Act
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IP	Internet Protocol
ISO	International Organization for Standardization
ITU	International Telecommunication Union Standardization Sector
LAN	Local Area Network
MD5	Message-Digest Algorithm 5
MIKEY	Multimadia Internet Keying
OSI	Open System Interconnection Reference Model
RA	Registration Authority
RFC	Request for Comments
PDA	Personal Digital Assistant
PID	Process Identifier
PKI	Public Key Infrastructure
RSA	Rivest Shamir Adleman
RTP	Real-time Transport Protocol
RTCP	Real-time Transport Control Protocol
SDP	Session Description Protocol
SIP	Session Initiation Protocol
SMTP	Simple Mail Transfer Protocol
SQL	Structured Query Language
SRTP	Secure Real-time Transport Protocol
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
VoIP	Voice Over Internet Protocol
XML	Extensible Markup Language

CHAPITRE 1

INTRODUCTION

Les réseaux IP sont aujourd'hui omniprésents dans notre vie quotidienne. Même si l'appellation Internet existe depuis 1974 pour désigner les réseaux TCP/IP [Information Sciences Institute, 1981b] [Information Sciences Institute, 1981a], ce n'est que vers la fin des années 90 qu'Internet est devenu une réalité pour la majorité de la population.

Depuis ce temps, ce secteur d'activité ne cesse de croître de façon impressionnante. On n'a qu'à penser aux courriels, aux moteurs de recherche, au clavardage, aux jeux en ligne ou au téléchargement de musique et de films pour s'en rendre compte. Plus récemment, les possibilités qu'offre Internet ont révolutionné un autre secteur d'activité : la santé.

Les logiciels de télémédecine permettent le transfert de données médicales via les lignes téléphoniques, Internet ou des réseaux dédiés. Les premières générations d'applications de télémédecine servaient principalement à envoyer de manière électronique des résultats médicaux. Cette façon de faire ne nécessitait pas la présence simultannée des interlocuteurs.

La génération suivante de logiciels de télémédecine a permis aux spécialistes de la santé de recevoir à distance les données provenant d'appareils médicaux. Un cas typique est celui d'un patient utilisant à la maison un appareil mesurant sa glycémie, et envoyant les résultats de manière automatique à son médecin.

La génération actuelle d'appareils de télémédecine permet une interaction complète entre le patient et le médecin, et ce, en temps réel. Ces outils profitent grandement des avantages et possibilités qu'offre Internet. Par contre, ils s'exposent aussi aux problèmes de ce canal de communication. La télémédecine permet en effet à des usagers distants de communiquer entre eux de l'information très sensible. Que ce soit des dossiers de patients, des données de signes vitaux ou simplement des discussions entre spécialistes voulant diagnostiquer un patient, toute cette information circule sur Internet, qui est un réseau à risques. Pour des raisons morales, de respect de la vie privée et de sécurité des patients, nous sommes en droit de nous demander si une telle approche est acceptable.

Cette situation nous amène à nous poser la question de recherche suivante :

> Les mécanismes et outils actuels des technologies de l'information permettent-ils de produire un environnement sécurisé pour une application de télémédecine utilisant Internet comme moyen de communication ?

L'importance et la position actuelle du commerce et des opérations bancaires en ligne nous portent à croire que la sécurisation d'une application utilisant Internet est une opération possible et réaliste au niveau technique. Comme ces types d'activités impliquent souvent des sommes d'argent considérables, les risques de fraudes sont omniprésents. Étant donné ces risques monétaires importants, il est probable que les technologies actuelles permettent de créer une architecture suffisamment sécuritaire pour permettre ce type d'opération. Par extrapolation, on peut établir comme hypothèse que les technologies présentes peuvent s'appliquer à un contexte médical pour fournir une architecture sécurisée.

Il faut cependant garder en tête qu'aucun système informatique n'est complètement infaillible. Que ce soit à cause d'attaques existantes, d'attaques théoriques ou actuellement inconnues qui seront fonctionnelles dans le futur, la sécurité d'une architecture logicielle n'est jamais à toute épreuve. C'est pourquoi le processus de sécurisation d'un système est évolutif dans le temps et vise à minimiser les risques qui y sont associés.

Dans le cas des applications de télémédecine, les risques liés à Internet, à l'entreposage, au transfert et à l'exactitude de l'information ainsi qu'à la robustesse du système se doivent eux aussi d'être réduits à un seuil acceptable. C'est uniquement de cette façon que la société aura confiance en ces types de service et qu'ils seront acceptés, ce qui leur permettra de s'implanter à grande échelle dans les réseaux de santé. Plusieurs impacts positifs pourront par la suite être observés :

- réduction des coûts du système de la santé ;
- augmentation de la productivité du personnel médical ;
- augmentation du niveau de qualité des traitements grâce à l'accès plus facile à des spécialistes ;
- réduction des séquelles des patients grâce à des interventions plus rapides ;
- ultimement, sauver la vie de patients.

Le mémoire présenté se veut un moyen de répondre à la problématique initiale soulevée. Plus précisément, la méthodologie suivante sera suivie.

Une présentation des objectifs du projet de recherche ainsi qu'une description d'un prototype d'application de télémédecine appelé MédicIP seront exposées au chapitre 2.

Le chapitre 3 présentera une étude de l'état de l'art dans le domaine de la sécurité informatique. Ce chapitre inclura de plus la description d'applications de télémédecine et de conférences multimédias déjà existantes.

Le chapitre 4 décrira de manière exhaustive l'architecture logicielle développée visant à sécuriser le prototype MédicIP.

Le chapitre 5 abordera concrètement le fonctionnement de l'architecture présenté au chapitre précédent. On y présentera la façon avec laquelle les différents modules de cette architecture interagissent.

Le chapitre 6 présentera l'analyse et la synthèse des résultats finaux du processus de sécurisation du prototype et finalement, le chapitre 7 se verra l'occasion de conclure sur une synthèse globale du projet de recherche.

CHAPITRE 2

DÉFINITION ET OBJECTIFS DU PROJET DE RECHERCHE

En situation d'urgence, la rapidité d'intervention du personnel médical est vitale. Même si plusieurs disciplines du domaine de la santé utilisent des appareils et logiciels de télémédecine de manière efficace, d'autres ne les utilisent pas correctement, ou tout simplement pas. Les communications entre les ambulances et les hôpitaux/urgentologues en sont un bon exemple.

- 10% des équipes ambulancières aux États-Unis utilisent l'ePCR (*Electronic Patient Care Record*). Ces appareils enregistrent l'historique des soins prodigués aux patients durant leur transport. L'inconvénient de ces dispositifs est qu'ils requièrent une personne supplémentaire dans l'équipe pour les utiliser [Zygowicz, 2008].
- 14% des hôpitaux aux États-Unis sont adéquatement équipés pour exploiter les données provenant des ePCR lors de l'arrivée des patients. Dans les autres cas, les patients sont traités avec peu d'information sur les traitements reçus avant l'arrivée à l'hôpital [Zygowicz, 2008].

Bien que l'ePCR ne constitue pas l'unique moyen de communication utilisé entre les ambulances et les hôpitaux, il reflète tout de même une réalité actuelle dans les systèmes de communication dans le milieu de la santé.

Le but global du projet de recherche est de produire une infrastructure logicielle permettant de remédier à cette réalité bien présente, c'est-à-dire de livrer une application de télémédecine efficace, simple d'utilisation et sécurisée. Plus précisément, le projet de recherche a pour rôle de cibler la portion sécurité d'une application de télémédecine existante. Les objectifs spécifiques qui y sont rattachés sont les suivants :

- développer un système d'authentification usager facile d'utilisation et sécuritaire ;
- permettre des échanges sécurisés (voix et données) entre les différents participants d'une conférence multimédia ;
- entreposer de manière sécuritaire les données sensibles utilisées par l'application de télémédecine.

Ces objectifs ont été déterminés suite à l'analyse d'un prototype d'application de télémédecine développée précédemment à l'Université de Sherbrooke : MédicIP. La section suivante décrira les objectifs de MédicIP, ses différents modules et leur fonctionnement. Finalement, les différents vecteurs d'attaques sur MédicIP seront exposés, permettant ainsi l'établissement des objectifs du processus de sécurisation de l'application.

2.1 Objectifs et contraintes du projet MédicIP

L'absence de moyens de communication efficaces et performants entre les ambulanciers et les cardiologues durant le transport d'un patient a motivé, en 2005, la création d'un prototype répondant à la question de recherche suivante [Dorais-Joncas, 2007] :

> Comment concevoir un moyen permettant de fournir l'accès à l'état d'un patient en temps réel durant son transport en ambulance et de partager ces informations avec d'autres intervenants ?

Ce premier prototype a notamment intégré la technologie ecgML [Wang *et al.*, 2003], permettant de représenter des électrocardiogrammes sous un format numérique bien précis, ainsi que la technologie SIP, utilisée dans un modèle de conférence complètement maillée.

Deux ans plus tard, en 2007, ce premier prototype a servi de base pour la création d'un second prototype aux allures d'une véritable application de télémédecine, MédicIP. L'équipe de développement de MédicIP a en effet proposé :

> Un système logiciel distribué reliant les ambulanciers, les urgentologues et les autres spécialistes médicaux permettant l'échange de données multimédias afin d'arriver au meilleur diagnostic possible, et ce, dans les plus brefs délais [Beauchemin *et al.*, 2007b].

Les objectifs globaux de ce projet étaient les suivants [Beauchemin *et al.*, 2007b] :

- permettre une communication multimédia entre les ambulanciers, les urgentologues et les autres spécialistes médicaux ;
- permettre l'échange de données physiologiques en temps réel (ex. : électrocardiogramme) ;

Plusieurs contraintes techniques ou choix technologiques étaient de plus reliés à ce projet [Beauchemin *et al.*, 2007b] :

- utilisation de divers protocoles et standards de transmission, de communication, de compression et de chiffrement de données ;

- utilisation de la pile SIP [Rosenberg *et al.*, 2002] 4.0 du client intermédiaire de MédicIP : M5T ;
- utilisation du protocole RTP [Schulzrinne *et al.*, 2003], du client intermédiaire M5T, pour les communications audio entre les clients ;
- utilisation de l'algorithme de compression de la voix Speex ;
- réimplémentation d'un système de communication complètement maillé (*full-mesh*) déjà existant conçu par Alexis Dorais-Joncas [Dorais-Joncas, 2007] ;
- implémentation d'un serveur de présence conçu par Fakher Ben Salah ([Salah, 2008]) ;
- utilisation du format numérique ecgML pour la transmission des électrocardiogrammes ;
- compatibilité avec le système d'exploitation Windows ;
- utilisation du langage de programmation C++ pour la communication ;
- utilisation du langage de programmation C# pour le client ;
- utilisation du pont C++ CLI pour relier la communication C++ au client C#.

2.2 Modules et fonctionnement du prototype

Le prototype MédicIP comprend plusieurs modules. La description de ceux-ci permettra une compréhension globale du fonctionnement du prototype.

2.2.1 Le client MédicIP

Le client MédicIP est une application graphique fonctionnant sous les systèmes d'exploitation Windows XP et Windows Vista. Voici l'allure graphique de cette application :

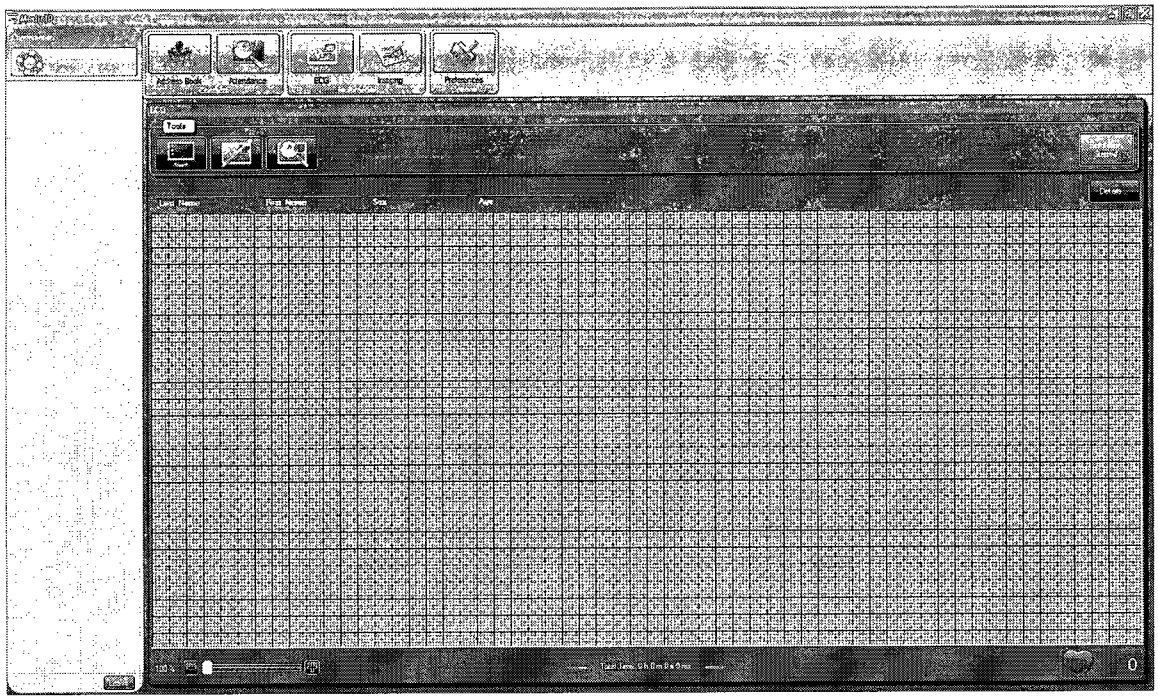


Figure 2.1 Capture d'écran du logiciel client MédicIP.

C'est cette application qui peut être utilisée par les spécialistes de la santé. La portion de gauche de l'application présente les différents utilisateurs d'une conférence multimédia ainsi que leur état. La portion supérieure expose les différents sous-menus (carnet d'adresses, présence, ECG, imagerie, préférences). Enfin, la partie inférieure droite regroupe les différents sous-menus affichés. Dans la figure précédente, il n'y a que la portion ECG qui est affichée. Le client MédicIP permet ainsi aux utilisateurs d'effectuer plusieurs actions :

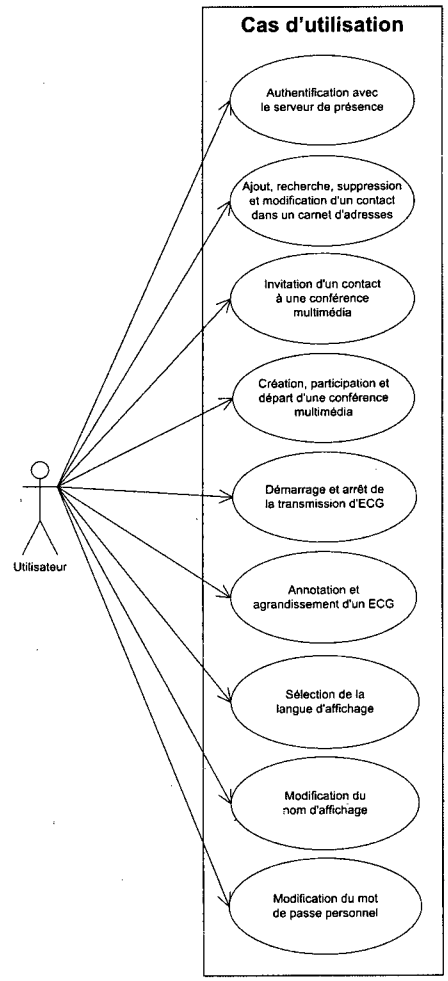


Figure 2.2 Diagramme de cas d'utilisation du prototype client MédicIP.

2.2.2 Le serveur de présence

Le serveur de présence offre trois fonctionnalités primaires dans le prototype MédicIP :

- gérer la présence des usagers ;
- authentifier les usagers utilisant un client MédicIP ;
- gérer les comptes utilisateurs.

Gérer la présence des usagers

L'objectif principal est de fournir un service permettant à un utilisateur (client MédicIP) d'être informé du changement d'état d'un lot d'utilisateurs, par exemple les utilisateurs de son carnet d'adresses. Ces changements d'état peuvent être les suivants :

- changement du nom d'affichage ;
- changement d'état (en ligne, déconnecté, occupé, absent) ;
- changement de spécialité ;
- changement d'endroit (localisation) ;
- changement d'adresse IP (usage interne uniquement) ;
- changement de ports de connexion (usage interne uniquement) ;

L'utilité d'un tel système est que si, par exemple, un usager se connecte avec une nouvelle adresse IP, car il a changé de client MédicIP, les autres utilisateurs l'ayant sur leur liste de contacts seront d'abord avertis que l'usager est en ligne et ensuite, qu'il est joignable à une nouvelle adresse IP.

Le serveur de présence possède de plus une liste complète des informations de tous les utilisateurs sous la forme de fichiers XML. Cela permet aux clients MédicIP de faire des recherches de contacts (afin de les ajouter à leur carnet d'adresses) en effectuant des requêtes au serveur de présence.

Authentifier les usagers utilisant un client MédicIP

Afin de pouvoir utiliser le client MédicIP, un utilisateur doit s'authentifier à l'aide de son nom d'utilisateur et de son mot de passe. Comme mentionné précédemment, le serveur de présence possède toutes les informations des usagers, y compris celles nécessaires à l'authentification. C'est donc au serveur de présence que les requêtes d'authentification sont envoyées afin de pouvoir utiliser l'application de télémédecine. Le chapitre 2.3.1 présente les détails techniques de ce processus.

Gérer les comptes utilisateurs

Le serveur de présence permet d'effectuer la gestion des comptes utilisateurs par l'entremise d'une application graphique fonctionnant sur les systèmes d'exploitation Windows XP et Windows Vista. Depuis cette application, il est possible d'ajouter, de supprimer et de modifier un compte utilisateur. Ceux-ci sont entreposés sous la forme de fichiers XML.

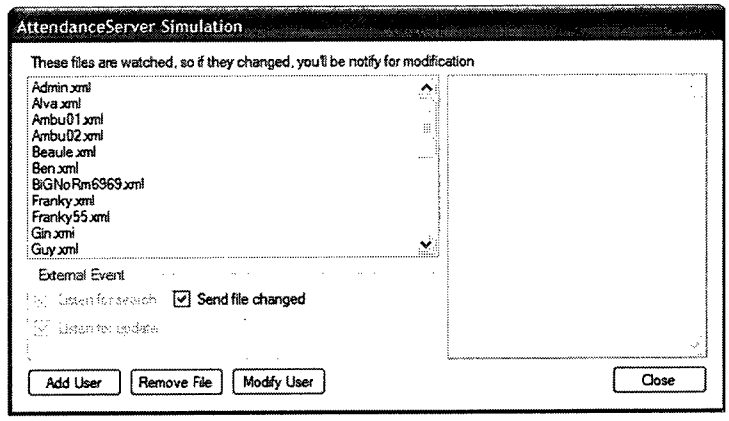


Figure 2.3 Capture d'écran de l'interface graphique du serveur de présence.

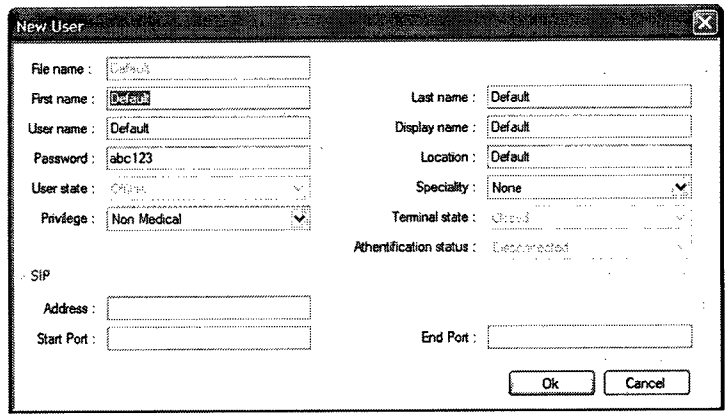


Figure 2.4 Capture d'écran du menu d'ajout d'un utilisateur.

2.2.3 Les communications

Les communications constituent un élément important du prototype MédicIP. Plusieurs technologies y sont intégrées :

- SIP (*Session Initiation Protocol*) (voir chapitre 3.4.3) ;
- communications complètement maillées (*full-mesh*) ;
- RTP (*Real-time Transport Protocol*) (voir chapitre 3.4.2) ;

Premièrement, le protocole SIP est utilisé afin d'établir les conférences entre les utilisateurs de MédicIP. Il s'agit d'un protocole de signalisation utilisé pour créer, modifier et terminer des sessions multimédias entre un ou plusieurs participants connectés entre eux au moyen d'un réseau IP [Dorais-Joncas, 2007]. Il existe plusieurs implémentations de ce protocole dont certaines sont gratuites (*open source*) et d'autres sont commerciales. La pile SIP

4.0 de l'entreprise M5T [M5T, 2008] a été utilisée dans ce prototype. Des explications supplémentaires sur ce protocole sont disponibles au chapitre 3.4.3.

Deuxièmement, le prototype MédicIP est basé sur des travaux [Dorais-Joncas, 2007] permettant l'utilisation du protocole SIP en mode *full-mesh*. Ce type de communication diffère du modèle de communication standard entre des clients et un serveur. En effet, une communication de type *full-mesh* implique que chaque paire d'entités est responsable de maintenir une communication directe entre elles. Pour n entités, il y aura donc n*(n-1)/2 communications. Voici la représentation graphique d'une communication complètement maillée :

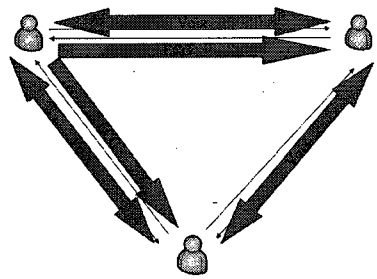


Figure 2.5 Conférence complètement maillée entre 3 clients MédicIP transférant de la voix et un électrocardiogramme [Beauchemin *et al.*, 2007a].

Afin d'établir ce type de conférence à l'aide du protocole SIP, la séquence de messages suivante doit être effectuée :

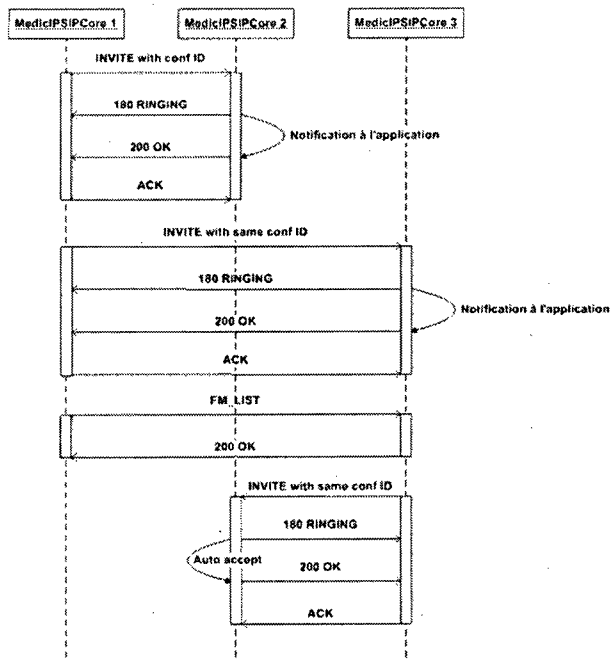


Figure 2.6 Établissement d'une communication complètement maillée avec le protocole SIP [Beauchemin *et al.*, 2007a].

Troisièmement, le prototype MédicIP permet les communications vocales entre les entités d'une conférence. Le protocole RTP (*Real-time Transport Protocol*) est utilisé afin de faire ce type d'échange (voir chapitre 3.4.2).

2.2.4 Résumé du fonctionnement global

En somme, le prototype MédicIP se compose d'un serveur de présence ainsi que de clients MédicIP. Un utilisateur doit s'authentifier au serveur de présence afin de pouvoir utiliser l'application. Ensuite, il est en mesure de gérer son carnet d'adresses personnel et de participer à des conférences multimédias pleinement maillées. Les utilisateurs sont en mesure de communiquer entre eux tant par les communications audios en temps réel que par l'envoi d'électrocardiogrammes.

De plus, il est possible d'effectuer certaines opérations sur les électrocardiogrammes comme ajouter des annotations sur ceux-ci ou leur associer des informations de base sur un patient (nom, prénom, sexe, âge, etc.).

Pour de plus amples informations sur les détails techniques ou les fonctionnalités avancées du prototype MédicIP, ainsi que sur les communications *full-mesh*, il est possible de se référer respectivement à [Beauchemin *et al.*, 2007b] et [Dorais-Joncas, 2007].

2.3 Vecteurs d'attaque et failles de sécurité du prototype MédicIP

Comme mentionné précédemment, les objectifs spécifiques du projet de recherche ont été déterminés grâce à l'analyse globale de la sécurité du prototype MédicIP. Les aspects suivants ont fait l'objet d'analyses :

- authentification client ;
- communication entre entités ;
- entreposage des données.

2.3.1 Authentification client

Le prototype MédicIP souffre d'un problème de sécurité important au niveau de l'authentification client. Voici la séquence générique des opérations lors de l'authentification :

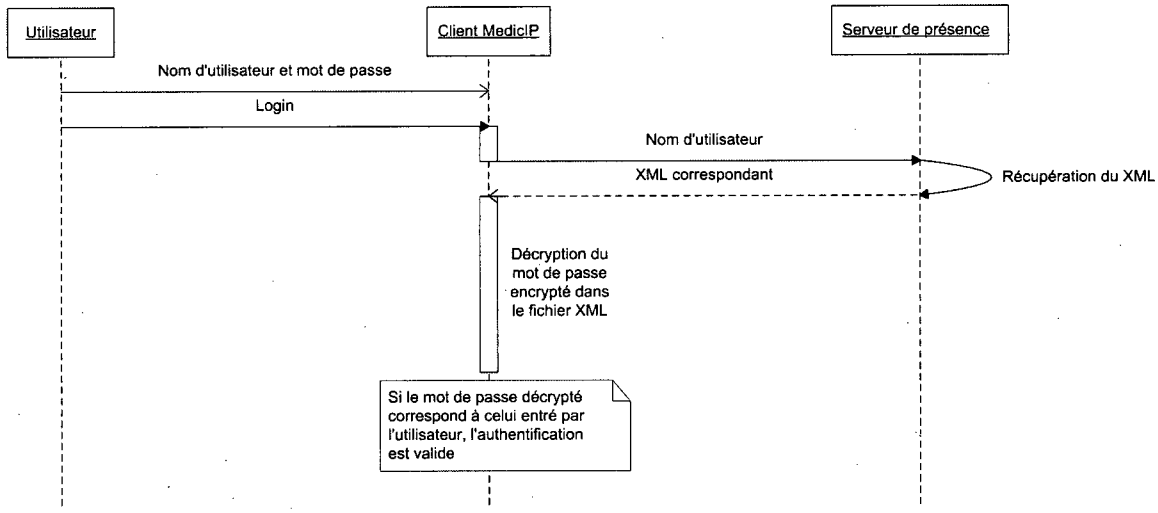


Figure 2.7 Diagramme de séquence du processus d'authentification de MédicIP.

On remarque qu'en tout temps, le client MédicIP se fait envoyer le fichier XML correspondant au nom d'utilisateur spécifié par l'usager. Ce fichier contient les informations personnelles suivantes :

- prénom ;
- nom de famille ;
- mot de passe (chiffré) ;
- nom d'affichage ;
- état ;

- spécialité ;
- niveau de privilèges ;
- situation géographique ;
- adresse IP ;
- ports de connexion.

Ces informations ne sont pas chiffrées et sont envoyées à un ordinateur client même si une authentification échoue. Pire encore, en aucun temps le serveur de présence ne valide l'authenticité du client lui envoyant une requête. Cela signifie que n'importe qui connaissant l'adresse IP du serveur de présence peut fabriquer des requêtes permettant de récupérer les données confidentielles mentionnées précédemment.

Voici une capture d'écran illustrant les données transférées lors du processus d'authentification :

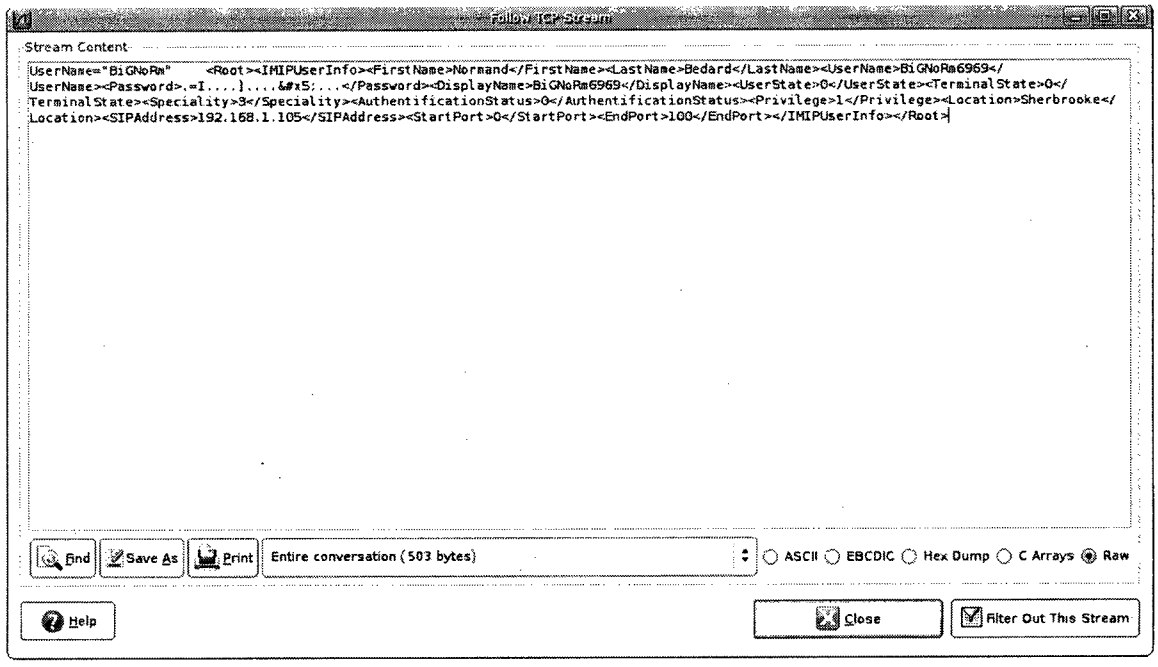


Figure 2.8 Capture de paquets lors du processus d'authentification de MédicIP.

Que ce soit depuis le client ou lors du transfert du fichier XML, ces données sont accessibles en clair. L'information recueillie par un attaquant pourrait être utile dans diverses situations (déni de service, ingénierie sociale, pourriel, infiltration de réseaux, vol d'informations confidentielles, etc.).

2.3.2 Communications entre entités

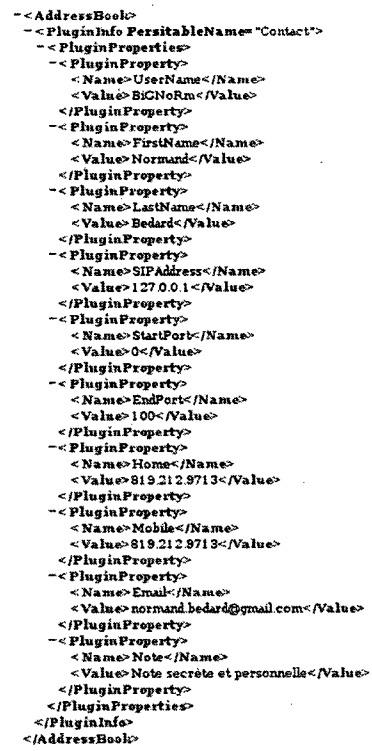
Lors de l'établissement d'une conférence multimédia, il y a utilisation du protocole SIP pour permettre la signalisation de la conférence (voir chapitre 3.4.3). Utilisé seul, le protocole SIP est sujet à plusieurs menaces . Parmi les menaces classiques, on retrouvent le détournement d'un enregistrement et l'usurpation d'un serveur. Une liste complète est disponible dans le RFC 3261 (*Request for Comments*) (RFC 3261, section 26 [Rosenberg *et al.*, 2002]).

Pendant une conférence, les échanges audios se font à l'aide du protocole RTP. Contrairement au protocole SRTP [Baugher *et al.*, 2004], RTP ne supporte pas le chiffrement, l'authentification et l'intégrité de messages, ni une protection contre les attaques de reprise (*Replay Attacks*). Ceci implique que les conversations vocales peuvent être interceptées, consultées et/ou modifiées.

Les échanges des électrocardiogrammes, quant à eux, se font à l'aide du protocole UDP. Ce protocole, situé au niveau 4 de la couche de l'OSI, a pour unique rôle le transport de paquets. Utilisé seul, celui-ci ne procure pas de mécanisme afin d'assurer la confidentialité des échanges.

2.3.3 Entreposage des données

Au niveau du client MédicIP, certaines données sont entreposées localement. Le carnet d'adresses d'un utilisateur en est un bon exemple : il est enregistré sous la forme d'un fichier XML nommé AddressBook.xml dans le répertoire C :/Program Files/Persistence. Ce fichier est libre d'accès et on peut y consulter certaines informations privées comme, entre autres, le numéro de téléphone de l'utilisateur, son adresse courriel et son adresse IP. Voici un exemple de carnet d'adresses récupéré sur un client MédicIP :



```
- <AddressBook>
  - <PluginInfo PersitableName="Contact">
    - <PluginProperties>
      - <PluginProperty>
          <Name>UserName</Name>
          <Value>BiGNoRm</Value>
        </PluginProperty>
      - <PluginProperty>
          <Name>FirstName</Name>
          <Value>Normand</Value>
        </PluginProperty>
      - <PluginProperty>
          <Name>LastName</Name>
          <Value>Bedard</Value>
        </PluginProperty>
      - <PluginProperty>
          <Name>SIPAddress</Name>
          <Value>127.0.0.1</Value>
        </PluginProperty>
      - <PluginProperty>
          <Name>StartPort</Name>
          <Value>0</Value>
        </PluginProperty>
      - <PluginProperty>
          <Name>EndPort</Name>
          <Value>100</Value>
        </PluginProperty>
      - <PluginProperty>
          <Name>Home</Name>
          <Value>819.212.9713</Value>
        </PluginProperty>
      - <PluginProperty>
          <Name>Mobile</Name>
          <Value>819.212.9713</Value>
        </PluginProperty>
      - <PluginProperty>
          <Name>Email</Name>
          <Value>normand.bedard@gmail.com</Value>
        </PluginProperty>
      - <PluginProperty>
          <Name>Note</Name>
          <Value>Note secrète et personnelle</Value>
        </PluginProperty>
      </PluginProperties>
    </PluginInfo>
  </AddressBook>
```


Figure 2.9 Extrait du fichier AddressBook.xml.

Bien qu'il ne s'agisse pas ici d'un élément de sécurité proprement dit, il aurait été intéressant que ce carnet d'adresses soit situé sur un serveur externe, de manière à y avoir accès depuis n'importe quel client MédicIP. La section 7.3.5 abordera les travaux futurs concernant cet aspect.

Néanmoins, la présence de ces informations non protégées sur le client est problématique. Les données médicales sont parmi les données les plus sensibles et critiques et requièrent de fortes mesures de sécurité pour les critères suivants [Saffron et Ring, 1995].

- confidentialité / accessibilité ;
- intégrité ;
- disponibilité.

L'ensemble de ces critères n'est pas respecté par MédicIP, ce qui représente une faille de sécurité importante du prototype. Un pirate qui réussirait à avoir accès à un ordinateur client pourrait détruire le contenu du carnet d'adresses ou le chiffrer afin de demander une compensation monétaire pour redonner l'accès des documents aux propriétaires.

2.4 Résumé

En somme, la description du prototype MédicIP a permis de cadrer le contexte d'utilisation de l'application. Une analyse des failles de sécurité du prototype a permis de constater les risques encourus ainsi que les dommages pouvant être causés par certains vecteurs d'attaque.

Cette analyse permet de réaffirmer les objectifs du projet de recherche qui ont été soulevés en début de chapitre :

- développer un système d'authentification usager facile d'utilisation et sécuritaire ;
- permettre des échanges sécurisés (voix et données) entre les différents utilisateurs d'une conférence multimédia ;
- entreposer de manière sécuritaire les données sensibles utilisées par l'application de télémédecine.

Voici le résumé des failles de sécurité du prototype MédicIP :

Failles de sécurité identifiées	Solutions envisagées	Solutions implémentées
Communication non sécurisée lors du processus d'authentification d'un usager		
Signalisation SIP non sécuritaire lors de la création d'une conférence		
Communication audio non sécuritaire		
Échange d'électrocardiogrammes non sécuritaire		
Entreposage non sécurisé du carnet d'adresses	Utiliser un carnet d'adresses centralisé	

Figure 2.10 Failles de sécurité du prototype MédicIP.

Pour arriver à remplir ces objectifs et trouver des solutions à envisager, une étude de l'état de l'art dans le domaine de la sécurité informatique devra être faite, de façon à cibler les moyens optimaux pour corriger les problèmes soulevés. Ce processus sera le sujet du prochain chapitre.

CHAPITRE 3
ÉTAT DE L'ART ET CADRE DE RÉFÉRENCE

Les différentes failles de sécurité observées au chapitre précédent touchent plusieurs domaines de la sécurité informatique :

- les techniques d'authentifications de type usager ;
- la cryptographie ;
- les infrastructures à clés publiques ;
- les protocoles de communication.

L'étude de l'état de l'art de ces sujets offrira une perspective didactique expliquant à haut niveau les concepts à maîtriser, et permettra de plus la mise en oeuvre d'un processus de sécurisation du prototype MédicIP.

Il est important de souligner que le contexte du projet de recherche se situe au niveau de la sécurisation du prototype MédicIP et ne vise en aucun cas la conception ou l'ajout de fonctionalités au logiciel MédicIP. Par conséquent, l'étude de l'état de l'art se concentrera principalement sur les concepts de sécurité ciblant les failles de sécurité identifiées précédemment. Néanmoins, une petite portion de l'état de l'art abordera brièvement les types de logiciel de télémédecine existants. Pour de plus amples informations à propos de l'étude des logiciels de télémédecine et des fonctionalités requises, il est possible de se référer à la documentation du projet MédicIP [Beauchemin *et al.*, 2007b].

3.1 Types d'authentification

L'authentification est le processus de vérification de l'identité numérique d'un utilisateur, d'un client, d'un serveur ou d'une entité quelconque. Plusieurs méthodes existent afin d'effectuer ce processus.

3.1.1 Mot de passe

Même s'il existe aujourd'hui de plus en plus de méthodes d'authentification élaborées et efficaces, l'utilisation de mots de passe demeure tout de même le moyen d'authentification le plus répandu [Campbell *et al.*, 2007]. Techniquement, l'utilisation de mots de passe comme moyen d'authentification procure un niveau de sécurité suffisant pour un grand

nombre d'applications [Campbell *et al.*, 2007], tout en étant facile d'utilisation pour un usager. En revanche, cette méthode peut être la cible de plusieurs types d'attaque bien connus [Marechal, 2007].

Afin de limiter les chances de succès de certains de ces vecteurs d'attaque, plusieurs pratiques existent quant à la sélection d'un mot de passe fiable. Bien qu'aucun standard ou politique unique ne certifie la force d'un mot de passe, les règles suivantes sont souvent conseillées [McDowell *et al.*, 2004] :

- ne pas être fondé d'après une information personnelle ;
- contenir un nombre de caractères minimum ;
- utiliser des lettres minuscules et majuscules ;
- utiliser des chiffres ;
- utiliser des caractères spéciaux ;
- ne pas utiliser des mots, noms, surnoms, marques de commerce, expressions populaires ou des parties de ceux-ci.

3.1.2 Cartes d'identification

Ce type de méthode d'authentification possède un éventail de produits très diversifié, tant au niveau des technologies utilisées que des possibilités qu'il offre.

Cartes de proximité

Ce moyen d'authentification utilise la technologie des radiofréquences. Pour s'identifier, un utilisateur doit posséder ce type de carte, qu'il n'a qu'à approcher d'un lecteur de cartes prévu à cet effet. Celui-ci émet en permanence de basses fréquences radio perceptibles à proximité. Lorsque la carte entre dans ce champ, celle-ci reçoit assez d'énergie du lecteur pour être en mesure d'émettre, depuis sa propre antenne, son identifiant unique. Le système composé du lecteur de cartes peut ensuite valider cet identifiant et effectuer les actions nécessaires [Timms, 1990].

Bien que ce type d'authentification soit très sécuritaire étant donné la complexité requise pour falsifier ce type de carte [Timms, 1990], il n'en demeure pas moins qu'elle est sujette au vol. Une fois volée, elle peut être utilisée par n'importe qui, car il n'existe aucun lien de sécurité entre la carte et la personne qui la possède.

Cartes intelligentes (*smart cards*)

Les cartes intelligentes apportent des fonctionnalités plus évoluées qu'un simple identifiant unique. Bien qu'il existe une panoplie de définitions et de modèles de cartes intelligentes, elles ont toutes un point en commun : la possibilité de se faire programmer grâce à l'existence de puces électroniques sur celles-ci. En effet, elles sont généralement constituées d'un microprocesseur, de plusieurs types de mémoire, ainsi que d'un port de communication. Elles sont en mesure d'effectuer des opérations arithmétiques, des décisions logiques, des opérations de lecture et d'écriture en mémoire, d'utiliser des algorithmes d'authentification et comportent différents protocoles de communication [Noore, 2000]. Les normes ISO 7816-X visent à standardiser les spécifications de ce type de cartes [International Organization for Standardization (ISO), 2007] :

- caractéristiques physiques ;
- dimension et disposition des contacts électriques ;
- signaux électroniques et protocoles de communication ;
- commandes pour les opérations de sécurité ;
- commandes pour la gestion de la carte ;
- applications cryptographiques s'y rattachant ;
- etc.

Une approche courante est d'enregistrer un certificat numérique (chapitre 3.3.1) sur cette carte. Ces certificats sont l'équivalent numérique des passeports [Adams et Ferrel, 1999]. Ils contiennent principalement l'identité de l'usager, l'autorité ayant délivré le certificat, la clé publique de l'usager ainsi que la période de validité du certificat. Lorsque deux entités ont confiance en une même autorité de certification, elles peuvent donc se référer à celle-ci pour valider le certificat d'autrui et ainsi avoir la certitude de l'identité de cette personne [Rescorla, 2000]

Ces cartes, bien qu'elles permettent davantage de possibilités que les cartes de proximité standards, sont elles aussi sujettes au vol.

Cartes biométriques

Ces cartes sont en quelque sorte une évolution des *smart cards* traditionnelles. En effet, en plus de fournir les mêmes fonctionnalités, elles possèdent des capteurs biométriques. Ces capteurs utilisent les caractéristiques physiologiques des êtres humains pour valider l'identité d'un individu. Parmi les capteurs biométriques courants, on en retrouve pour la géométrie des mains, la géométrie des doigts, les empreintes digitales, les propriétés de l'iris, les modèles rétiniens, les propriétés du visage, la voix, la dynamique des entrées au clavier, la forme de l'oreille, etc. [Jain *et al.*, 1999].

Bien que des modèles de cartes biométriques et leurs lecteurs possèdent des capteurs de toutes sortes, le lecteur d'empreintes digitales est parmi les plus utilisés, entre autres à cause de sa taille et de sa maturation technologique. Ces capteurs peuvent être intégrés directement sur la carte qui elle, possède un processeur de cryptographie spécifique ainsi que des algorithmes d'analyse d'empreintes digitales pour valider l'identité de l'individu possédant la carte [Noore, 2000]. Le traitement et l'analyse de l'identité de la personne peuvent donc se faire exclusivement sur la carte.

De tels systèmes rendent la contrefaçon encore plus difficile et réduisent considérablement les possibilités d'utilisation de la carte après un vol.

Cartes intelligentes réseautées (*Network smart cards*)

Bien que les *smart cards* offrent plusieurs avantages comme la sécurité, la portabilité, la difficulté de duplication et la résistance aux modifications non autorisées (*tamper resistant*), elles possèdent certains inconvénients [Ali et Montgomery, 2005] :

- Impossibilité de garantir la sécurité des informations après que celles-ci aient été récupérées sur la carte. L'entreposage sécuritaire d'informations sur une carte est une chose, les utiliser de manière sécuritaire en est une autre.
- Les commerçants sur Internet peuvent avoir un haut niveau de confiance envers l'information présente sur une *smart card* mais pas envers l'ordinateur qui l'utilise ni le canal de communication utilisé.
- Lorsque les *smart cards* sont connectées à un ordinateur, les applications ne peuvent les utiliser directement de concert avec les différentes interfaces réseau. L'utilisation de matériel ou de logiciels spécifiques (pilotes (*drivers*), *middlewares*) est nécessaire.

Les *network smart cards* apportent de nouvelles fonctionnalités permettant de corriger ces lacunes. En effet, les standards de communication TCP/IP ainsi que SSL/TLS [Dierks et Rescola, 2008] sont supportés et intégrés à même la carte [Ali et Montgomery, 2005]. Cela leur permet d'une part d'être perçues comme un noeud réseau indépendant, tout comme n'importe quel ordinateur distant effectuant une requête, et d'autre part, d'éliminer la nécessité d'utiliser des pilotes (*drivers*) ou *middlewares* spécifiques [Ali et Montgomery, 2005].

Ce type de carte permet donc des authentifications et des échanges sur Internet de manière plus sécuritaire.

3.2 Cryptographie

La cryptographie est la pratique et la science de rendre illisible de l'information. Ce processus implique trois grands types d'opérations :

- le chiffrement ;
- le déchiffrement ;
- le hachage.

Le chiffrement est le processus par lequel un message initial A est transformé, à l'aide d'algorithmes mathématiques, en un message brouillé B. Ce processus requiert l'utilisation d'une clé de chiffrement, en plus du message à chiffrer.

Le déchiffrement est le processus inverse du chiffrement, par lequel un message chiffré B est déchiffré pour obtenir le message original A. Ce processus requiert l'utilisation d'une clé de déchiffrement, en plus du message à déchiffrer.

Le chiffrement et le déchiffrement sont généralement utilisés soit pour stocker de l'information de manière sécuritaire, soit pour protéger de l'information lors de son transfert dans un canal de communication non sécurisé entre deux entités.

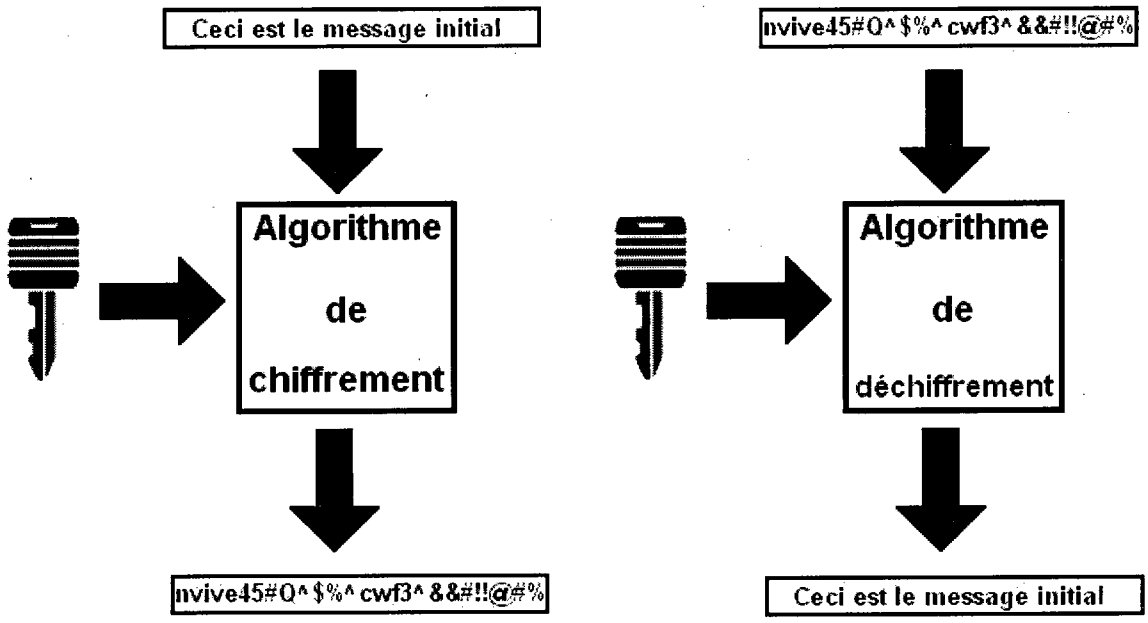


Figure 3.1 Processus générique de chiffrement et de déchiffrement.

Le hachage, aussi appelé *hashing*, est un processus par lequel un ensemble de données X de taille quelconque est transformé en un ensemble de données de longueur fixe Y à l'aide d'un algorithme mathématique. Une fonction de hachage idéale possède les caractéristiques suivantes :

- il existe un seul hachage pour un message donné ;
- il est facile et rapide de calculer le hachage de n'importe quel message ;
- il est impossible de trouver le message initial depuis son hachage ;
- il est impossible de modifier un message sans changer son hachage ;
- il est très difficile de trouver deux messages différents qui ont le même hachage.

Le hachage peut servir dans diverses circonstances :

- vérification de l'intégrité d'un message en comparant le hachage du message avant et après son transfert/utilisation.
- stockage de mots de passe de manière sécuritaire (il est impossible de retrouver le mot de passe depuis son hachage, mais il est possible de comparer le hachage d'un mot de passe à valider avec le hachage original stocké).
- signature numérique de document (le hachage d'un document représente son empreinte unique).

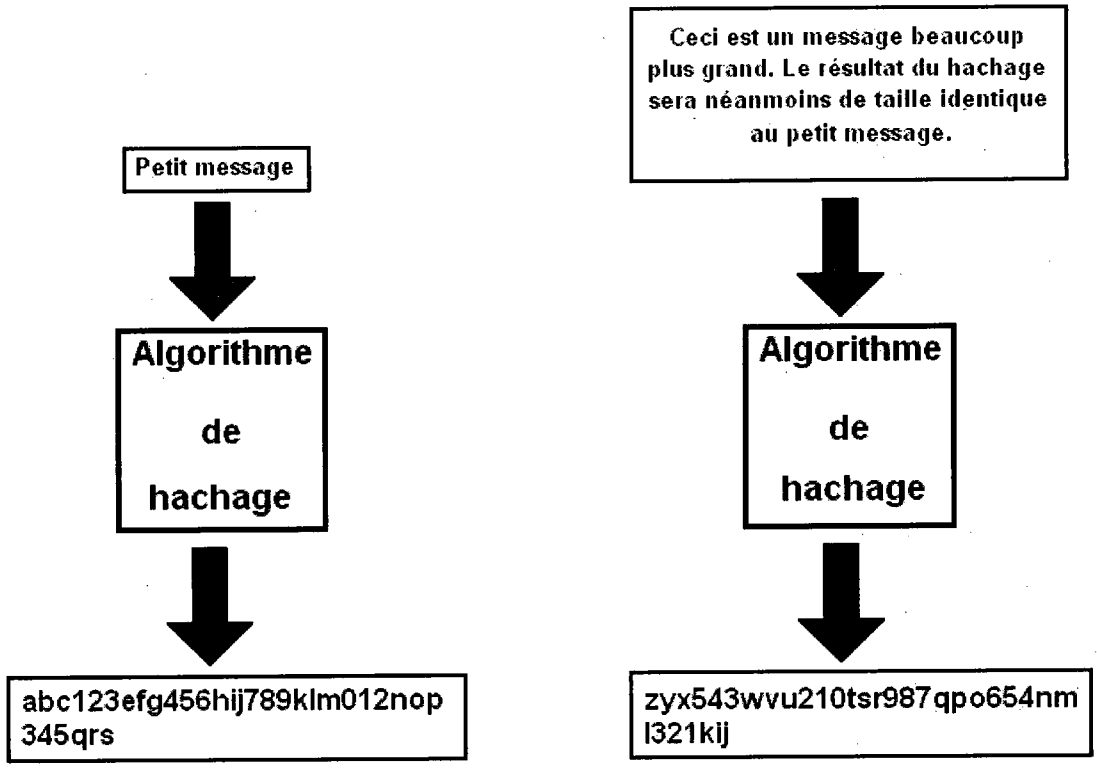


Figure 3.2 Processus générique du hachage.

Les processus de chiffrement et de déchiffrement sont regroupés en deux catégories bien distinctes :

- la cryptographie symétrique ;
- la cryptographie asymétrique.

Ces catégories seront abordées dans les sections qui suivent.

3.2.1 Cryptographie symétrique

La cryptographie symétrique réfère à des algorithmes de cryptographie qui requièrent une même clé pour le chiffrement et le déchiffrement de messages. Cette technique nécessite donc que les deux entités voulant communiquer possèdent cette même clé afin de s'échanger un message de manière sécuritaire. Les algorithmes de cryptographie symétrique sont typiquement très performants. Par contre, utilisés seuls, ces algorithmes posent un certain problème, soit celui d'échanger de manière sécuritaire la clé symétrique entre les deux entités.

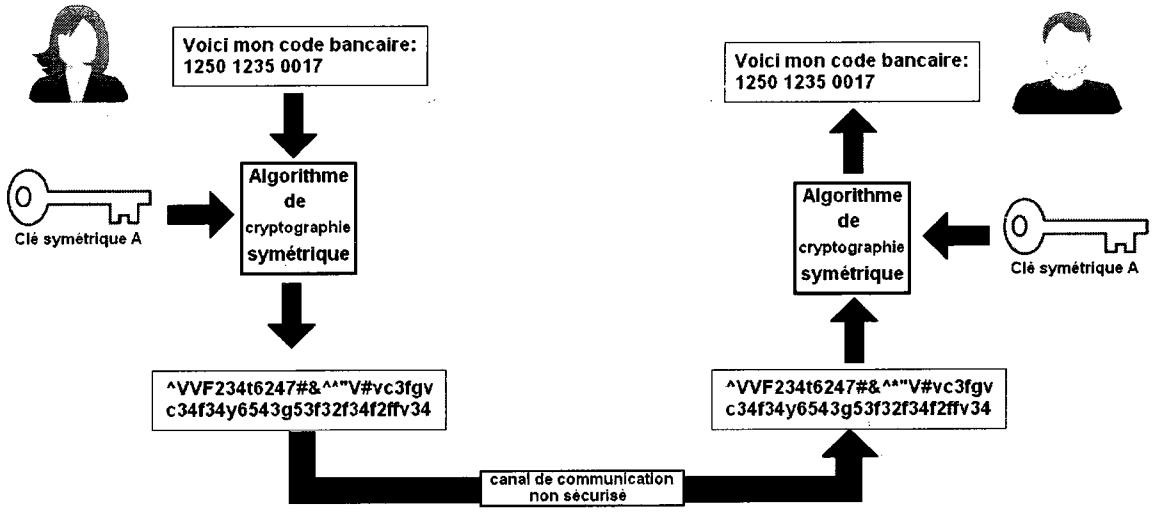


Figure 3.3 Exemple d'une communication utilisant la cryptographie symétrique.

Parmis les algorithmes de cryptographie, on retrouve :

- Twofish ;
- Serpent ;
- AES ;
- Blowfish.

3.2.2 Cryptographie asymétrique

La cryptographie asymétrique, aussi appelée cryptographie à clés publiques, réfère à des algorithmes de cryptographie qui requièrent deux clés différentes. La clé qui sert au chiffrement d'un message n'est pas la même que celle utilisée pour déchiffrer le message. Cette paire de clés (A et B) doit être générée de manière à posséder les caractéristiques suivantes :

- A permet de chiffrer un message pouvant être déchiffré uniquement par B ;
- B permet de chiffrer un message pouvant être déchiffré uniquement par A ;
- A ne peut être déduit de B ;
- B ne peut être déduit de A.

Une de ces clés est dite publique, ce qui signifie qu'elle est connue de tous, on peut donc la distribuer sur Internet sans aucun risque. L'autre clé, quant à elle, est dite privée. Cette clé se doit d'être connue exclusivement par le propriétaire de la paire de clés. Ce principe fondamental est à la base d'un système à clés publiques ; si la clé privée est révélée, toute la sécurité des échanges est mise en péril. Les algorithmes de cryptographie de

type asymétrique sont typiquement peu performants, mais ne requièrent pas le partage et l'échange sécuritaire d'une clé unique comme c'est le cas pour la cryptographie symétrique. Les systèmes asymétriques requièrent uniquement que les entités voulant communiquer entre elles de manière bidirectionnelle possèdent chacune leur propre paire de clés, et qu'elles s'échangent leur clé publique, qui ne requiert aucun moyen de protection.

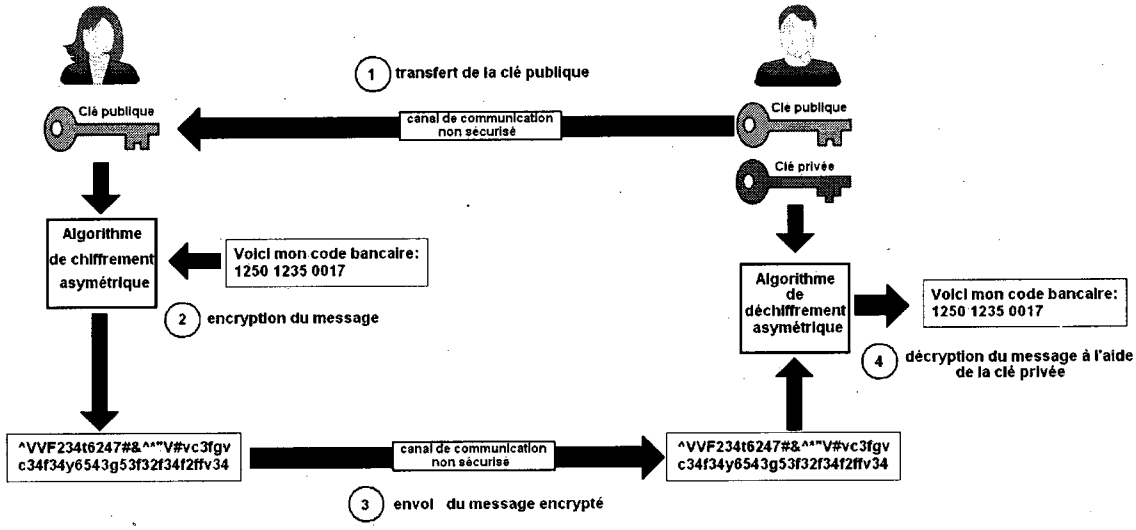


Figure 3.4 Exemple d'une communication unidirectionnelle utilisant la cryptographie asymétrique.

Parmis les algorithmes de cryptographie, on retrouve :

- Diffie-Hellman ;
- ElGamal ;
- Elliptic Curve ;
- RSA.

Afin d'appuyer le concept général de l'encryption asymétrique, présenté dans le graphique suivant, une description du fonctionnement du protocole RSA sera présentée.

Protocole RSA

Les clés publiques et privées du protocole RSA sont basées sur l'utilisation des nombres premiers. Premièrement, on doit déterminer deux nombres, qu'on nomme respectivement "p" et "q". Pour des raisons de sécurité, ces nombres doivent être déterminés de façon aléatoire, comporter un nombre de bits similaire, être très grands, et être premiers.

Deuxièmement, on doit calculer le résultat de la multiplication des nombres "p" et "q", qu'on nomme "n".

Troisièmement, on doit déterminer $\phi(n)$. Pour ce faire, on doit effectuer le calcul (p - 1)(q - 1).

Quatrièmement, on doit déterminer un entier qu'on nommera "e". Cet entier doit être compris entre 1 et $\phi(n)$, donc $1 < e < \phi(n)$. L'entier "e" et $\phi(n)$ doivent être relativement premiers, c'est-à-dire qu'ils ne doivent partager aucun diviseur commun autre que 1.

L'entier "e" trouvé correspond à l'exposant de la clé publique. On peut par la suite trouver l'entier "d" correspondant à l'exposant de la clé privée, et qui est l'inverse multiplicatif du nombre e mod $(\phi(n))$. Pour ce faire, on doit déterminer un nombre "d" qui satisfait l'équation $de \equiv \text{mod}(\phi(n))$.

La clé publique correspond au modulo de "n", avec l'exposant "e". La clé privée correspond au modulo "n", avec l'exposant "d", qui doit être tenu secret.

Pour chiffrer le message "m", on effectue l'opération $c = m^e$ mod (n). Pour déchiffrer un message, on effectue l'opération $m = c^d$ mod (n).

La sécurité d'un tel système repose sur la confidentialité de "d". En effet, pour calculer "d", on doit posséder les nombres "p" et "q". Même si l'attaquant est en possession de "n", il ne peut retrouver les deux facteurs l'ayant formé, étant donné que la factorisation d'un très grand nombre est une tâche qui est actuellement impensable. À ce jour, le plus grand nombre factorisé est formé de 768 bits [Kleinjung *et al.*, 2010], et des clés RSA de 1024 bits sont habituellement utilisées, chaque bit supplémentaire augmentant d'environ un facteur 2 la complexité de la factorisation.

3.2.3 Cryptographie symétrique VS cryptographie asymétrique

On remarque que les cryptographies symétriques et asymétriques sont complémentaires. Une est performante et l'autre non (facteur de centaines de milliers), une requiert un échange sécurisé et l'autre non.

Ces deux types de cryptographie sont donc souvent utilisés ensemble. La cryptographie asymétrique est utilisée initialement pour s'échanger sécuritairement une clé symétrique, qui est ensuite utilisée durant le reste de la communication pour chiffrer et déchiffrer de manière performante les messages échangés.

Des systèmes utilisent cette combinaison particulière de système cryptographique afin de permettre à des entités de communiquer de l'information entre eux. Le procotole TLS,

abordé plus loin au chapitre 3.4.1 utilise entre autre ce principe. La section suivante est un autre exemple, qui se sert cette fois-ci des propriétés de la cryptographie asymétrique.

3.3 Infrastructure à clés publiques

Une infrastructure à clés publiques (PKI) est une architecture ayant pour objectif l'identification d'une entité face à une autre. Avant d'aborder le fonctionnement d'un tel système, il est important de définir les éléments suivants :

- certificats numériques ;
- autorité de certification ;
- autorité d'enregistrement.

3.3.1 Certificats numériques

Les certificats numériques sont des documents électroniques permettant de déterminer l'identité numérique d'un individu ou d'une entité quelconque. Ce type de certificat est considéré public, dans le sens où il peut librement être distribué sur Internet sans aucune mesure de protection, puisqu'il ne contient aucune information confidentielle.

Le ITU a normalisé le format des certificats ainsi que leur contenu avec la norme X.509 v3 :

1. version du certificat ;
2. numéro de série ;
3. algorithme d'identification ;
4. émetteur du certificat (autorité de certification (voir chapitre 3.3.2)) ;
5. période de validité du certificat ;
6. sujet (propriétaire du certificat, aussi appelé *Common Name*) ;
7. algorithme de clé publique ;
8. clé publique du sujet ;
9. algorithme de signature du certificat ;
10. signature chiffrée du certificat ;

L'objectif général d'un certificat numérique est de lier une clé publique unique à une entité (sujet). Bien qu'il soit possible de créer ou modifier un certificat de manière arbitraire,

celui-ci ne sera pas authentique étant donné la manière dont il doit être généré (chapitre 3.3.4). Un processus de validation de certificats numériques permet de valider leur authenticité (chapitre 3.3.5).

3.3.2 Autorité de certification

Une autorité de certification (CA) est responsable de créer et délivrer des certificats numériques. Elle atteste (certifie) que l'information présente dans un certificat qu'elle génère, en particulier la clé publique, appartient bel et bien au propriétaire (sujet). Elle est de plus chargée de la mise à jour de la liste des certificats, qu'ils soient nouveaux, expirés, modifiés ou révoqués.

La CA est le point central d'une PKI (*Public Key Infrastructure*), dans le sens où toute la sécurité du système repose sur cette entité, plus particulièrement sur sa clé privée. Comme il s'agit du *single point of failure* de l'architecture, il est important de tout mettre en oeuvre pour la protéger.

3.3.3 Autorité d'enregistrement

Cette entité s'occupe d'enrôler les nouveaux utilisateurs dans une PKI. Elle reçoit les demandes de certificat (CSR) et est responsable de les vérifier. Les demandes de certification doivent respecter le standard PKCS#10 [Nystrom et Kaliksi, 2000]. Il s'agit essentiellement du format et de la syntaxe des demandes.

Le processus de vérification est très important, il vise à valider le contenu du formulaire, à s'assurer que le demandeur possède bien la clé privée et la clé publique associées à la demande et à vérifier si le demandeur a bien l'autorisation de son organisation pour faire la demande de certificat.

Ce processus peut se faire de différentes manières : échanges de courriels, appels téléphoniques, enquêtes par des organismes tiers, etc. Si la demande est acceptée, elle est transférée au CA (chapitre 3.3.2). Le transfert de la demande n'est que partiel, seules les informations strictement indispensables lui sont envoyées. Il est à noter qu'en fonction de la répartition des autorités, un CA peut parfois jouer le rôle d'une autorité d'enregistrement.

3.3.4 Création d'un certificat numérique

En partant de ces principes, un CA est en mesure d'assurer l'authenticité d'un certificat numérique. Lorsque le CA produit un certificat, il génère une signature, par exemple avec

l'algorithme de hachage MD5, des éléments 1 à 9 présentés au chapitre 3.3.1. Le CA chiffre ensuite cette signature avec sa clé privée et l'ajoute à la fin du certificat (élément 10). Voici une représentation graphique de ce processus :

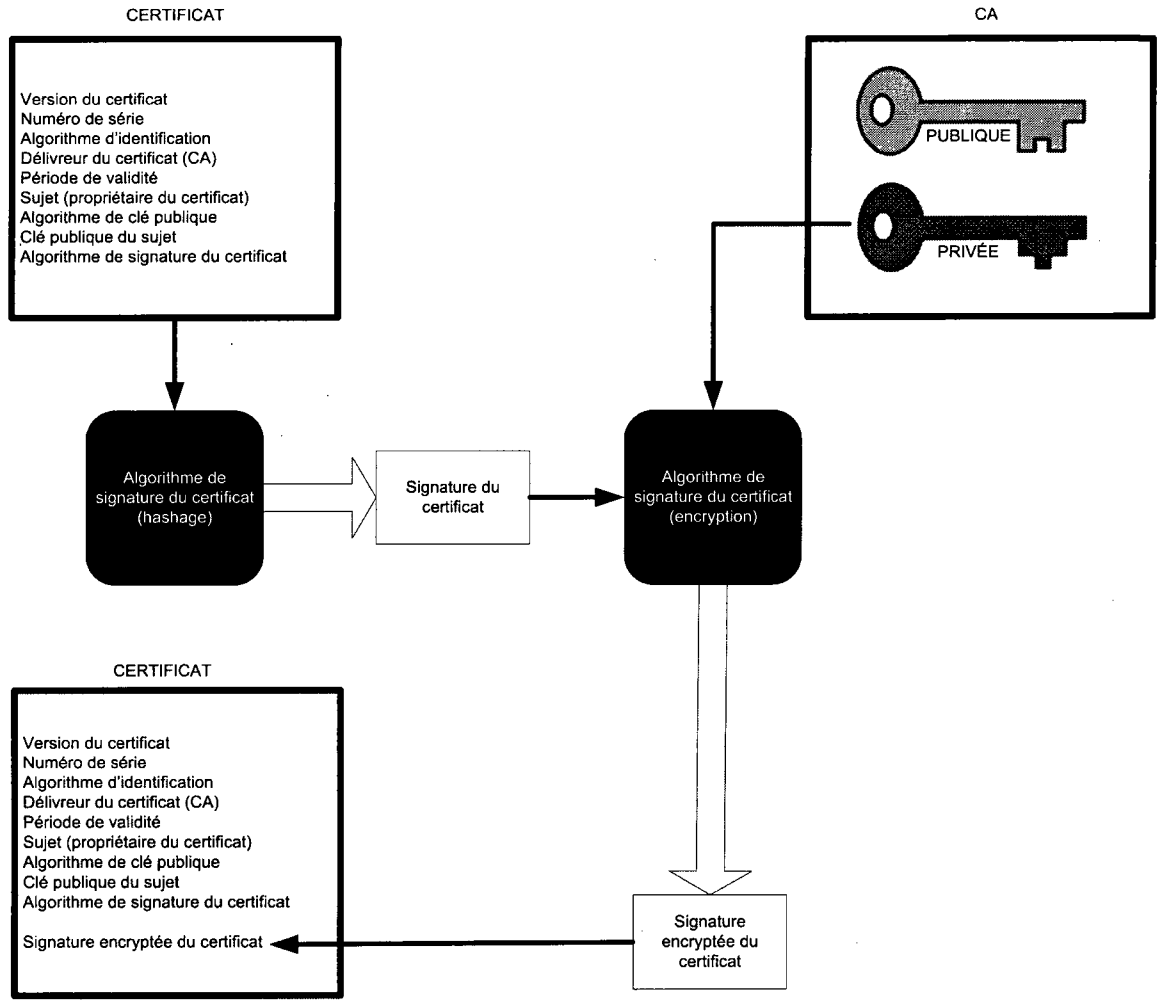


Figure 3.5 Processus de signature d'un certificat numérique.

Comme la signature a été chiffrée avec la clé privée du CA, cela signifie que personne d'autre n'aurait été en mesure de produire cette signature chiffrée du certificat (la clé privée du CA est connue seulement de celui-ci). Le processus de validation d'un certificat permet de s'assurer de l'authenticité d'un certificat, c'est-à-dire qu'il n'a pas été altéré et qu'il est certifié par l'autorité de certification qui l'a signé.

3.3.5 Validation d'un certificat numérique

Pour valider un certificat et ainsi s'assurer qu'il est véridique, une entité a besoin du certificat du CA l'ayant délivré/signé. Évidemment, l'entité désirant faire la validation doit avoir confiance en l'autorité de certification. Étant donné que le certificat à valider est signé à l'aide de la clé privée du CA, il est possible de déchiffrer la signature à l'aide de

la clé publique du CA, qui à l'opposé de sa clé privée, est disponible à tous (la clé publique est présente dans le certificat numérique du CA). En utilisant l'algorithme de signature du certificat (élément 9 du chapitre 3.3.1), il est possible de générer la signature du certificat et de la comparer à celle chiffrée par le CA. Voici une représentation graphique de ce processus :

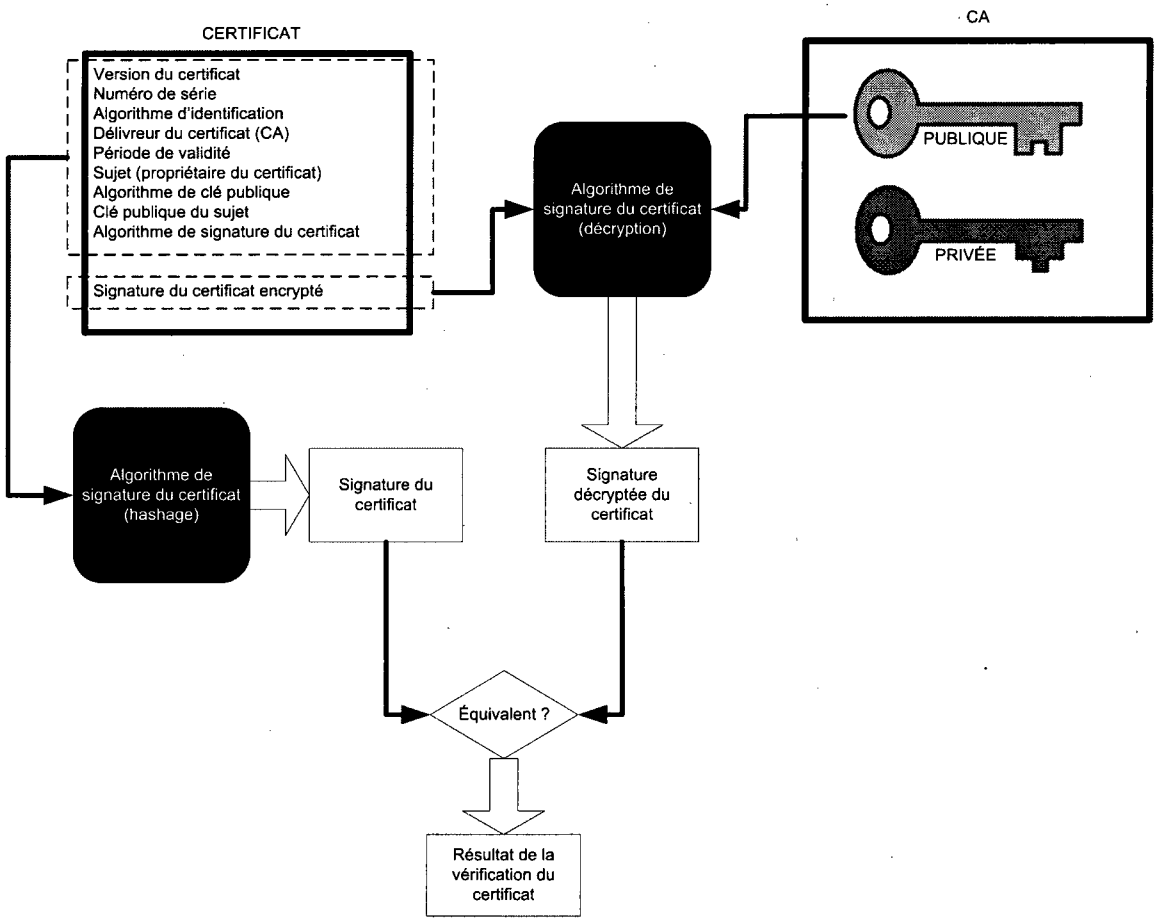


Figure 3.6 Processus de validation d'un certificat numérique.

Prenons le cas où quelqu'un voudrait modifier arbitrairement la clé publique d'un certificat (élément 8 du chapitre 3.3.1) afin de pouvoir l'utiliser en se faisant passer pour un organisme (élément 6 du chapitre 3.3.1). En modifiant une partie du certificat, ne serait-ce qu'un seul caractère, le résultat de l'opération de signature sera complètement différent. La comparaison entre cette signature et celle déchiffrée avec la clé publique du CA permettra de détecter cette tentative.

Si quelqu'un voulait modifier la signature chiffrée (élément 10 du chapitre 3.3.1) en plus de la clé publique du sujet (élément 8 du chapitre 3.3.1), de manière à ce que la signature

concorde avec l'opération de signature du certificat, il devrait tout de même la chiffrer avec une certaine clé. Comme la clé privée du CA est inconnue, la personne devrait en utiliser une autre. Par conséquent, lorsqu'il y aura validation du certificat, il ne pourra y avoir déchiffrement de la signature, car c'est la clé publique du CA présent dans son certificat qui sera utilisée. Donc, le processus de validation échouera.

3.3.6 Exemple de fonctionnement d'une infrastructure à clés publiques

Les PKI peuvent être utilisées dans plusieurs contextes. Prenons l'exemple du site web de la RBC Banque Royale. Initialement, cette entreprise a fait la demande de certificat à une RA (3.3.3). Cette autorité a ensuite fait une enquête pour s'assurer de l'identité de l'entreprise. Suite à cette vérification et à l'acceptation de la demande, la RA a transféré la demande à un CA (3.3.2), dans ce cas-ci VeriSign Trust Network. Le CA a par la suite émis le certificat et l'a remis à la RBC Banque Royale.

Depuis le site de la RBC Banque Royale, un usager peut établir une connexion sécurisée. Avant d'établir cette connexion, l'usager est en mesure de s'assurer que le site en question est sécuritaire, du moins que c'est bel et bien le site officiel de la compagnie et qu'il n'est pas victime d'hameçonnage (*phishing*). En effet, plusieurs navigateurs Internet sont des applications PKI-Compatible. De manière transparente, le fureteur Web récupèrera le certificat du site de l'entreprise et en effectuera la validation (voir chapitre 3.3.5). Cette opération, faite à l'aide du certificat du CA VeriSign, permettra de s'assurer que le certificat soit authentique, c'est-à-dire qu'il n'ait pas été modifié ou créé de toutes pièces.

Une portion du certificat contient l'information à propos du sujet (élément 6 du chapitre 3.3.1), notamment le *Common Name* (CN). Le CN représente généralement l'adresse du site Web en question. En vérifiant cette information avec le site sur lequel est connecté l'usager, le navigateur Internet pourra aviser l'utilisateur de toute anomalie.

Voici un extrait de l'information récupérée du certificat de la RBC Banque Royale depuis le navigateur Internet Firefox 3.0 :

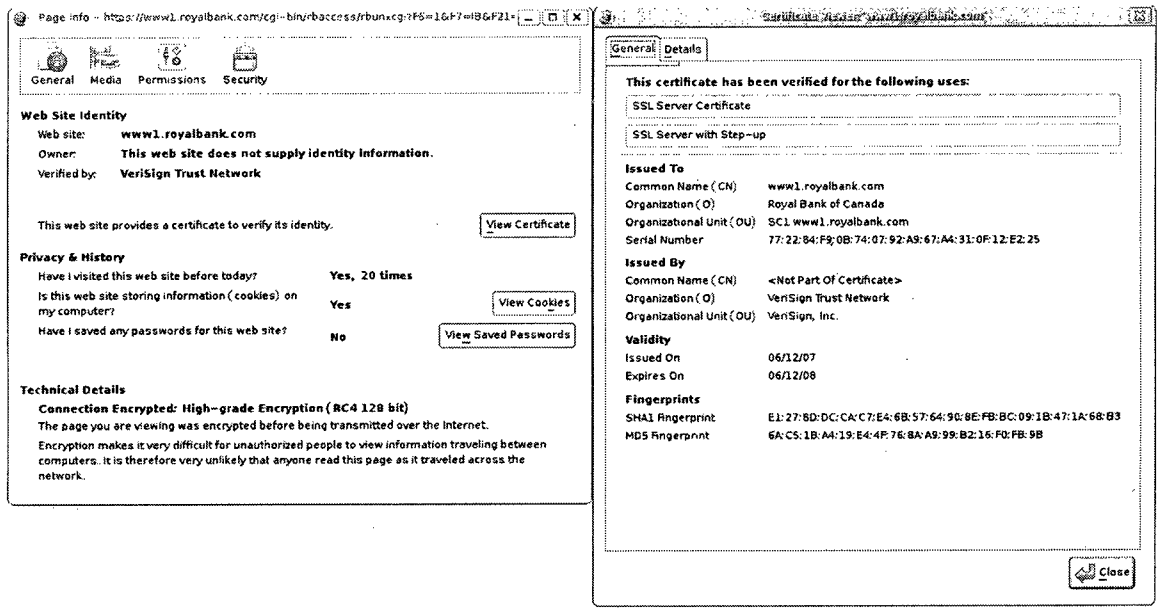


Figure 3.7 Capture d'écran de Firefox 3.0 montrant différentes informations du certificat numérique de la RBC Banque Royale.

En somme, une PKI permet à une entité (serveur, client, usager, organisation, etc.) de s'assurer de la validité du certificat numérique d'autrui. Comme ce certificat contient la clé publique de l'entité, il est par la suite possible de communiquer avec elle en chiffrant un message avec sa clé publique. Comme l'entité est la seule à posséder la clé privée associée à cette clé, elle seule sera en mesure de décoder l'information qui lui a été transmise.

3.4 Protocoles de communications

3.4.1 *Transport Layer Security*

Les échanges sur un réseau IP se doivent d'utiliser des protocoles et des standards établis. Parmi ceux-ci, TCP (*Transmission Control Protocol*), situé à la couche 4 (couche transport) du modèle de l'OSI, est l'un des plus utilisés dans la suite de protocoles Internet. Le type de transmission offert par TCP ne fournit par contre aucune mesure de protection des données. Ce protocole est défini par le RFC 793.

Le protocole TLS, situé à la couche 6 (couche présentation) du modèle de l'OSI, est un protocole cryptographique généralement implémenté au-dessus de n'importe quel protocole de la couche transport du modèle de l'OSI, dont le protocole TCP. Il permet d'encapsuler et de sécuriser les données des protocoles de la couche 7 (couche application) du modèle de l'OSI, comme HTTP, FTP, SMTP et SIP. Par exemple, lorsque HTTP est utilisé de

concert avec TLS, on parle alors du protocole HTTPS, qui est grandement utilisé pour l'accès sécuritaire à des sites web. Le protocole TLS permet plusieurs fonctionnalités reliées à la sécurisation d'une communication :

- l'authentification d'un serveur ;
- l'authentification optionnelle d'un client ;
- le chiffrement/déchiffrement de messages ;
- l'intégrité des messages.

L'établissement d'une connexion sécurisée entre deux entités avec TLS se produit à l'aide d'une procédure appelée poignée de main (*handshake*). Ce *handshake* peut se résumer en trois grandes étapes :

- négociation des algorithmes cryptographiques ;
- échange des clés et authentification des entités ;
- chiffrement symétrique et authentification des messages.

La négociation des algorithmes cryptographiques permet aux deux entités de s'entendre sur des jeux de chiffrement à utiliser lorsque la communication sécurisée débutera. Comme les deux entités ne supportent pas nécessairement les mêmes, cette négociation permet la sélection des jeux de chiffrement communs les plus sécuritaires.

L'échange des clés se fait généralement à l'aide d'algorithmes cryptographiques asymétriques (voir chapitre 3.2.2). Cet échange, combiné à la génération de nombres aléatoires par les deux entités, permet de générer un nombre commun partagé par les deux parties, que l'on nomme *Master Secret*. Ce secret est par la suite divisé pour être utilisé comme clé symétrique (voir chapitre 3.2.1) pour le chiffrement et le déchiffrement des messages échangés. La validation de l'identité des entités quant à elle est possible grâce à l'utilisation de certificats numériques (voir chapitre 3.3.1), échangés en même temps que les clés.

Le *handshake* TLS est un processus complexe. L'explication exhaustive de celui-ci ne sera pas donnée dans le présent mémoire, mais est disponible dans le RFC 5246 [Dierks et Rescola, 2008]. Voici néanmoins un aperçu du *handshake* complet lors d'une authentification serveur et client.

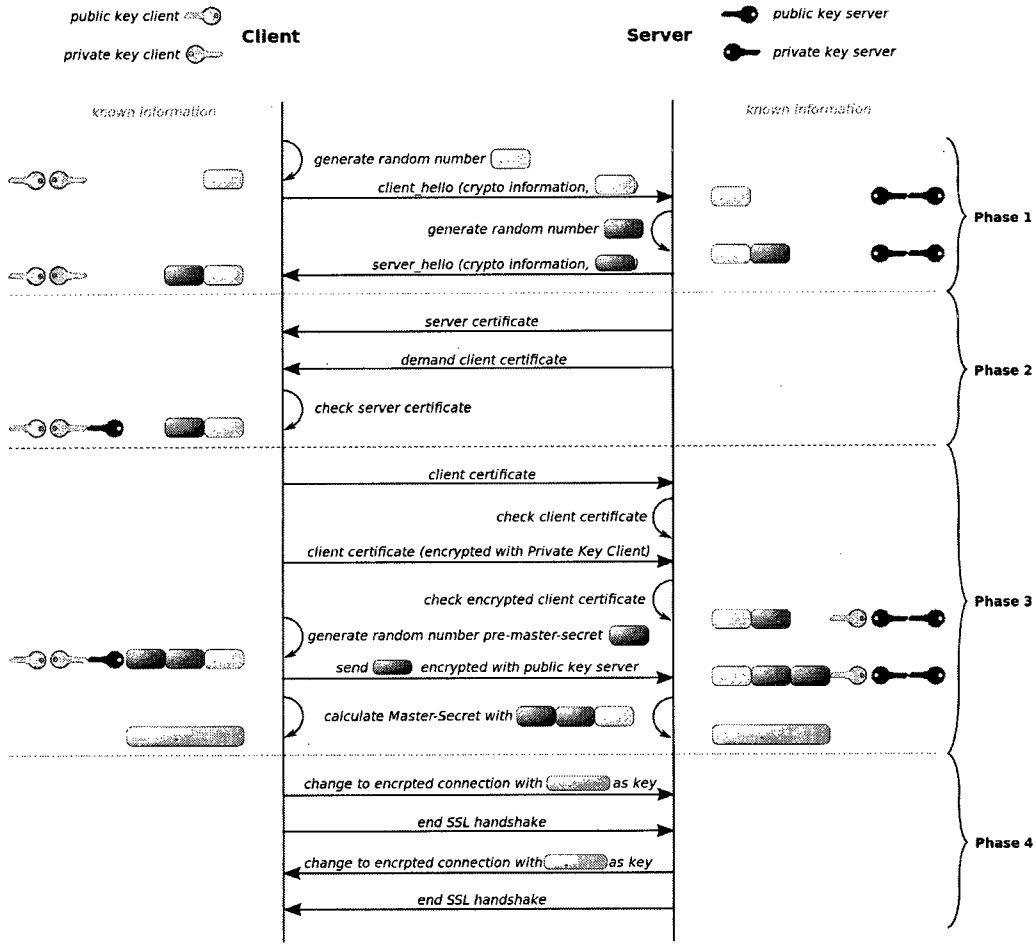


Figure 3.8 Description détaillée du *handshake* TLS lors d'une authentification mutuelle.

En somme, le protocole TLS permet de sécuriser les communications TCP entre deux entités. Un *handshake* entre les deux entités voulant communiquer permet la sélection d'algorithmes de cryptographie, la validation de l'identité d'autrui, ainsi que la génération d'une clé symétrique utilisée pour chiffrer les communications. Le protocole permet une protection contre l'écoute d'une communication privée par un tiers, contre la modification par un tiers des messages envoyés entre les deux entités, ainsi que contre la création complète de faux messages provenant d'un tiers.

3.4.2 *Real-time Transport Protocol / Secure Real-time Transport Protocol*

Le protocole RTP (*Real-time Transport Protocol*), défini par le RFC 3550, est situé à la couche 7 (couche application) du modèle de l'OSI. Il définit un standard pour le format des paquets IP utilisés dans la transmission vidéo et audio. Ce protocole est largement utilisé dans les systèmes de téléphonie, de vidéoconférence et de *streaming* multimédia. Le protocole RTP permet le transport des données multimédias uniquement ; il n'offre pas de contrôle sur la qualité de la transmission. C'est pourquoi il est utilisé de concert avec le protocole RTCP, qui lui, gère le contrôle du transfert et fournit des statistiques sur la qualité de la transmission.

Lors de l'établissement d'une transmission multimédia, avec *streams* audios et vidéos, une session est établie pour chacun des *streams*. Chacune des sessions est composée d'une adresse IP avec une paire de ports : un pour le protocole RTP, l'autre pour le protocole RTCP. Ces protocoles ne permettent toutefois pas la négociation des différents paramètres nécessaires à l'établissement d'une transmission. L'utilisation d'un protocole tiers comme RTSP ou SIP (voir chapitre 3.4.3) est nécessaire pour faire la gestion d'une session RTP. La négociation des paramètres quant à elle peut être exécutée à l'aide du protocole SDP.

Le protocole RTP ne fournit aucune mesure de protection sur la transmission des données. Pour assurer cette protection, on doit avoir recours au protocole SRTP. Ce protocole, défini par le RFC 3711 [Baugher *et al.*, 2004], offre les mesures de protection suivantes :

- chiffrement des messages ;
- authentification des messages ;
- intégrité des messages ;
- protection contre les *Replay Attacks*.

Le chiffrement des communications se fait généralement avec l'algorithme de cryptographie symétrique AES. Le protocole SRTP n'offre toutefois pas le partage sécuritaire de l'information nécessaire à la génération de la clé symétrique entre les deux parties, et qui est utilisée par le protocole AES. L'utilisation d'un protocole tiers comme SDES, ZRTP, ou MIKEY est nécessaire.

En somme, le protocole RTP est utilisé pour le transfert de données multimédias sur un réseau IP. Comme celui-ci ne gère pas la sécurité des transmissions, l'utilisation du protocole SRTP est requise lorsque nécessaire.

3.4.3 *Session Initiation Protocol*

Comme mentionné précédemment, un protocole comme RTP ne peut être utilisé directement. Avant l'établissement d'une transmission multimédia, il doit se produire ce qu'on appelle une signalisation, pour permettre la négociation de paramètres requis au démarrage de la transmission.

Le protocole SIP, défini par le RFC 3261 et situé à la couche application du modèle de l'OSI, est un protocole de signalisation utilisé pour la gestion de sessions multimédias (vidéoconférence, *streaming* multimédia, messagerie instantanée, jeux en ligne, téléphonie (VoIP), etc.). En aucun cas SIP ne transporte les données utiles d'une conférence comme de l'audio ou du vidéo.

SIP permet de créer, terminer ou modifier une session. Cela peut comprendre le changement de ports ou d'adresses IP, l'ajout de participants supplémentaires ainsi que l'ajout ou la suppression de *streams* à une session en cours. Lors de la signalisation, SIP se charge en plus de l'authentification et de la localisation des participants.

SIP repose principalement sur trois types d'entités :

- *user agent* ;
- *registrar* ;
- *proxy.*

Les *user agents* représentent les entités logiques permettant d'envoyer ou de recevoir des messages SIP. Les téléphones SIP, les logiciels de téléphonie sur IP, les ordinateurs, les PDA ou les passerelles SIP peuvent agir en tant que *user agent.*

Le *registrar* est une entité gérant les requêtes de type *REGISTER.* Son rôle est de fournir un service de localisation en jumelant une ou plusieurs adresses IP à l'URI d'un *user agent.*

Le *proxy* SIP sert d'intermédiaire entre deux *user agents* lorsque ceux-ci ne connaissent pas leur localisation respective. Comme les associations *user agent* / adresses IP sont enregistrées dans la base de données d'un *registrar*, un *proxy* peut contacter celui-ci pour récupérer l'adresse qui lui est inconnue, et ainsi initier une nouvelle transaction avec le destinataire pour lui acheminer la requête initiale. Le *proxy* ne fait que transmettre les messages SIP pour créer, gérer et fermer une session. Lorsque la transmission est établie entre les deux *user agents* voulant communiquer, les flux multimédias sont transmis directement entre les *user agents*, sans passer par le *proxy.*

En somme, le protocole SIP permet la création, la gestion et la terminaison de sessions multimédias. Le protocole n'est pas responsable du transport des données utiles des flux multimédias, il ne fait que la signalisation entre les entités voulant communiquer, en permettant une négociation des paramètres de sessions avec le protocole SDP. Le protocole TLS, présenté présédemment, peut de plus être combiné au procotole SIP afin de permettre un échange des clés de cryptographie, servant par exemple à échanger de manière sécurisée les paramètres SDP nécessaires à la négociation de la session multimédia. On parlera alors ici de SIPS.

3.5 Applications de conférence multimédia / télémédecine

Il existe sur le marché une très grande quantité d'appareils de télémédecine et de logiciels de vidéoconférence. Leur quantité rend impossible leur étude complète dans le cadre de ce mémoire. Voici néanmoins trois produits ainsi que leurs fonctionnalités principales, qui permettront de soulever une problématique qui sera abordée dans la conclusion de ce chapitre.

3.5.1 AFHCAN

La compagnie AFHCAN propose plusieurs produits de télémédecine. Parmi ceux-ci, le ECG 12-Lead est un appareil permettant la capture d'électrocardiogrammes sur un patient. L'appareil permet facilement la création, la sauvegarde et la transmission d'électrocardiogrammes à un ordinateur.

Figure 3.9 ECG 12-Lead (www.afhcan.org)

Plusieurs autres types d'appareil de mesure sont développés par cette compagnie. Une station transportable, le *Telemedicine Cart Version 3*, permet d'utiliser ces appareils et d'assurer la connectivité avec le système principal. Il est doté de connectivité aux réseaux

sans-fil afin de permettre la communication entre les stations. Ce système est utile dans les centres hospitaliers, mais ne peut être utilisé à l'extérieur, comme dans des ambulances, étant donné l'espace requis par la station.

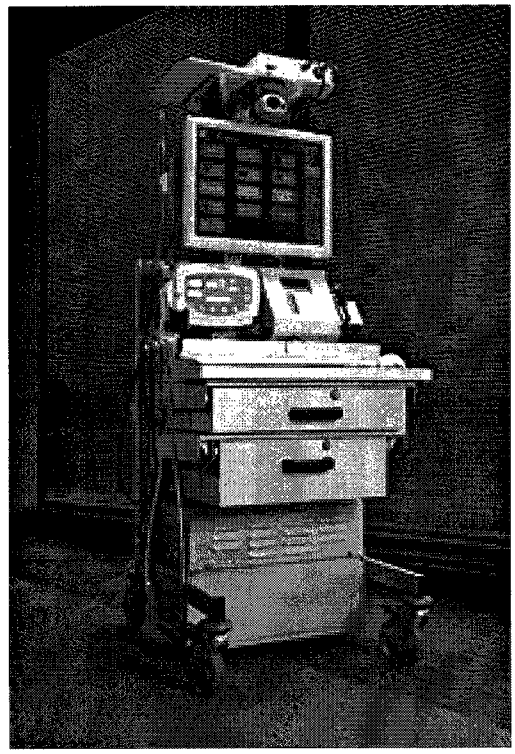


Figure 3.10 *Telemedicine Cart Version 3* (www.afhcan.org)

3.5.2 Audisoft

Audisoft, une compagnie québécoise, offre le *Frontline Communicator*. Ce système permet la communication bidirectionnelle audio et vidéo sur un réseau de communication IP (*Mobile/3G*, *WiFi*, *mesh*, *satellite*). Il permet l'enregistrement et le transfert de sessions. Le système peut être utilisé dans diverses situations comme l'assistance à distance, l'entraînement à distance, la transmission d'informations critiques, l'enregistrement d'investigations et de preuves, etc.

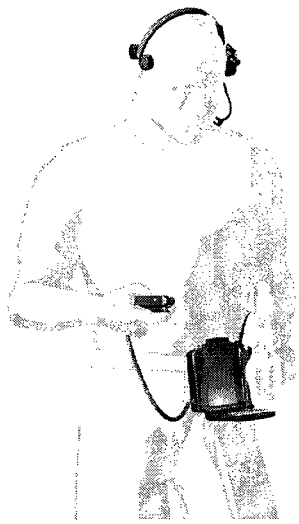


Figure 3.11 *Frontline Communicator* (www.audisoft.net)

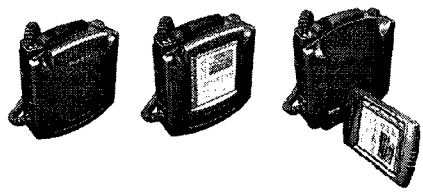


Figure 3.12 *Frontline Communicator* (www.audisoft.net)

Ce système n'est pas donc pas réservé à un usage médical. Par contre, il possède un ensemble de fonctionnalités souhaitables pour une application de télémédecine comme la consultation de patients à distance, le diagnostic temps-réel ainsi que la mobilité.

3.5.3 Viterion

La compagnie Viterion offre une gamme de produits de télémédecine pour les spécialistes de la santé et les patients à domicile. Leurs types d'appareils sont destinés à l'utilisation à domicile (ou dans de petites cliniques) par les patients, afin de pouvoir faire des suivis avec le personnel médical d'un hôpital.

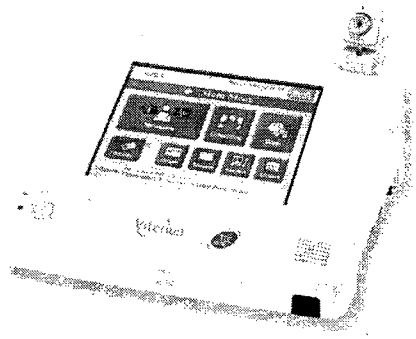


Figure 3.13 *Viterion 500* (www.viterion.com)

L'appareil, doté de connectivité sur un réseau IP, a été conçu pour être facile d'utilisation pour les patients. L'interface graphique comporte de grands et visibles dessins, icônes et claviers, le rendant facile d'approche.

Les données sont sécurisées et le système répond aux critères du HIPAA.

3.5.4 Conclusion

On remarque qu'il existe plusieurs types d'appareils et de logiciels. Chacun a leurs fonctionalités, leurs avantages et leurs défauts. Le produit idéal devrait remplir les conditions suivantes :

- facile d'utilisation ;
- mobile et compact ;
- permets de communiquer sur un réseau IP sans-fil ;
- permets d'établir des conférences multimédias à plusieurs ;
- permets les échanges de données physiologiques de patients en temps réels ;
- possède des mesures de sécurité pour protéger l'information confidentielle ;
- permets une authentification des usagers pour des raisons médicales et légales ;
- conçu pour les spécialistes de la santé en situation d'urgence.

Les produits disponibles ne remplissent que quelques une de ces conditions. Ils sont généralement complémentaires entre eux, mais aucun ne possède tous ces critères à la fois. Le projet MédicIP et ses travaux futurs visent à se positionner sur le marché avec une solution respectant tous ces requis.

3.6 Résumé

Une analyse de l'état de l'art a permis de cibler des technologies qui pourront corriger les failles de sécurité du prototype MédicIP. Voici le résumé des failles de sécurité du prototype ainsi que les solutions envisagées pour les corriger.

Failles de sécurité identifiées	Solutions envisagées	Solutions implémentées
Communication non sécurisée lors du processus d'authentification d'un usager.	Création d'une infrastructure à clés publiques permettant la validation de l'identité des entités, ainsi que la création d'un canal de communication sécurisé.	
Signalisation SIP non sécuritaire lors de la création d'une conférence.	Utilisation du protocole TLS pour effectuer les échanges SIP de manière sécuritaire.	
Communication audio non sécuritaire.	Utilisation du protocole sécurisé SRTP.	
Échange d'électrocardiogrammes non sécuritaire.	Utilisation du protocole TLS pour effectuer les échanges de données de manière sécuritaire.	
Entreposage non sécurisé du carnet d'adresses.	Utiliser un carnet d'adresses centralisé.	

Figure 3.14 Failles de sécurité du prototype MédicIP.

CHAPITRE 4
ARCHITECTURE LOGICIELLE

Afin de répondre à la question de recherche soulevée au chapitre 1, le projet SécureMédic a vu le jour. Ce projet de recherche consiste en l'ajout de modules logiciels au prototype MédicIP ainsi qu'en la création d'un tout nouveau serveur d'authentification. SécureMédic se veut par conséquent un moyen de répondre aux objectifs fixés au chapitre 2 :

- développer un système d'authentification usager facile d'utilisation et sécuritaire ;
- permettre des échanges sécurisés (voix et données) entre les différents utilisateurs d'une conférence multimédia ;
- entreposer de manière sécuritaire les données sensibles utilisées par l'application de télémédecine.

La description de l'architecture développée dans le projet SécureMédic sera présentée en trois parties :

1. architecture globale ;
2. architecture du serveur d'authentification ;
3. architecture du client SécureMédic.

Le chapitre suivant (chapitre 5) se voudra une explication des différents processus de sécurisation auxquels les modules présentés dans le présent chapitre participent.

4.1 Architecture - Globale

Cette section met en évidence les modifications du prototype MédicIP engendrées par les ajouts du projet SécureMédic. Il s'agit d'une brève introduction technique au projet de recherche permettant de se familiariser avec les éléments de base de l'architecture, afin d'avoir une vue globale du projet.

4.1.1 Schémas

Voici le schéma des modules globaux présents :

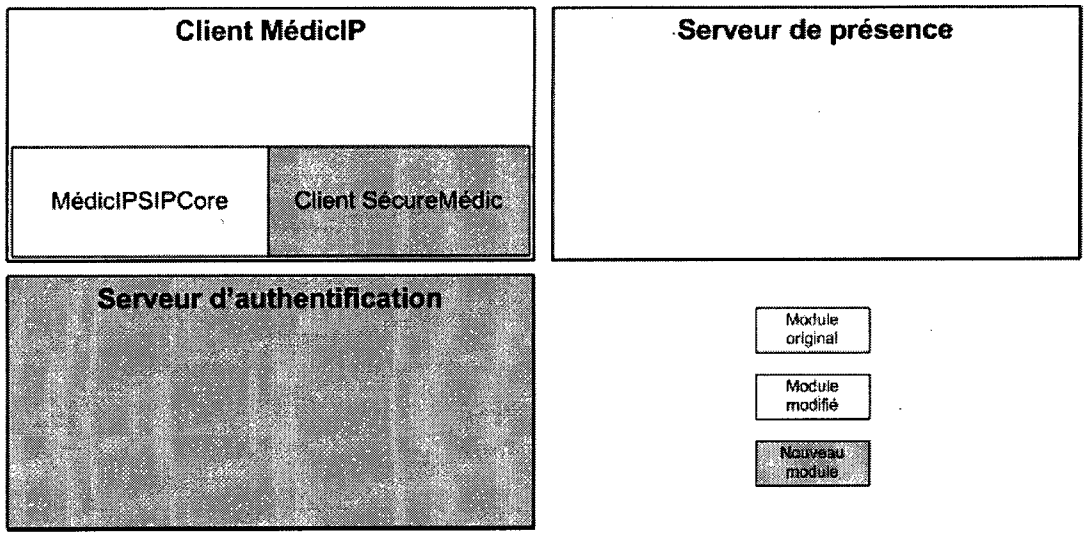


Figure 4.1 Impacts du projet SécureMédic.

Ces ajouts et modifications ont permis l'intégration d'une infrastructure à clés publiques (*PKI*) dans le projet MédicIP. Cette *PKI* permet d'attaquer directement le premier objectif fixé au chapitre 2 : l'implémentation d'un système d'authentification sécurisé pour les usagers voulant utiliser l'application MédicIP.

4.1.2 Entités

Afin de bien comprendre les différents éléments en jeu dans la nouvelle architecture du projet, il est important de définir les entités de base suivantes :

- usager ;
- client ;
- serveur d'authentification.

Usager

Les usagers représentent les utilisateurs de l'application de télémédecine MédicIP.

Client

Les clients, aussi appelés clients MédicIP, représentent l'application de télémédecine utilisée par les usagers. L'utilisation d'un client n'est pas restreinte à un seul usager et vice versa. Le client est décrit au chapitre 2.2.1.

Serveur d'authentification

Le serveur d'authentification est unique et représente l'entité permettant l'authentification des usagers avec un client. Il est le point central du processus d'authentification et agit en tant que *certificate authority* dans l'infrastructure à clés publiques (voir chapitre 3.3.2). Il s'agit du coeur du projet SécureMédic.

4.1.3 Clés

L'infrastructure à clés publiques du projet MédicIP comporte deux types de paire de clés :

- paire de clés de type client ;
- paire de clés de type serveur.

Dans les deux cas, il s'agit de paires de clés de type RSA de 2048 bits, taille recommandé par RSA Laboratories [Kaliski, 2003]. Les clés privées sont chiffrées à l'aide d'un mot de passe et de l'algorithme de cryptographie symétrique AES256. Chiffrer les clés privées de cette façon permet de les entreposer dans un système de fichiers sans avoir à se préoccuper de leur confidentialité.

Clés de type client

Chaque client possède sa propre paire de clés unique. Le mot de passe servant à chiffrer la clé privée est déterminé par l'administrateur du système et celui-ci est incorporé (*hardcode*) à même le code logiciel du client.

Clés du serveur d'authentification

Cette paire de clés, aussi appelée paire de clés *root*, est unique et n'existe que dans le serveur d'authentification. De la même manière, le mot de passe utilisé pour chiffrer cette clé est incorporé (*hardcode*) à même le code logiciel du serveur.

Il est à noter que le serveur d'authentification est le point central de l'infrastructure à clés publiques. Ceci implique que toute la sécurité de l'infrastructure dépend de la protection entourant la clé privée *root*. Dans le cas du serveur d'authentification, tant la clé privée que son mot de passe sont restreints par des privilèges d'accès au niveau des fichiers du système d'exploitation.

4.1.4 Certificats numériques

L'infrastructure à clés publiques du projet MédicIP comprend deux types de certificats numériques :

- certificat de type client ;
- certificat de type serveur.

Certificat de type client

Comme chaque client possède sa propre paire de clés, chaque client possède son propre certificat numérique. Outre la clé publique associée au certificat, ce qui distingue chaque certificat de type client est le *common name*, qui correspond à l'adresse IP statique de l'hôte sur lequel le client est exécuté.

Chaque certificat de type client est signé avec la clé privée du serveur d'authentification, qui fait office de *certificate authority*.

La raison qui motive l'utilisation d'une adresse IP statique est le fait que le prototole "SIP *over* TLS" effectue une validation du certificat de type client d'après l'adresse IP de l'entité. Il s'agit donc d'une limitation dans le sens où une utilisation du système sur un réseau sans-fil serait problématique, étant donné l'assignation des adresses IP de manière dynamique. Le chapitre 7.3.4 abordera des travaux futurs concernant cette limitation. L'utilisation d'adresses IP statiques permet néanmoins la création d'une preuve de concept suffisante.

Certificat de type serveur

Le certificat de type serveur est aussi appelé *root certificate* ou *trusted certificate*. Comme ce certificat est utilisé pour valider l'identité du serveur d'authentification, il est présent sur chacun des clients ainsi que sur le serveur d'authentification.

4.2 Architecture - Serveur d'authentification

4.2.1 Rôle

Comme mentionné dans le chapitre 4.1.2, le serveur d'authentification a pour objectif de permettre à un utilisateur du logiciel MédicIP de s'authentifier afin d'utiliser l'application de télémédecine.

Ce chapitre présentera la description technique des composantes du serveur d'authentification tandis que le chapitre 5 se concentrera sur le processus d'authentification, où ces composantes sont impliquées.

4.2.2 Dépendances logicielles

Le serveur d'authentification est une application avec plusieurs fils d'exécution (*multi-thread*) programmée en C++ permettant de gérer les requêtes venant des clients MédicIP. Cette application a été conçue pour fonctionner sur un système d'exploitation Linux. Plus précisément, la distribution Linux Ubuntu 9.04 utilisant le Kernel Linux 2.6.28-13 a été utilisée pour le développement du serveur d'authentification.

Plusieurs dépendances logicielles sont présentes afin de permettre le fonctionnement du serveur d'authentification :

- OpenSSL ;
- MySQL ;
- libmysqlclient ;
- Connector C++.

OpenSSL

OpenSSL est une implémentation code libre (*open source*) des protocoles SSL et TLS ainsi que de plusieurs algorithmes de cryptographie. Dans le cadre du projet SécureMédic, la librairie OpenSSL a été utilisée pour les raisons suivantes :

- génération de clés RSA ;
- génération et validation de certificats numériques ;
- établissement de communications sécurisées ;
- hachage de données ;
- chiffrement / déchiffrement de données.

La version 0.9.8k de OpenSSL a été utilisée lors du développement du serveur d'authentification.

MySQL

MySQL est une base de données relationnelle disponible sous les termes de la licence *GNU General Public License*. Les détails sur l'utilisation de la base de données par le serveur d'authentification seront présentés au chapitre 4.2.4.

La version 5.0.77-linux-i686-glibc23 a été utilisée lors du développement du serveur d'authentification.

libmysqlclient

libmysqlclient est un paquet (*package*) incluant des librairies de développement ainsi que des fichiers d'entête (*header*). Ce *package* est nécessaire au fonctionnement de la librairie Connector C++.

La version utilisée lors du développement est libmysqlclient15-dev (5.1.30).

Connector C++

Connector C++ est une librairie fournissant un API permettant de communiquer avec une base de données MySQL depuis du code logiciel écrit en C++. Une des particularités de cette librairie est qu'elle permet l'utilisation de ce qu'on appelle les *prepared statements*. Ce type de requêtes est utilisé pour procurer des gains en performance lors de requêtes SQL, ainsi que pour offrir une protection contre les attaques de type injection SQL.

4.2.3 Modules

Le diagramme de classes suivant présente les différents modules composant le serveur d'authentification :

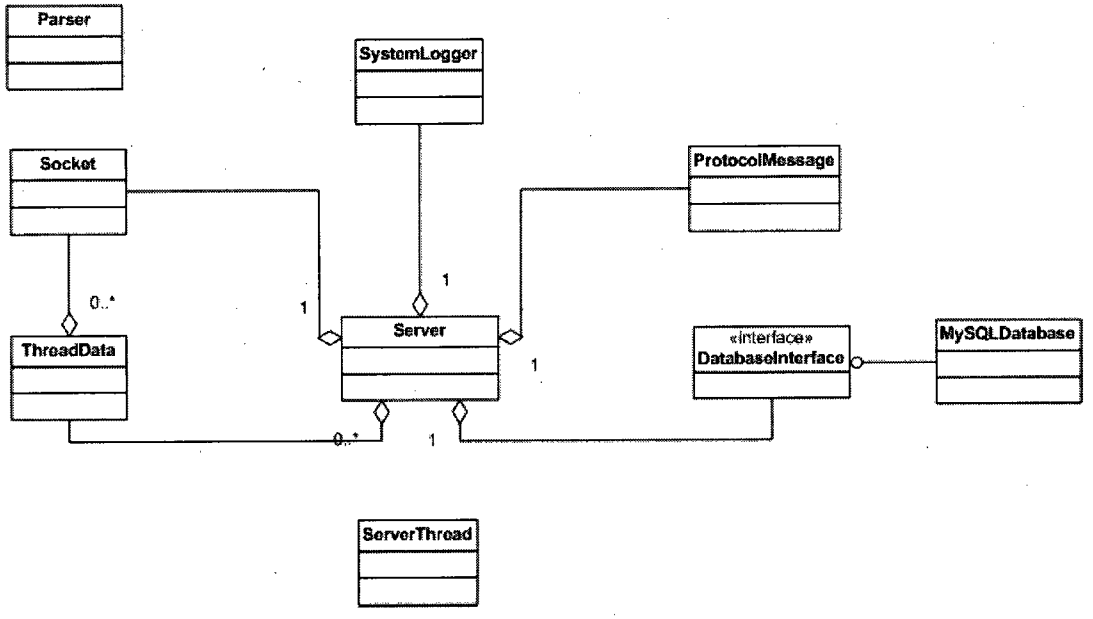


Figure 4.2 Diagramme de classes du serveur d'authentification.

La classe Server est le module principal du serveur d'authentification. Avant d'entreprendre sa description, les modules gravitant autour de celui-ci seront abordés.

Socket

La classe Socket, comme son nom l'indique, représente un socket TCP. Cette classe a été conçue spécifiquement pour le serveur d'authentification et permet les opérations de bases sur des sockets.

SystemLogger

La classe SystemLogger est un module permettant au serveur d'authentification d'écrire des événements dans un fichier de *logs*. Elle est conçue selon le patron de conception *Singleton*, permettant ainsi à n'importe quel module du serveur d'authentification d'enregistrer des événements. Il est important de spécifier que le type d'événements géré par ce module ne consiste pas en des événements reliés au fonctionnement normal de l'application, comme des authentifications réussies ou échouées. Il sert plutôt à signaler des événements internes à l'application, majoritairement des erreurs fatales qui empêchent le démarrage de l'application. Voici quelques types d'erreurs classiques :

- fichier de configuration manquant ;
- erreur de connexion à la base de données ;
- erreur dans le chargement des jeux de chiffrement supportés ;
- erreur dans le chargement de certificats ou de clés.

ProtocolMessage

Les échanges entre les clients et le serveur respectent un protocole de communication, c'est-à-dire un format de message permettant aux entités de s'échanger de l'information. Il existe six types de message pouvant être échangés entre un client et le serveur :

- demande d'authentification ;
- authentification échouée ;
- authentification réussie ;
- demande de changement de mot de passe ;
- changement de mot de passe réussi ;
- changement de mot de passe échoué.

Chacun de ces types de message possède un code qui lui est spécifique. Ce code est utilisé dans la convention de format suivante :

Le module ProtocolMessage permet d'utiliser le protocole de communication en faisant abstraction de tout ce qui concerne le format et la construction des messages.

Par exemple, un développeur voulant construire un message de type "demande d'authentification" n'a qu'à appeler la fonction du module ProtocolMessage prévue à cet effet, en

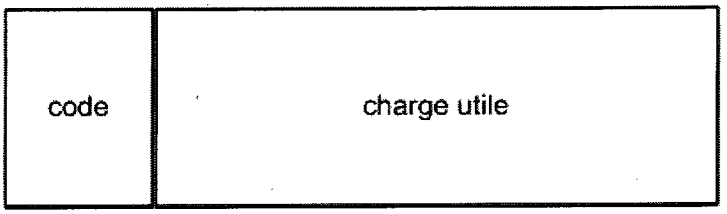


Figure 4.3 Format des messages utilisé dans le protocole de communication.

lui passant les paramètres nécessaires (nom d'usager et mot de passe). La classe produit en sortie un message pouvant être envoyé directement à l'autre entité.

Lors de la réception du message, il suffit d'appeler certaines fonctions du module afin d'en extraire le code ainsi que les données de sa charge utile.

DatabaseInterface et MySQLDatabase

Le serveur d'authentification utilise une base de données MySQL (voir chapitre 4.2.4). Deux classes permettent l'utilisation de celle-ci : DatabaseInterface et MySQLDatabase.

Le module de base de données a été conçu de manière à supporter n'importe quelle base de données. Cela est possible grâce à l'utilisation du patron de conception *Strategy*. Ce patron offre la possibilité de changer de base de données sans affecter le code du serveur d'authentification. La seule restriction est que la nouvelle classe gérant la base de données implémente l'interface DatabaseInterface.

ServerThread

La classe ServerThread est quelque peu particulière ; elle ne contient que les points d'entrées des différents fils d'exécution (*threads*) existant dans le serveur d'authentification (voir chapitre 4.2.5). Ce regroupement de fonctions sous la forme d'une classe a pour objectif une meilleure lisibilité du code logiciel, en déchargeant la classe principale (Server) de cette portion de code.

L'extraction de ces fonctions de la classe principale est possible grâce à l'utilisation d'une classe permettant la gestion des fils d'exécution actifs : ThreadData. Ces points d'entrée requièrent un objet de type ThreadData, qui contient toute l'information reliée à un fil d'exécution.

ThreadData

Comme mentionné précédemment, le serveur d'authentification est une application *multi-thread*. Afin d'en faciliter la gestion, la classe principale Server possède une liste d'objets de type ThreadData.

La classe ThreadData est en fait un regroupement de toutes les informations relatives à un fil d'exécution. Parmi celles-ci, on retrouve entre autre :

- le socket client en cours d'utilisation ;
- le contexte SSL (OpenSSL) en cours d'utilisation ;
- l'identificateur du fil d'exécution (PID) ;
- les informations relatives à l'usager en cours d'authentification (nom d'usager, mot de passe) ;
- l'état du processus d'authentification ;
- le compteur (*timer*) du fil d'exécution (discuté au chapitre 4.2.5) ;

Lorsqu'un fil d'exécution est lancé, via la classe ServerThread, un objet de type ThreadData lui étant associé lui est passé en argument (la classe Server en garde une référence). Le fil d'exécution peut ainsi mettre à jour ou récupérer à n'importe quel moment les informations nécessaires le concernant.

Parser

La classe Parser est utilisée uniquement lors du démarrage du serveur d'authentification. Cette classe a pour rôle la lecture et l'extraction des paramètres du fichier de configuration du serveur d'authentification.

Parmi les informations contenues dans le fichier de configuration, on retrouve :

- le port d'écoute du serveur ;
- les noms de fichier et de répertoire de la paire de clés de type serveur ;
- le mot de passe de la clé privée de type serveur ;
- le nom du fichier et de répertoire du certificat numérique de type serveur ;
- le jeu de chiffrement utilisé par le serveur ;
- le nombre maximal de fils d'exécution (voir chapitre 4.2.5) ;
- la valeur maximale du compteur **nettoyeur** (voir chapitre 4.2.5) ;
- la valeur minimale du compteur **nettoyeur** (voir chapitre 4.2.5) ;
- la valeur maximale du compteur individuel (voir chapitre 4.2.5) ;
- le nom de la base de données ;
- le nom d'utilisateur de la base de données ;
- le mot de passe de la base de données.

Comme ce fichier de configuration contient des informations sensibles, les privilèges de celui-ci permettent uniquement à l'utilisateur *root* de le consulter / modifier.

4.2.4 Base de données

Le serveur d'authentification se sert d'une base de données afin d'entreposer l'information nécessaire à l'authentification des usagers. Trois tables sont nécessaires pour effectuer cette tâche :

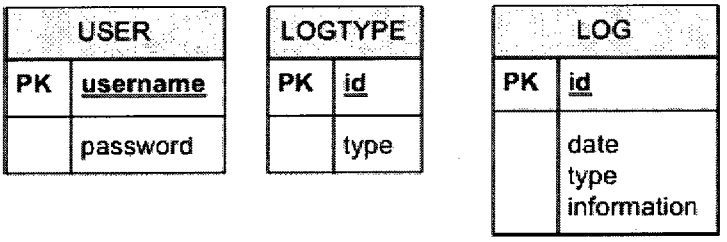


Figure 4.4 Tables utilisées dans la base de données du serveur d'authentification.

La table USER est utilisée pour entreposer les noms d'usager ainsi que leurs mots de passe. Les mots de passe sont entreposés sous le format de hachage SHA512 afin d'en assurer la confidentialité.

La table LOGTYPE contient les types d'événements pouvant être répertoriés par le serveur d'authentication.

La table LOG contient l'historique des événements s'étant produits. La date, le type d'événement ainsi que de l'information optionnelle font partie des données représentant un événement.

Il existe sept types d'événements pouvant être ajoutés dans la table LOG :

- authentification d'un usager échouée ;
- authentification d'un usager réussie ;
- fil d'exécution tué par le **nettoyeur** (discuté au chapitre 4.2.5) ;
- expiration d'un fil d'exécution (discuté au chapitre 4.2.5) ;
- nombre de fils d'exécution maximal atteint ;
- mise à jour d'un mot de passe échouée ;
- mise à jour d'un mot de passe réussie ;

Afin d'augmenter la sécurité des données entreposées dans la base de données, cette dernière est configurée pour être uniquement accessible localement, via l'adresse IP 127.0.0.1 (*localhost*).

De plus, les applications locales voulant se connecter à la base de données doivent s'authentifier avec un nom d'usager et un mot de passe. Dans le cas du serveur d'authentification,

ces informations se retrouvent dans le fichier de configuration, dont les privilèges d'accès sont restreints. Les données dans ce fichier, ainsi que les données dans les tables de la base de données ne sont pas chiffrées. On considère dans le cadre du projet que l'accès physique au serveur est limité.

4.2.5 Fils d'exécution

Plusieurs types de fils d'exécution coexistent dans le serveur d'authentification :

- **socketServerThread** ;
- **connectionManagerThread** ;
- **individualTimerThread** ;
- **nettoyeur**.

Ce chapitre expliquera le rôle global de chacun de ces fils d'exécution. Pour l'instant, l'objectif est de les introduire brièvement. Le chapitre suivant présentera le contexte dans lequel ils sont utilisés.

socketServerThread

Le fil d'exécution **socketServerThread** s'occupe d'accepter les connexions sockets entrantres sur le port spécifié dans le fichier de configuration du serveur d'authentification.

Ce fil d'exécution est unique.

connectionManagerThread

Le fil d'exécution **connectionManagerThread** a pour rôle la gestion des échanges de messages entre un client et le serveur d'authentification. C'est ce fil d'exécution qui s'occupe du processus d'authentification d'un usager, décrit dans le chapitre 5.1. Il existe un fil d'exécution **connectionManagerThread** pour chaque connexion entre un client et un serveur.

Les fils d'exécution **connectionManagerThread** sont représentés par les objets ThreadData décrits au chapitre 4.2.3. Ces objets ThreadData sont enregistrés dans une liste qu'on appelle "liste de gestion".

individualTimerThread

Les fils d'exécution **individualTimerThread** fonctionnent en paire avec les fils d'exécution **connectionManagerThread**. Pour chaque fil d'exécution **connectionManagerThread**, il existe un seul et unique fil d'exécution **individualTimerThread**.

Leur rôle est de gérer la durée de vie du fil d'exécution **connectionManagerThread** qui lui est associé.

Leur identificateur de fil d'exécution (*process ID*) est contenu dans le même objet ThreadData que celui du fil d'exécution **connectionManagerThread** auquel il est associé.

nettoyeur

Le fil d'exécution **nettoyeur** a pour rôle la surveillance de la "liste de gestion". Le **nettoyeur** s'occupe de vider la "liste de gestion" lorsque celle-ci est surchargée.

Le **nettoyeur** est en fait un compteur spécial qui lorsqu'expiré, tue l'object ThreadData le plus récent contenu dans la "liste de gestion", correspondant au fil d'exécution **connectionManagerThread** le plus récent.

L'objectif est d'éviter au serveur d'authentification de se faire saturer par des connexions sockets passives, résultant par exemple d'une attaque de type déni de service, qui empêcheraient à des connexions légitimes d'être traitées.

La durée du compteur du fil d'exécution **nettoyeur** n'est pas fixe, elle est fonction du taux d'occupation de la liste de gestion. Trois paramètres, spécifiés dans le fichier de configuration, permettent de déterminer la valeur du compteur :

- MaxTimeout ;
- MinTimeout ;
- MaxThreads.

Le paramètre MaxTimeout indique la valeur maximale du compteur **nettoyeur** (lorsque la liste de gestion contient un seul fil d'exécution). MinTimeout indique la valeur minimale du compteur **nettoyeur** (lorsque la "liste de gestion" est pleine). Finalement, MaxThreads indique la taille maximale de la liste de gestion.

Deux formules impliquant ces paramètres permettent de calculer la valeur du compteur :

> variation = (MaxTimeout - MinTimeout) / MaxThread
>
> durée du compteur = MaxTimeout - ((Nombre de fils d'exécution dans la liste - 1) * variation)

Chaque fois qu'un fil d'exécution est ajouté ou supprimé de la "liste de gestion", le **nettoyeur** est tué, et est relancé, avec la nouvelle valeur du compteur. La raison qui motive une durée variable du compteur est la suivante : plus la liste est pleine, plus il est urgent

de la vider ; plus la liste est vide, moins il est urgent de la vider. L'objectif est que la liste de gestion ne soit jamais pleine.

4.2.6 Considérations tardives

Lors de la période d'analyse et de conception du serveur d'authentification, certaines décisions ont été basées sur des hypothèses erronées. Cela s'est traduit par la réalisation de deux modules n'étant pas intégrés dans la version finale de SécureMédic.

Infrastructure à clés publiques initiale

À l'origine, il avait été convenu que les usagers seraient identifiés eux aussi par une paire de clés privée/publique et un certificat numérique, comme le sont les entités de type client et de type serveur (voir chapitre 4.1.2). Cette décision fut basée sur l'hypothèse suivante :

> la validation du *common name* du certificat numérique (voir chapitre 3.3.1) n'est pas faite de manière systématique pendant l'authentification client, lors du *handshake* TLS (voir chapitre 3.4.1).

Or, cette hypothèse était fausse. L'infrastructure à clés publiques initiale prévoyait que le serveur d'authentification possède toutes les paires de clés de type usager ainsi que tous les certificats de type usager. Ainsi, lorsqu'un usager s'authentifiait auprès du serveur d'authentification, celui-ci envoyait la paire de clés et le certificat de l'usager au client MédicIP. De cette façon, chaque usager était en possession de son identité numérique pour créer une conférence multimédia sécurisée à l'aide du protocole SIP.

Malheureusement, il ne fut point constaté que le protocole SIP faisait une vérification systématique du *common name* du certificat lors du *handshake* TLS. Cela a provoqué un grave problème au niveau de l'exécution globale du système. En effet, le *common name* d'un certificat est typiquement l'adresse IP de l'entité. Comme un usager peut se connecter sur un client MédicIP depuis n'importe quel ordinateur (donc depuis n'importe quelle adresse IP), la signalisation SIP provoquait une erreur, car le *common name* ne correspondait pas à l'adresse IP actuelle de l'usager.

Cette situation a nécessité un changement d'approche. L'utilisation distribuée des certificats usagers ne pouvant être fonctionnelle étant donné le fonctionnement du protocole SIP, il a été convenu de modifier l'infrastructure à clés publiques afin qu'elle corresponde à celle présentée précédemment. Le travail effectué à la version initiale de la PKI a donc été fait en vain, se résultant en une perte de temps significative.

C'est aussi cette validation du *common name* d'après l'adresse IP qui a nécessité l'utilisation d'adresses IP statiques, comme mentionné au chapitre 4.1.4.

Serveur relais

L'équipe MédicIP a développé et intégré un système de communication complètement maillé à leur prototype d'application de télémédecine (voir chapitre 2.2.3). Dans la même optique de vouloir améliorer la robustesse du modèle de communication, il avait été décidé d'inclure un serveur relais au projet SécureMédic.

La motivation derrière ce module était l'hypothèse suivante :

> Si un usager n'est pas en mesure de rejoindre le serveur d'authentification pour s'authentifier (ex. : problème/panne réseau), il pourrait s'authentifier à l'aide d'un relais situé sur le client MédicIP d'un autre usager.

Le serveur relais était donc un module présent dans le projet SécureMédic, mais était une entité logique indépendante du client MédicIP. Il s'agissait en fait d'un processus (*daemon*) tournant sur l'hôte sur lequel un client MédicIP est installé.

L'idée globale était simple. Si un client MédicIP n'était pas en mesure de rejoindre le serveur d'authentification, il tentait d'effectuer automatiquement une connexion à un serveur relais afin que celui-ci transmette la requête au serveur d'authentification, tout en assurant la confidentialité des échanges.

En cours de développement, la raison d'être de ce type de serveur a été remise en question. En effet, à cause de la manière dont fonctionne le protocole Internet, si une entité ne peut rejoindre un serveur, une autre ne le pourra pas non plus. Le protocole Internet permet de gérer lui-même l'accès à des adresses IP, ainsi que de détourner le trafic lors de pannes empêchant l'utilisation d'une route optimale.

Une seconde hypothèse a donc été formulée :

> Si un usager n'est pas en mesure de rejoindre le serveur d'authentification pour s'authentifier (ex : problème/panne réseau), cela indique que le serveur d'authentification ne répond pas et qu'il est en panne.

Suite à cette hypothèse, le développement en cours du serveur relais a été interrompu. Il a néanmoins été discuté du développement possible de relais sans-fil, fonctionnant directement à la couche transport du modèle de l'OSI, permettant l'accès sécurisé au serveur

d'authentification à des clients MédicIP n'étant pas en mesure de se connecter à un réseau sans-fil pour des raisons de portée.

Les retards / pertes de temps engendrés par l'annulation du développement de ce module ont été beaucoup plus modestes que ceux engendrés par l'infrastructure à clés publiques initiale. On parle ici d'environ 4 semaines.

4.3 Architecture - Client SécureMédic

4.3.1 Rôle

Le client SécureMédic est un module ajouté au client MédicIP. Celui-ci a pour rôle de gérer les communications avec le serveur d'authentification, permettant ainsi à un utilisateur de s'authentifier.

Dans le chapitre 4.1, on remarque aussi que le module MédicIPSIPCore a été modifié. MédicIPSIPCore permet d'utiliser la librairie commerciale SIP V4.0 de la compagnie M5T inc. [M5T, 2008]. Il s'agit plus particulièrement d'une librairie (DLL) programmée en C++ qui est utilisée par MédicIP. Cette DLL n'était initialement pas en mesure de gérer la sécurité au niveau de l'établissement et du déroulement d'une conférence multimédia. Les modifications apportées ont réglé cet aspect. Les détails sur le processus d'établissement de communications sécurisées seront néanmoins abordés au chapitre 5.1.

4.3.2 Dépendances logicielles

Le module client SécureMédic, tout comme le serveur d'authentification, possède une dépendance logicielle à la librairie OpenSSL. Les informations relatives à cette dépendance sont les mêmes que celles présentées pour le serveur d'authentification au chapitre 4.2.2.

Le module client SécureMédic a été développé en C++. Excepté la dépendance à OpenSSL, ce module est complètement indépendant et ne fournit que les fonctionnalités de base pour l'authentification d'un usager ainsi que pour le changement de mot de passe d'un usager. Par conséquent, ce module peut être intégré à n'importe quel logiciel voulant utiliser le mécanisme d'authentification développé pour le serveur d'authentification. De plus, un pont spécifique pour le client SécureMédic a été développé pour pouvoir intégrer ce code C++ dans un projet C#, comme dans le cas présent avec MédicIP.

4.3.3 Modules

Le diagramme de classes suivant est présenté afin d'introduire les différents modules composant le client SécureMédic :

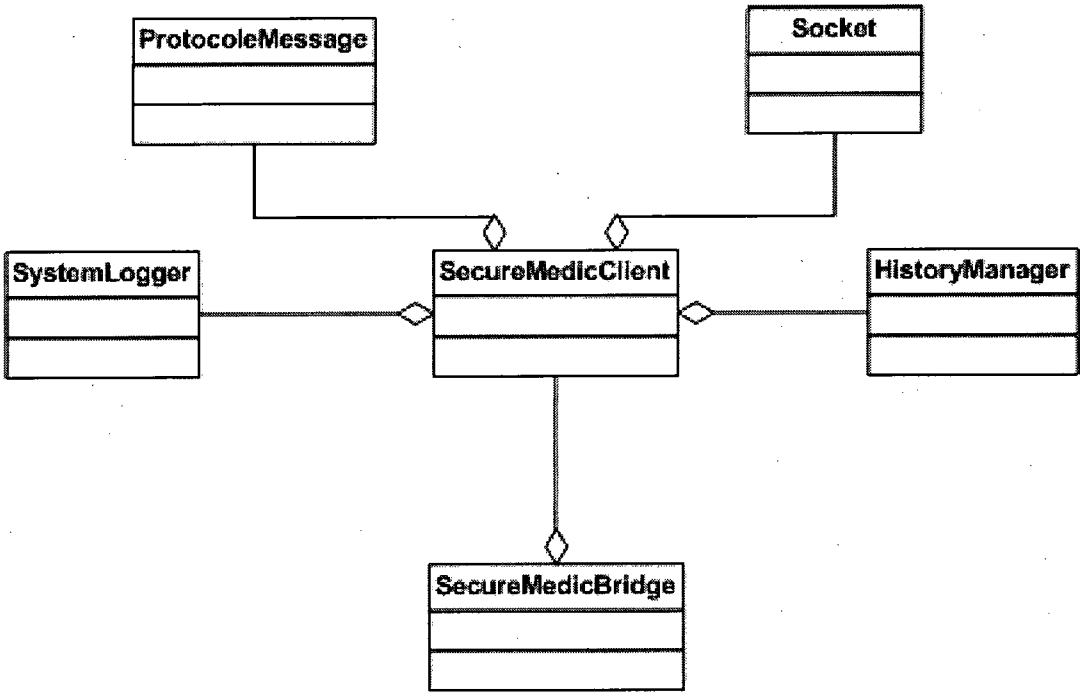


Figure 4.5 Diagramme de classes du module client SécureMédic.

SystemLogger

La classe SystemLogger est un module permettant au client SécureMédic d'écrire des événements dans un fichier de *logs* qui lui est propre, étant donné que le Client SécureMédic est modulaire et peut être utilisé dans d'autre projets. Cette classe est donc utilisée exclusivement par le module client SécureMédic et évite à celui-ci de dépendre d'un module de *logs* externe et propre à une application tierce.

La classe SystemLogger a été conçue selon le patron de conception *Singleton*, permettant ainsi à n'importe quel module du client SécureMédic d'enregistrer des événements. Il est important de spécifier que le type d'événements géré par ce module ne consiste pas en des événements reliés au fonctionnement normal de l'application. Il sert plutôt à signaler des événements internes au module client SécureMédic, majoritairement des erreurs fatales qui empêchent son bon fonctionnement. Voici quelques types d'erreurs classiques :

- erreur dans le chargement des jeux de chiffrement supportés ;
- erreur dans le chargement de certificats ou de clés ;
- erreur dans le chargement de l'historique local (voir HistoryManager de ce chapitre) ;
- erreur dans la validation du *trusted certificate* (voir chapitre 4.3.4) ;
- erreur fatale dans l'utilisation du protocole OpenSSL ;

– problème de connexion socket.

Socket

La classe Socket, comme son nom l'indique, représente un socket TCP. Cette classe a été conçue spécifiquement pour communiquer avec le serveur d'authentification et permet les opérations de base sur des sockets.

ProtocolMessage

ProtocolMessage constitue le même module que celui décrit pour le serveur d'authentification (voir chapitre 4.2.3 pour davantage de détails).

HistoryManager

La classe HistoryManager permet de gérer l'historique local, utilisé lors de l'authentification locale d'un usager, situation qui n'a jusqu'ici pas été abordée. Les informations concernant le fonctionnement et le cas d'utilisation de l'authentification locale seront présentées dans le chapitre 5.1.2. Pour l'instant, une description technique du module suffira.

L'historique local contient un historique des authentifications réussies avec le serveur d'authentification. Chaque fois qu'un usager s'authentifie correctement au logiciel MédicIP, la date, son nom d'utilisateur ainsi que le résultat du hachage de son mot de passe sont inscrits dans l'historique, qui est représenté sous la forme d'un fichier entreposé sur le système de fichiers de l'hôte du client.

Deux mesures de protection sont en vigueur sur ce fichier. Premièrement, il est chiffré à l'aide de la clé de type client, de manière à ce que personne ne puisse accéder au contenu confidentiel. Deuxièmement, le protocole permettant la sauvegarde du fichier utilise un système de hachage (voir section 3.2) afin de vérifier si son contenu est altéré. On parle ici d'intégrité des données.

Voici le format de l'historique local :

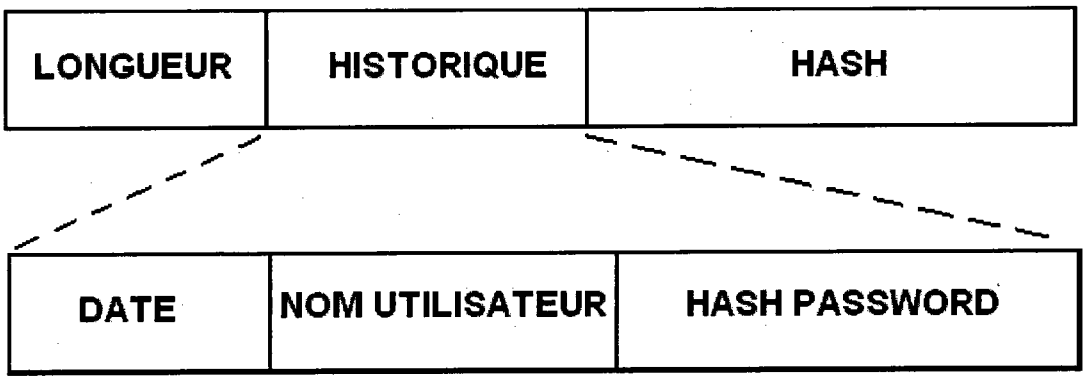


Figure 4.6 Format de l'historique local.

Dans la première section, on remarque que le fichier est séparé en trois parties : LONGUEUR, HISTORIQUE et HASH.

La LONGUEUR représente la taille de la section HISTORIQUE en plus de la taille de la section HASH.

L'HISTORIQUE contient la charge utile pour l'authentification locale, c'est-à-dire les entrées faites dans l'historique.

Le HASH contient le résultat du hachage de la section LONGUEUR et HISTORIQUE.

À première vue, on peut se demander comment il est possible de construire ce fichier, puisque la section LONGUEUR dépend de la section HASH, et que la section HASH dépend de la section LONGUEUR. Quelle section est alors construite en premier ? Il est important de noter que la section HASH possède une taille fixe, étant donné que le résultat d'un hachage est toujours de taille fixe. Il est par conséquent possible de construire la section LONGUEUR, connaissant la taille qu'aura la section HASH avant même qu'elle soit construite. L'ordre de construction du fichier est donc le suivant : HISTORIQUE, LONGUEUR, HASH.

La section HISTORIQUE, quant à elle, est formée des informations suivantes : DATE, NOM UTILISATEUR, HASH PASSWORD.

La DATE correspondant à la date de la dernière authentification réussie de l'usager avec le serveur d'authentification.

Le NOM UTILISATEUR correspond au nom d'utilisateur de l'usager.

Le HASH PASSWORD correspond au hachage du mot de passe de l'usager.

Voici un exemple complet du format de l'historique :

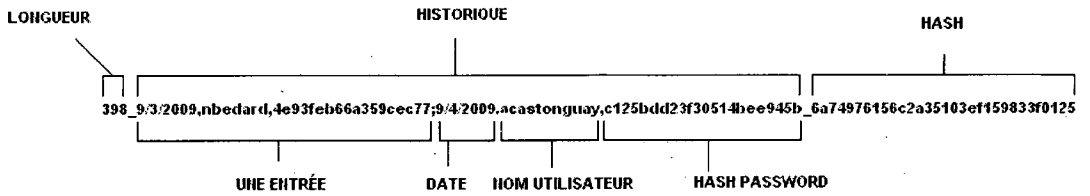


Figure 4.7 Exemple de l'historique local complet.

Les sections LONGUEUR, HISTORIQUE et HASH sont séparées avec le caractère de séparation "_".

Les sections DATE, NOM UTILISATEUR et HASH PASSWORD (contenues dans une entrée de l'HISTORIQUE) sont séparées avec le caractère de séparation ",".

Les différentes entrées de la section HISTORIQUE sont séparées avec le caractère de séparation " ;".

SecureMedicClient

Il s'agit du coeur du module Client SécureMédic. Cette classe permet d'effectuer les opérations de base permettant une authentification usager :

- authentification distante (avec le serveur d'authentification) ;
- authentification locale (avec l'historique local, voir chapitre 5.1.2) ;
- mise à jour du mot de passe usager.

Afin d'utiliser cette classe, celle-ci doit être initialisée avec les paramètres suivants :

- adresse IP du serveur d'authentification ;
- port de connexion du serveur d'authentification ;
- répertoire du fichier de l'historique local ;
- clé privée de type client (voir chapitre 4.1.3) ;
- mot de passe de la clé privée client (voir chapitre 4.1.3) ;
- certificat numérique de type client (voir chapitre 4.1.3) ;
- certificat numérique de tyep serveur, aussi appelé *trusted certificate* (voir chapitre 4.1.4).

SecureMedicBridge

Comme mentionné précédemment, le projet MédicIP est programmé en C#. Afin de pouvoir utiliser le module Client SécureMédic, développé en C++, un pont CLI/C++ dédié a été développé.

Il s'agit essentiellement d'une classe contenant un pointeur vers la classe SecureMedicClient, et où chacune des fonctions publiques de la classe SecureMedicClient possède sa propre fonction CLI/C++, qui est accessible depuis du code écrit en C++.

4.3.4 TrustedCertificateProtector

Le certificat de type serveur, aussi appelé *trusted certificate* (voir chapitre 4.1.4), est utilisé par le client MédicIP afin de valider l'authenticité du serveur d'authentification.

Dans l'éventualité où un pirate venait à s'infiltrer sur l'hôte d'un client MédicIP, celui-ci pourrait théoriquement remplacer le certificat de type serveur. Cette attaque permettrait à un pirate de créer un faux serveur d'authentification sur lequel un client s'authentifierait, permettant ainsi au faux serveur de voler les paramètres d'authentification des usagers utilisant ce client MédicIP.

Pour pallier ce type d'attaque, le client MédicIP possède une mesure de protection permettant la validation du *trusted certificate*.

Préalablement, une procédure doit être effectuée afin d'inclure cette mesure de protection dans le code compilé du client SécureMédic. Un outil indépendant a été développé, pour permettre d'effectuer cette procédure de manière rapide et automatisée.

Cet outil requiert en entrée le *trusted certificate* en question, la clé privée du client MédicIP, ainsi que son mot de passe (voir chapitre 4.1.3). Trois étapes principales sont alors exécutées :

1. opération de hachage (SHA512) sur le *trusted certificate* original ;
2. chiffrement du hachage avec la clé privée du client et l'algorithme AES ;
3. génération automatique du fichier TrustedCertificateProtection.h contenant le chiffrement du hachage du *trusted certificate*.

Le fichier TrustedCertificateProtection.h est nécessaire à la compilation du module Client SécureMédic. Ainsi, lors de l'initialisation du module client SécureMédic, il y a exécution du processus de "validation du *trusted certificate*".

Cette vérification implique les opérations suivantes :

1. récupération de la clé publique du client MédicIP depuis son certificat numérique ;

2. déchiffrement du hachage chiffré se trouvant dans le fichier TrustedCertificateProtection.h. Après cette étape, on est donc en possession du hachage du *Trusted Certificate* original ;
3. opération de hachage sur le *trusted certificate* à valider (celui potentiellement piraté) ;
4. comparaison des deux résultats de hachage (étape 2 et étape 3).

Si l'étape 4 permet de valider que les deux résultats de hachage sont identiques, cela signifie que nous sommes en présence de l'authentique *trusted certificate.*

En résumé, l'outil permet de créer un fichier contenant la signature chiffrée du *trusted certificate.* Lors de l'initialisation du client SécureMédic, il y a déchiffrement de cette signature. Finalement, la signature du *trusted certificate* à valider est comparée avec celle fournie par l'outil.

CHAPITRE 5

FONCTIONNEMENT DU SYSTÈME

5.1 Authentification usager sécurisée

Afin de pouvoir utiliser l'application MédicIP, un usager doit préalablement s'authentifier avec succès. Deux types d'authentification sont possibles dans le projet SécureMédic :

- authentification distante ;
- authentification locale.

Ces deux types d'authentification visent à corriger une faille de sécurité importante du prototype MédicIP présentée dans le chapitre 2.3.1 : l'authentification non sécurisée des usagers. La raison motivant la présence d'un mode d'authentification alternatif (l'authentification locale) est de permettre une plus grande flexibilité advenant le cas où le serveur d'authentification est en panne. Pour plus de détails, il est possible de se référer à la section 5.1.2 .

Voici donc en détail chacun de ces processus.

5.1.1 Authentification à distance

Avant d'aborder le fonctionnement de l'authentification à distance, il est important de présenter la répartition initiale des différents éléments de l'infrastructure présentée au chapitre précédent.

A Mot de passe servant à déchiffrer la clé privée de type client D (voir chapitre 4.1.3). Ce mot de passe est incorporé (*hardcode*) à même le code logiciel du Client SécureMédic.

B Il s'agit de la mesure de protection du certificat numérique de type serveur E (*trusted certificate*) (voir chapitre 4.3.4).

C Certificat numérique de type client (voir chapitre 4.1.4).

D Clé publique et clé privée de type client (voir chapitre 4.1.3). La clé privée peut être déchiffrée à l'aide du mot de passe A.

E Certificat numérique de type serveur, aussi appelé *trusted certificate* (voir chapitre 4.1.4).

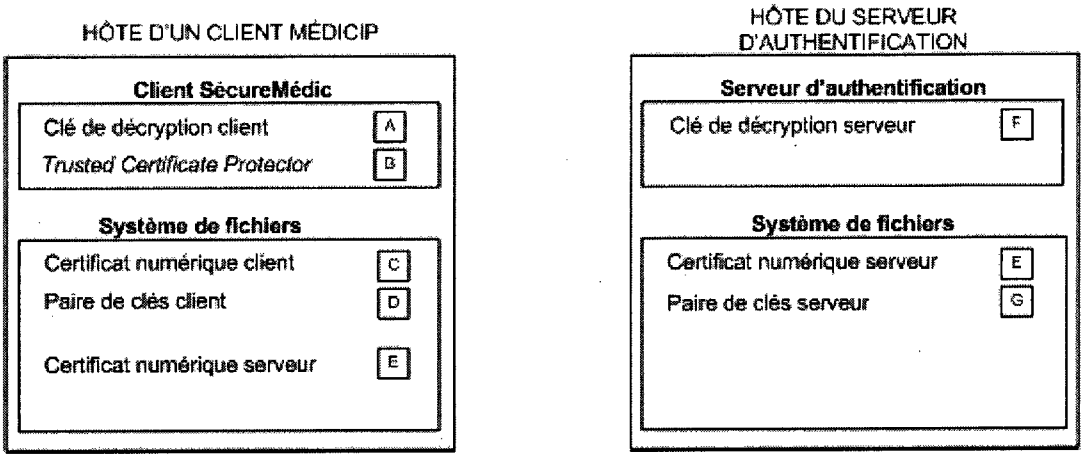


Figure 5.1 Répartition initiale des différents éléments de l'infrastructure.

F Mot de passe servant à déchiffrer la clé privée de type serveur G (voir chapitre 4.1.3). Ce mot de passe est incorporé (*hardcode*) à même le code logiciel du serveur d'authentification.

G Clé publique et clé privée de type serveur (voir chapitre 4.1.3). La clé privée peut être déchiffrée à l'aide du mot de passe F.

L'authentification à distance, avec le serveur d'authentification, s'effectue en deux phases :

1. authentification mutuelle ;

2. authentification usager.

Voici une ligne du temps situant ces deux grandes phases ainsi que les différents fils d'exécution présentés au chapitre 4.2.5 :

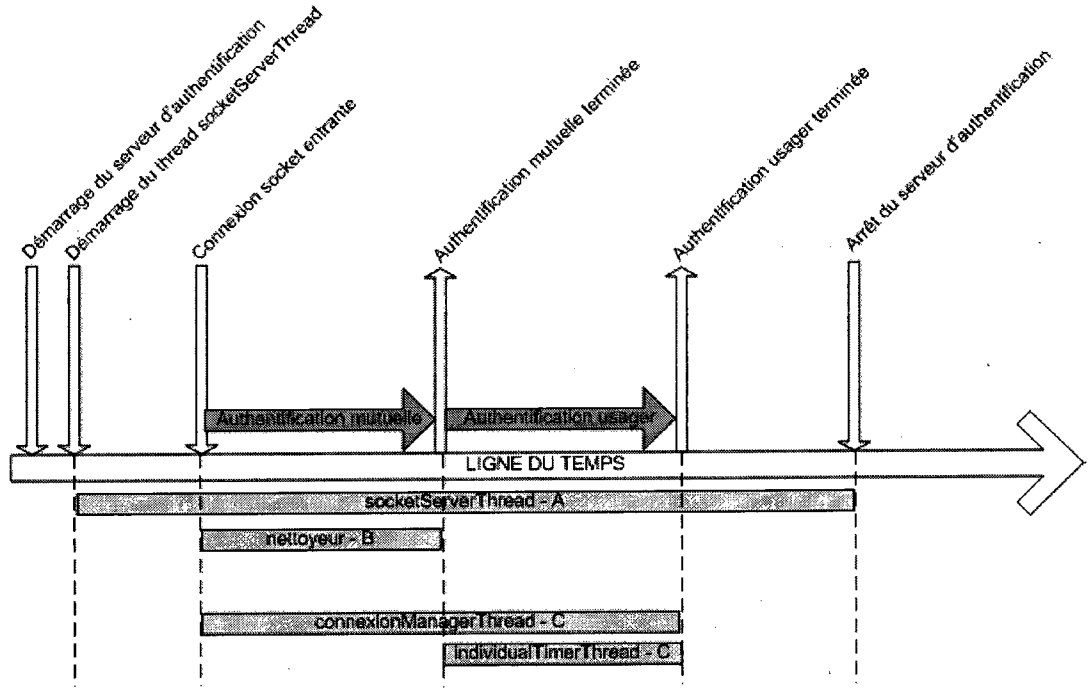


Figure 5.2 Ligne du temps représentant le processus d'authentification à distance de manière globale.

Les deux phases globales seront détaillées dans les sections suivantes, tout comme les relations entre les fils d'exécution.

Authentification mutuelle

L'authentification mutuelle joue un rôle clé dans le processus d'authentification des usagers. Elle permet en effet de valider deux critères importants :

1. assure au client que le serveur d'authentification est le bon ;
2. assure au serveur d'authentification que le client est le bon.

Cette procédure de validation permet d'exclure les "fausses" entités voulant participer au processus d'authentification. Pour ce faire, deux étapes sont nécessaires.

Premièrement, un client MédicIP doit initier une connexion socket avec le serveur d'authentification qui lui, est en attente de connexions entrantes. C'est le fil d'exécution **socketServerThread** du serveur d'authentification qui s'occupe de cette tâche (voir chapitre 4.2.5).

Deuxièmement, après l'établissement de la connexion socket entre le client et le serveur d'authentification, il y a création d'un objet ThreadData représentant cette connexion. Le

fil d'exécution **socketServerThread** lance ensuite un fil d'exécution **connectionManagerThread** (voir chapitre 4.2.5), lui passe l'objet ThreadData correspondant, et se remet en attente de connexions entrantes. Il se produit par la suite ce qu'on appelle un *handshake* TLS (voir chapitre 3.4.1). Globalement, ce *handshake* permet aux deux entités de s'entendre sur un jeu de chiffrement et sur une clé cryptographique symétrique, permettant ainsi l'établissement d'un canal de communication sécurisé. L'étape clé dans l'authentification mutuelle est la vérification des certificats numériques d'autrui (voir chapitre 3.3.5), s'effectuant lors du *handshake* TLS.

Lorsque ces deux étapes s'effectuent correctement, les deux entités (le client MédicIP et le serveur d'authentification) sont assurées de leur authenticité respective et peuvent s'échanger de l'information de manière sécuritaire. Ce *handshake* TLS, ainsi que les échanges de données via le canal sécurisé, sont gérés en totalité par le fil d'exécution **connectionManagerThread**.

On remarque que durant le processus de l'authentification mutuelle, le fil d'exécution **connexionManagerThread** n'est pas encore associé à un fil d'exécution **individualTimerThread**. Tant et aussi longtemps que l'authentification mutuelle n'est pas complétée, c'est le **nettoyeur** qui est chargé de gérer la durée de vie des fils d'exécution **connexionManagerThread**. Ce n'est que lorsque l'authentification mutuelle est complétée que le fil d'exécution **connexionManagerThread** est retiré de la "liste de gestion" et qu'un fil d'exécution **individualTimerThread** lui est attribué. De cette façon, si plusieurs connexions socket passives s'effectuent, le **nettoyeur** pourra rapidement les éliminer, de manière à ce que le nombre maximal de connexions ne soit jamais atteint.

Tout le processus de l'authentification mutuelle est transparent pour l'usager utilisant l'application MédicIP. Voici maintenant le résumé du déroulement de l'authentification mutuelle.

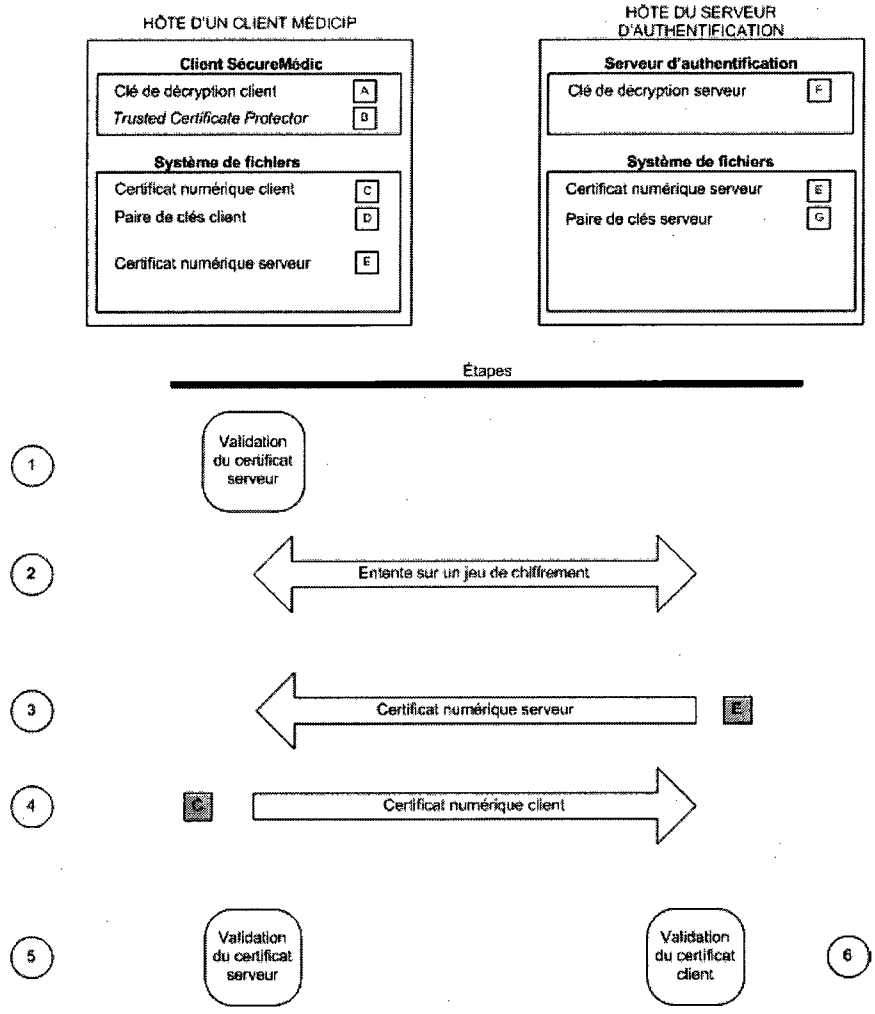


Figure 5.3 Authentification mutuelle et établissement d'un canal de communication sécurisé.

Étape 1 Cette étape a pour objectif de vérifier la validité du *trusted certificate* (C) à l'aide de la protection du certificat numérique (B). Voir chapitre 4.3.4.

Étape 2 Cette étape se produit dans le *handshake* TLS.

Étape 3 et 4 Le client et le serveur s'échangent leur certificat numérique.

Étape 5 Le client valide l'authenticité du certificat reçu.

Étape 6 Le serveur valide l'authenticité du certificat reçu.

Authentification usager

Lorsque l'authentification mutuelle entre un client MédicIP et le serveur d'authentification se termine avec succès, il se produit par la suite l'authentification usager.

Comme les deux entités peuvent se communiquer de l'information via un canal sécurisé établi après l'authentification mutuelle, il est donc possible pour le client MédicIP d'envoyer au serveur d'authentification les informations relatives à l'authentification de l'usager : son nom d'utilisateur ainsi que son mot de passe.

Si l'authentification usager réussit, l'application MédicIP démarre, permettant à l'usager de l'utiliser. De plus, le client MédicIP met à jour l'historique local (voir chapitre 4.3.3).

Voici le résumé du déroulement de l'authentification usager :

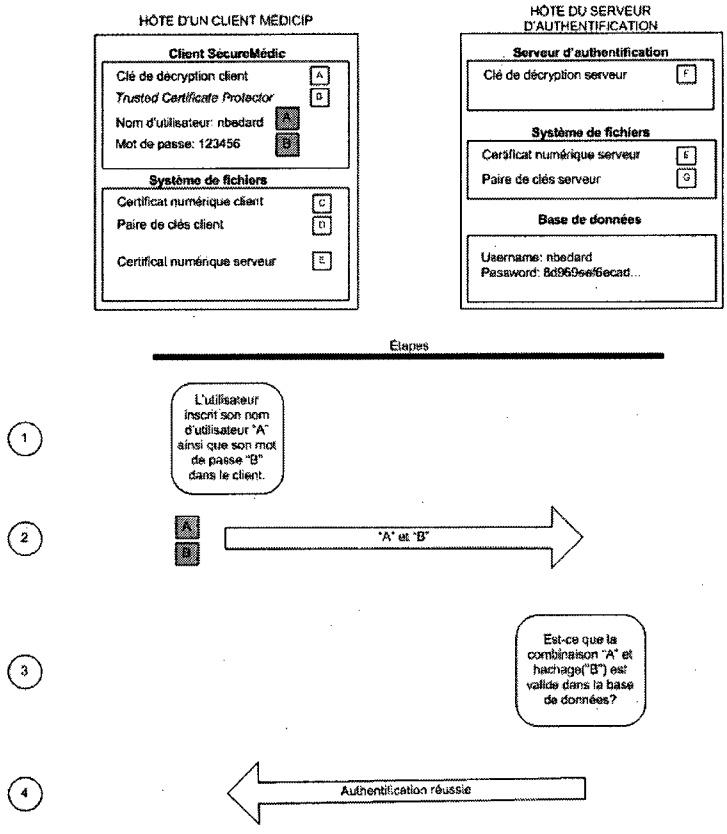


Figure 5.4 Authentification usager.

Étape 1 L'usager inscrit, depuis le client, son nom d'usager et son mot de passe.

Étape 2 Le client envoie, à travers le canal sécurisé établi précédemment, le nom d'usager et le mot de passe au serveur.

Étape 3 Le serveur valide l'authentification usager (voir chapitre 4.2.4).

Étape 4 Le serveur envoie au client le résultat de l'authentification, permettant à l'usager d'utiliser MédicIP.

Scénario fictif 1

Imaginons un pirate informatique ayant créé un logiciel effectuant des connexions socket au serveur d'authentification. Après chacune de celles-ci, le serveur d'authentification sera en attente du *handshake* TLS venant du client (voir chapitre 5.1.1).

Trois possibilités peuvent donc se produire :

1. le faux client n'initie aucun *handshake* TLS ;
2. le faux client initie un *handshake* TLS ;
3. le faux client envoie de l'information arbitraire.

Dans le premier cas, on sera en présence d'une connexion socket passive. Dans ce genre de situation, on pourrait penser qu'en très grand nombre, ce type de connexion provoquerait une surcharge au niveau du serveur d'authentification. Le **nettoyeur**, présenté au chapitre 4.2.5, agit dans ce cas bien précis, afin d'éviter que le serveur ne soit saturé. Le temps requis pour effectuer l'authentification complète d'un usager est très court (moins d'une seconde), dépendamment de la puissance du serveur d'authentification et de la congestion du réseau. Il est donc facile de configurer la valeur du compteur du nettoyeur (voir chapitre 4.2.5), afin de s'assurer que des connexions passives soient interrompues par le nettoyeur.

Dans le deuxième cas, un faux client ne sera pas en possession d'un certificat numérique signé par la clé privée *root*, ou tout simplement pas en possession d'un certificat numérique de quelque nature que ce soit. Dans cette situation, le *handshake* TLS échouera et le serveur d'authentification mettra fin à cette connexion.

Dans le troisième cas, il se produira le même résultat que dans le cas précédent. Le serveur d'authentification est en attente d'un *handshake* TLS venant d'un client MédicIP. Si tout autre type de donnée/requête est envoyé au serveur d'authentification, celui-ci mettra fin à la connexion.

Scénario fictif 2

Supposons qu'un client MédicIP tente de s'authentifier avec le serveur d'authentification, mais qu'après l'authentification mutuelle, il se produit une erreur réseau quelconque. À ce moment, le client MédicIP n'est pas en mesure de terminer le processus d'authentification, qui est au stade de l'authentification usager. Le serveur est donc en attente du nom d'usager ainsi que du mot de passe de l'utilisateur.

En temps normal, le fil d'exécution **connectionManagerThread** (voir chapitre 4.2.5) resterait actif indéfiniment. Grâce au fil d'exécution **individualTimerThread** (voir chapitre

4.2.5), qui est lancé après l'authentification mutuelle, le fil d'exécution **connectionManagerThread** en cours sera tué lorsque sa durée de vie maximale sera atteinte.

Le fil d'exécution **individualTimerThread** s'assure donc qu'un fil d'exécution actif ne tombe pas dans un état de transition sans fin.

5.1.2 Authentification locale

Dans les cas classiques d'authentification présentés jusqu'ici, le serveur d'authentification est toujours impliqué. Dans le but d'offrir une fiabilité accrue, le client SécureMédic offre un type d'authentification permettant à un usager d'utiliser l'application MédicIP même si le serveur d'authentification est en panne, en maintenance ou inaccessible.

Lorsque le client MédicIP essaie de contacter le serveur d'authentification lors de l'authentification mutuelle (voir chapitre 5.1.1), mais que celui-ci ne répond pas,le processus d'authentification locale set met alors en marche. Pour qu'un utilisateur puisse s'authentifier de façon locale, une condition importante doit être respectée : l'utilisateur en question doit préalablement s'être authentifié à distance sur ce même client MédicIP.

Lors d'une authentification à distance réussie, le client SécureMédic met à jour un historique local contenant une liste des authentifications précédentes. Le format de cette liste est présenté au chapitre 4.3.3. Ainsi, lorsque le client SécureMédic n'est pas en mesure de contacter le serveur d'authentification, celui-ci se tourne vers cet historique local.

Ce système permet donc à un usager d'utiliser l'application et de participer à une conférence même, si le serveur d'authentification ne répond pas.

5.2 Conférences multimédias sécurisées

Dans le chapitre 2.3.2, il a été démontré que des failles de sécurité importantes existaient au niveau du prototype MédicIP. Ces failles touchaient principalement trois aspects :

1. la signalisation SIP (voir chapitre 3.4.3) n'est pas sécurisée ;
2. les échanges vocaux entre les participants d'une conférence ne sont pas sécurisés ;
3. les échanges d'électrocardiogrammes entre les participants d'une conférence ne sont pas sécurisés.

Comme stipulé dans le chapitre 4.3.1, la portion adressant ces aspects de sécurité se trouve dans le module MédicIPSIPCore. Les concepts derrière ce processus de sécurisation seront exposés dans les sections suivantes.

5.2.1 Sécurisation du protocole SIP

Le prototype MédicIP utilisait le protocole SIP pour faire la signalisation des conférences multimédias. Afin de sécuriser ce protocole, le projet SécureMédic le combine avec le protocole TLS (voir chapitre 3.4.1).

L'objectif est de permettre à la librairie SIP d'être utilisée par dessus le protocole TLS, d'où le terme "SIP *over* TLS". Il ne s'agit donc pas d'un protocole proprement dit, mais bien de deux protocoles distincts utilisés à des niveaux différents. Ainsi, avant d'amorcer les échanges SIP, il y a établissement d'un canal de communication sécurisé TLS.

La librairie offerte par M5T inc. permet l'utilisation de "SIP *over* TLS". Pour ce faire, le projet SécureMédic configure la librairie en suivant des étapes bien précises. Voici les étapes générales de configuration, qui s'appliquent à n'importe quelle implémentation de SIP *over* TLS :

1. Récupération de la clé privée de l'usager.
2. Récupération du certificat numérique de l'usager.
3. Récupération du certificat numérique serveur (*root certificate*).
4. Création d'une chaîne de certificats (certificat usager et certificat serveur).
5. Création de contextes de type serveur et client à l'aide de la chaîne de certificats.

Le concept deumeure le même que lorsqu'on utilise le protocole TLS seul. L'utilisation de certificats et de clés permet d'effectuer un *handshake* et ainsi, de créer le canal sécurisé par où transitent les échanges SIP.

La différence dans ce cas-ci est que le projet SécureMédic n'utilise pas la librairie OpenSSL, mais bien l'implémentation offerte par M5T inc.

5.2.2 Sécurisation des échanges vocaux

Lorsqu'une conférence démarre après la signalisation SIP, d'autres protocoles prennent le relais pour ce qui est du transfert des données entre les entités. Dans le cas des échanges

vocaux, c'est le protocole RTP (voir chapitre 3.4.2) qui entre en jeu, tel que décrit dans le chapitre 2.3.2.

Les problèmes de sécurité reliés au protocole RTP (voir chapitre 3.4.2) requièrent un correctif pour permettre des échanges vocaux sécurisés. La librairie de M5T inc. fournit une implémentation du protocole SRTP, soit une version sécurisée du protocole RTP (voir section 3.4.2). Dans le même ordre d'idées que pour la sécurisation SIP, le projet SécureMédic configure la librairie, permettant ainsi l'utilisation de SRTP.

Dans une situation idéale, l'utilisation de SRTP se fait de concert avec un protocole permettant l'échange, ou la négociation, d'une clé de cryptographie symétrique, utilisée lors du chiffrement des paquets. Dans la version projet SécureMédic proposée, la clé symétrique utilisée est incorporée (*hardcode*) à même le code logiciel du client MédicIP. La section 7.3 abordera les tavaux futurs à propos de cet aspect.

5.2.3 Sécurisation des échanges de données

Le prototype MédicIP utilise la librairie de M5T inc. pour envoyer les électrocardiogrammes via le protocole UDP. Tel que mentionné dans le chapitre 2.3.2, ce protocole n'offre pas de mesures de protection adéquates.

L'utilisation de cette librairie par MédicIP est telle que l'envoi d'électrocardiogrammes se fait par l'entremise des mêmes contextes SIP que ceux utilisés lors de la signalisation (voir chapitre 5.2.1). Étant donné que le projet SécureMédic configure la librairie de manière à utiliser SIP de concert avec TLS, les contextes SIP utilisés sont donc sécurisés, ce qui permet du même coup la sécurisation des échanges de données.

5.3 Résumé

Ce chapitre a permis de montrer comment les modules de l'architecture présentés au chapitre 4 fonctionnent et de quelle manière ils viennent corriger les failles de sécurité précédemment ciblées.

Le chapitre suivant présentera une analyse de fonctionnement réel du projet SécureMédic d'après certains tests effectués.

CHAPITRE 6
ANALYSE ET SYNTHÈSE DES RÉSULTATS

L'objectif de ce chapitre est de porter un jugement critique sur le système développé et d'effectuer une synthèse des résultats des tests effectués.

Cette analyse portera sur les points suivants :

1. analyse des performances du serveur d'authentification ;
2. analyse du comportement du processus d'authentification à distance ;
3. analyse du comportement du processus d'authentification locale ;
4. analyse des conférences sécurisées.

6.1 Analyse des performances du serveur

L'annexe A présente les résultats de différents tests effectués pour permettre de quantifier les performances du serveur d'authentification d'après trois critères, soient :

- l'utilisation de la mémoire ;
- le temps requis pour les authentifications ;
- l'utilisation du processeur.

6.1.1 Utilisation de la mémoire

Pour ce qui est de l'utilisation de la mémoire, on remarque qu'à l'état initial, le serveur d'authentification requiert très peu de mémoire : 3040 kilo-octets. Lors d'une authentification, réussie ou échouée, le serveur requiert l'utilisation d'une certaine quantité de mémoire supplémentaire, soit environ 85 kilo-octets (voir annexe A).

À des fins de comparaison, 12 connexions utiliseraient environ 1024 kilo-octets. Un ordinateur typique vendu en octobre 2009 possède en moyenne 4096 méga-octets de mémoire, ce qui représente l'équivalent de près de 50 000 connexions.

On remarque par contre un problème au niveau des résultats. En effet, ces authentifications ne sont pas simultanées, mais bien consécutives, signifiant la présence d'une fuite de mémoire (*memory leak*) dans le prototype du serveur d'authentification.

En théorie, après 50 000 connexions simultanées ou successives, l'hôte sur lequel est exécuté le serveur d'authentification serait complètement saturé, nécessitant l'utilisation de la mémoire *swap*, et dégradant par le fait même les performances générales du serveur.

Dans le cadre du développement d'un prototype, cette fuite de mémoire ne vient pas compromettre l'objectif initial de sécurisation de MédicIP. On constate néanmoins que la solution proposée ne serait pas viable dans un contexte d'utilisation réel. Cette fuite de mémoire de 85 kilo-octets par authentification devrait être corrigée, de manière à ce que le serveur d'authentification ne manque jamais de ressources mémoire, sauf dans le cas d'authentifications parfaitement simultanées et en grande quantité.

6.1.2 Temps requis pour les authentifications et utilisation du processeur

Les résultats concernant les temps requis pour les authentifications nous indiquent que la durée moyenne d'une authentification usager prend 0.04 seconde au serveur d'authentification.

Lors de ce test, les authentifications ont été faites de manière successive. L'idéal aurait été de pouvoir effectuer un grand nombre d'authentifications simultanées, afin de voir l'impact sur le temps moyen requis, qui serait probablement à la hausse. Ce genre de test est par contre difficile à réaliser pour plusieurs raisons.

Premièrement, la durée moyenne d'authentification très courte requiert une synchronisation presque parfaite des demandes d'authentification pour effectuer un test significatif. Un simple délai de quelques centièmes de seconde entre les demandes d'authentification viendrait mettre en péril la réalisation et les résultats du test.

Deuxièmement, comme chaque client MédicIP effectuant une authentification a besoin de son propre certificat numérique concordant avec son adresse IP, cela implique une certaine logistique. Par exemple, un test nécessitant 30 connexions simultanées requiert 30 clients MédicIP, donc 30 hôtes, avec chacun leur certificat numérique. Il devient donc rapidement complexe d'organiser un tel environnement de tests.

Pour ce qui est de l'utilisation du processeur, il a été observé que son utilisation est nulle lorsque le serveur d'authentification est en attente de demandes d'authentification. Pendant le traitement d'une authentification, l'utilisation grimpe à 100% le temps que le traitement soit complété. Ce test fut également réalisé à l'aide de connexions consécutives.

6.1.3 Conclusion sur les performances du serveur

Les tests de performance ont été effectués dans un environnement où la réalisation du test optimal s'est avérée impossible étant donné sa complexité de mise en oeuvre. Une période de rodage du système, dans le cadre d'un déploiement complet, mais expérimental, permettrait d'apporter de nouvelles statistiques et de porter un jugement plus complet sur les performances du système développé.

Néanmoins, les résultats actuels permettent d'en arriver à certaines conclusions. D'abord, le serveur d'authentification est performant et peu exigeant au niveau des ressources. Bien qu'il s'agisse d'une conclusion basée sur un banc de tests non optimal, les résultats concernant l'utilisation mémoire sont transposables dans le cas de connexions simultanées.

Pour ce qui est du temps moyen d'authentification et de l'utilisation du processeur, on peut supposer que plusieurs connexions simultanées provoqueraient une utilisation de 100% du processeur, répartie entre les différents fils d'exécution, entraînant un temps d'authentification plus lent. L'important ici n'est pas de proposer une méthode pour extrapoler et spéculer sur les performances du système dans le cas d'authentifications simultanées, mais bien de voir la tendance qu'on pourrait observer. En effet, il y a beaucoup de facteurs à considérer dans ce type de prévisions : les processeurs multicoeurs, la vitesse des bus de données pour le disque dur, la mémoire vive et les cartes réseaux, la vitesse d'opération de la mémoire vive et des disques durs, les accès concourants et partage de la mémoire, etc.

On peut cependant être catégorique pour ce qui est de la fuite de mémoire observée dans le serveur d'authentification. Dans le cadre d'une preuve de concept, le système développé est fonctionnel et ne met pas en péril la sécurité globale de l'infrastructure. D'un point de vue pratique, il est par contre impensable d'utiliser un tel système dans un environnement médical, étant donné les pannes occasionnées par un redémarrage du serveur d'authentification dans le but de libérer la mémoire utilisée.

6.2 Analyse du processus d'authentification à distance

L'annexe B présente les résultats de différents tests effectués qui ont permis d'analyser le comportement de l'infrastructure gérant le processus d'authentification. Plusieurs scénarios de tests ont été évalués dans cette annexe :

- authentification réussie ;
- authentification échouée (mauvais certificat client) ;
- authentification échouée (mauvais *trusted certificate*) ;

- authentification échouée (arrêt par le **nettoyeur**) ;
- authentification échouée (arrêt par le fil d'exécution **individualTimerThread**).

Lors d'une authentification à distance réussie, les résultats exposés dans l'annexe B démontrent que lorsque les conditions nécessaires sont rassemblées, c'est-à-dire que la librarie SIP de M5T est bien configurée, que les certificats et clés utilisés sont valides et qu'un nom d'utilisateur et un mot de passe valides sont utilisés, il est possible d'authentifier un utilisateur, et ce, de manière sécuritaire. Les résultats permettent d'observer les deux phases du processus d'authentification à distance, de la connexion socket, en passant par l'établissement d'un canal de communication sécurisé, jusqu'à l'envoi de la demande d'authentification et la réception de la réponse.

Les résultats de l'annexe B permettent aussi de constater que lorsqu'une des conditions nécessaires n'est pas respectée, le système est en mesure de détecter correctement l'anomalie et de réagir adéquatement, en refusant la demande d'authentification. Le TrustedCertificateProtector (4.3.4) ainsi que le fil d'exécution **individualTimerThread** (4.2.5) ont démontré leur efficacité en permettant d'interrompre le processus d'authentification en cours, et ce, de manière sécuritaire, en empêchant l'envoi de données confidentielles depuis le serveur d'authentification.

Les résultats démontrent également l'efficacité du **nettoyeur**. On remarque toutefois les limitations d'un tel système. Advenant le cas où une véritable attaque de déni de service se produisait, c'est-à-dire qu'une quantité innombrable de connexions socket passives serait lancée, et ce, sur une période de temps très longue, le serveur d'authentification deviendrait alors inopérable. Celui-ci ne serait pas en état de surcharge, mais il cesserait d'accepter de nouvelles connexions, jusqu'à ce que le **nettoyeur** vide la "liste de gestion". Avec un peu de chance, il se pourrait qu'une demande d'authentification légitime puisse être acceptée. Mais on se rend compte que le système proposé a ses limites et qu'une attaque de déni de service bien orchestrée rendrait le service d'authentification inutilisable.

Le **nettoyeur** est aussi efficace dans deux autres situations. Premièrement, dans le cas où une demande d'authentification légitime se produirait, mais que pour une raison particulière (problème réseau par exemple), la communication entre le client et le serveur serait interrompue immédiatement après l'établissement de la connexion socket (et avant le *handshake* TLS). Dans ce cas, il ne faudrait pas que cette connexion persiste dans le temps inutilement. Deuxièmement, imaginons le scénario où le serveur est dans un état de quasi-saturation, et où les performances sont très affectées dû à une mauvaise configuration du serveur d'authentification (mauvaise durée du compteur individuel, taille de la "liste de

gestion" trop grande pour les capacités du serveur, etc.). Dans ce cas-ci, il est possible que l'état du serveur ne l'empêche pas de traiter rapidement et convenablement les demandes d'authentification en cours. Cela provoquerait un possible refus de toute autre connexion entrante, légitime ou non. Le **nettoyeur** entrerait donc en action pour décharger le serveur d'authentification. Bien entendu, il est possible qu'une demande d'authentification légitime soit interrompue, au profit de la libération de ressources du serveur.

Le deuxième scénario tout juste présenté représente un cas limite qui, idéalement, ne se produirait pas. D'après les résultats présentés sur les performances de l'utilisation de la mémoire, de l'utilisation du processeur et du temps d'authentification, en plus des techniques permettant de configurer le serveur d'authentification avec le fichier de configuration, on réalise que ce scénario ne se produirait que très rarement et que, malgré tout, les répercussions ne seraient pas critiques.

Que l'authentification d'un usager soit réussie ou non, les résultats démontrent que la prise en charge par le système est complète. Premièrement, la bonne décision est prise concernant l'acceptation sécuritaire d'une demande d'authentification. Deuxièmement, le module de base de données du serveur d'authentification enregistre correctement les différents événements se produisant pendant le processus d'authentification. Troisièmement, le module SystemLogger est en mesure d'enregistrer les alertes lorsqu'il se produit des erreurs critiques. Finalement, le client MédicIP est en mesure d'afficher des messages informatifs à l'utilisateur, en fonction des événements se produisant.

6.3 Analyse du processus d'authentification locale

Le processus d'authentification locale se voulait une alternative permettant d'offrir une robustesse accrue à l'infrastructure. Comme le modèle de communication complètement maillé ne requiert pas la présence du serveur d'authentification pour établir et gérer les conférences entre les utilisateurs, il fallait s'assurer qu'une panne du serveur n'empêche pas les usagers d'utiliser le système.

Les résultats de l'annexe C ont permis d'observer le comportement du module d'authentification selon trois scénarios :

- authentification réussie ;
- authentification échouée (mauvaise clé privée) ;
- authentification échouée (historique local altéré).

Les résultats de ces trois tests permettent de mettre en évidence certaines conclusions. En premier lieu, les résultats démontrent le bon fonctionnement du module : il permet à un utilisateur de s'authentifier si le serveur d'authentification est en panne.

En deuxième lieu, les résultats permettent d'affirmer que cette authentification locale est sécuritaire grâce aux mesures de protection intégrées à ce module. Les données sensibles permettant cette authentification locale sont protégées. Des mécanismes permettent de plus la détection d'anomalies et d'irrégularités, ainsi que l'enregistrement d'événements.

En troisième lieu, on remarque cependant une limitation du système. En effet, pour être utilisable, cette méthode d'authentification locale requiert qu'un usager se soit préalablement authentifié sur le client. Comme il s'agit d'un module complémentaire au système d'authentification principal, cette limitation est acceptable.

Une autre alternative aurait été de mettre à jour périodiquement une seule et unique base de données locale, sur chacun des clients. Cette façon de faire aurait permis d'avoir un système d'authentification locale plus uniforme, en supposant que les clients soient toujours en ligne, ce qui n'est pas toujours le cas dans le modèle d'utilisation envisagé.

6.4 Analyse des conférences multimédias sécurisées

Afin de corriger les failles du prototype MédicIP, le processus de sécurisation s'étend jusqu'aux conférences multimédias. L'annexe D présente plusieurs tests permettant d'analyser la sécurité et la performance des aspects suivants :

- création d'une conférence multimédia sécurisée ;
- échec de la création d'une conférence multimédia sécurisée dû à l'utilisation d'un certificat *root* non autorisé ;
- échec de la création d'une conférence multimédia sécurisée dû à l'utilisation d'un certificat client non autorisé ;
- temps requis pour l'établissement d'une conférence sécurisée par rapport à une conférence non sécurisée ;
- bande passante requise pour l'établissement d'une conférence sécurisée par rapport à une conférence non sécurisée ;
- bande passante, et durée requises, pour l'envoi d'ecgML lors d'une conférence sécurisée, par rapport à une conférence non sécurisée.

Voici un tableau regroupant l'ensemble des résultats des tests de performance reliés à la création et au déroulement de conférences multimédias sécurisées :

Comparatif	MédicIP	SécureMédic	Constat
Temps requis pour établir une conférence (sec)	4.47	7.52	MédicIP ~40% plus rapide
Bande passante requise pour établir une conférence (octets)	8065	14647	MédicIP ~45% moins de bande passante
Temps requis pour l'envoi des ecgML (sec)	0.54	4.35	MédicIP ~88% plus rapide
Bande passante requise pour l'envoi des ecgML (octets)	171 874	194 748	MédicIP ~12% moins de bande passante
Vitesse de transfert pour l'envoi des ecgML (octets / sec)	318 285	44 770	MédicIP ~85% plus rapide
Temps requis pour une conférence complète (sec)	52.48	54.08	MédicIP ~2% plus rapide
Temps requis pour l'envoi des données audios (sec)	3.11	3.25	MédicIP ~5% plus rapide
Bande passante requise pour l'envoi des données audios (octets)	30 070	30 131	MédicIP ~1% moins de bande passante
Vitesse de transfert pour l'envoi des données audios (octects / sec)	9688	9344	MédicIP ~4% plus rapide

Figure 6.1 Comparatif global des tests effectués.

Les sections suivantes aborderont la manière dont ces tests ont été effectués, et présenteront les conclusions tirées de ceux-ci.

6.4.1 Création d'une conférence multimédia sécurisée

L'annexe D.1 présente une comparaison au niveau de la création d'une conférence (signalisation SIP) sécurisée par rapport à une conférence non sécurisée.

Les captures de trafic réseau illustrent l'apport qu'offre l'utilisation du protocole TLS lors de la signalisation SIP. Utilisé seul, le protocole SIP est sujet à plusieurs menaces (RFC 3261, section 26 [Rosenberg *et al.*, 2002]), situation qui est inexistante lorsque les échanges SIP s'effectuent dans un canal de communication sécurisé, offert par le protocole TLS.

Du point de vue de la sécurité, l'utilisation de "SIP over TLS" qu'offre le projet SécureMédic permet d'adresser la portion qui concerne l'établissement d'une conférence de manière sécuritaire. Il existe par contre certaines situations où les conditions nécessaires à l'utilisation de "SIP over TLS" ne sont pas complètement respectées, entraînant ainsi certains risques. Les sections qui suivent aborderont certaines de ces situations, afin de valider si elles sont bien gérées.

6.4.2 Échec de la création d'une conférence multimédia sécurisée

Afin de permettre un fonctionnement optimal du protocole SIP avec TLS, une certaine configuration des librairies doit être effectuée (voir chapitre D.1.1).

Que ce soit de manière volontaire ou involontaire, il peut arriver que certains aspects de cette configuration ne soient pas valides. Les annexes D.2 et D.3 présentent les cas où le certificat client, ou le certificat *root*, est invalide, c'est-à-dire qu'il ne fait pas partie de l'infrastructure à clés publiques.

Ces tests ont permis de prouver que l'implémentation du système développé respecte bien les contraintes requises par le protocole TLS, qui constitue l'unique barrière protégeant le protocole SIP des risques qui lui sont associés. Dans les deux cas, le *handshake* TLS décèle l'anomalie dans les certificats utilisés et empêche tout échange SIP de manière non sécurisée.

6.4.3 Performance lors de l'établissement d'une conférence multimédia sécurisée

Bien que les sections précédentes démontrent que le processus de signalisation des conférences est sécuritaire, il n'en demeure pas moins qu'il a un coût, dû aux calculs et aux échanges réseau supplémentaires requis par le protocole TLS.

L'annexe D.4.1 présente les résultats venant d'un banc de tests quantifiant les temps requis pour l'établissement d'une conférence multimédia sécurisée par rapport à une non sécurisée. L'application MédicIP originale prend en moyenne 4.47 secondes pour effectuer la signalisation, tandis que l'application MédicIP sécurisée en prend 7.52 en moyenne. En prenant comme temps de référence l'application sécurisée, on peut affirmer que le prototype MédicIP initial était 40,56% plus rapide.

L'annexe D.4.2, quant à elle, présente les résultats de tests quantifiant la bande passante requise pour l'établissement d'une conférence sécurisée par rapport à une conférence non sécurisée. Le prototype MédicIP requiert en moyenne 8065 octets tandis que le prototype sécurisé en requiert en moyenne 14647. En prenant comme bande passante de référence l'application sécurisée, on peut affirmer que le prototype MédicIP initial demandait 44,94% moins de bande passante.

6.4.4 Performance lors des échanges d'électrocardiogrammes durant une conférence multimédia sécurisée

Lors de transferts d'électrocardiogrammes, on remarque des impacts au niveau de la performance, dus à la sécurisation du prototype. L'annexe D.4.3 présente les résultats des tests comparatifs lors d'échanges des ecgML au niveau des temps de transfert, ainsi qu'au niveau de la bande passante requise.

Pour ce qui est du transfert des électrocardiogrammes, l'application MédicIP nécessite en moyenne une durée cumulative de 0.54 seconde, comparativement à 4.35 secondes pour l'application SécureMédic, pour envoyer une séquence de cinq ecgML. En prenant comme

durée de référence l'application sécurisée, on peut affirmer que l'application MédicIP prend 87,59% moins de temps.

Pour ce qui est de la bande passante requise lors des transferts d'électrocardiogrammes, l'application MédicIP nécessite en moyenne 171 874 octets, comparativement à 194 748 octets pour l'application SécureMédic. En prenant comme bande passante de référence celle de l'application sécurisée, on peut affirmer que l'application MédicIP prend 11,75% moins de bande passante.

On remarque que pour une taille relativement similaire (écart de 11.75%), on obtient des temps d'envoi très divergents (écart de 87.59%). Comme l'envoi sécurisé se fait à l'aide des protocoles TCP et TLS, comparativement à UDP (*User Datagram Protocol*) pour l'envoi non sécurisé, il est normal qu'on observe des différences au niveau des temps requis pour la transmission des ecgML. Ce sont principalement les délais engendrés par le chiffrement et le déchiffrement des données qui provoquent ces écarts.

Ce constat s'observe aussi lorsqu'on s'attarde à la vitesse de transfert moyenne. On remarque que la relation, entre la quantité de données à envoyer ainsi que le temps requis pour faire cet envoi, n'est pas proportionnelle. La vitesse de transfert moyenne pour l'application MédicIP est de 318 285 octets par seconde, comparativement à 44 770 octets par seconde pour l'application sécurisée. En prenant comme vitesse de référence celle de SécureMédic, on peut affirmer que l'application MédicIP a une vitesse 85,93% plus grande. Comme les tests ont été effectués sur le même réseau local, cette statistique permet de mettre en évidence l'impact du chiffrement des données et de la surcharge causée par le protocole TLS.

L'annexe D.4.3 présente aussi la durée totale de la conférence, qui est représentée par le temps écoulé entre le début et la fin du transfert des ecgML (incluant la signalisation SIP initiale). On remarque que la durée totale moyenne de la conférence pour l'application MédicIP est de 52,48 secondes, comparativement à 54,08 secondes pour l'application SécureMédic. En prenant comme durée moyenne de référence celle de l'application sécurisée, on peut affirmer que l'application MédicIP prend 2,4% moins de temps.

On remarque des écarts importants entre les comparaisons pour la durée totale de conférence ainsi que pour la durée des transferts d'électrocardiogrammes. Bien que la durée totale de la conférence inclus le temps nécessaire à la signalisation initiale, celle-ci dépend surtout du fonctionnement interne du prototype MédicIP. Voici une représentation graphique caricaturée qui illustre, dans le temps, l'envoi des ecgML pour chacune des plateformes :

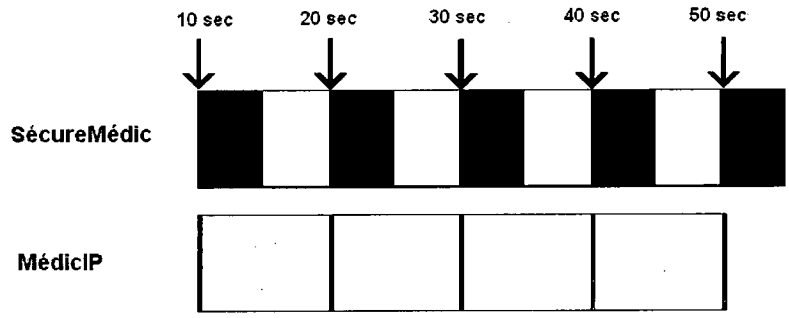


Figure 6.2 Transfert des ecgML dans le temps pour chacune des plateformes.

On remarque que l'envoi des ecgML se fait de manière périodique. Chaque 10 secondes, l'envoi d'un ecgML débute. Ainsi, même si SécureMédic requiert davantage de temps pour la transmission d'une quantité semblable de données, chacun des transferts individuels se termine avant le début de l'envoi du prochain ecgML, expliquant ainsi la raison pour laquelle la durée globale d'une conférence n'est pas influencée de façon importante par le temps requis pour l'envoi d'un ecgML.

6.4.5 Performance lors des échanges vocaux durant une conférence multimédia sécurisée

Malheureusement, comme le protocole SRTP possède les mêmes entêtes que le protocole RTP, il est impossible de démontrer à l'aide de captures de trafic réseau que c'est bien le protocole SRTP qui est en action. Cependant, comme l'utilisation du protocole sécurisé SRTP, pour remplacer le protocole RTP, provoque des dégradations au niveau des performances, une analyse des résultats comparatifs permettra de porter un jugement sur cet aspect.

L'annexe D.4.4 présente les résultats des tests effectués sur les applications MédicIP et SécureMédic. On remarque que la durée moyenne pour transférer les données audio est de 3,25 secondes pour l'application SécureMédic, comparativement à 3,11 secondes pour MédicIP. En prenant comme référence la durée de l'application SécureMédic, on peut affirmer que l'application MédicIP requiert 4,30% moins de temps pour transférer le contenu audio préenregistré.

Les résultats démontrent de plus que la bande passante requise pour ce même transfert est d'en moyenne 30 070 octets pour MédicIP, comparativement à 30 131 octets pour SécureMédic. En prenant comme référence la bande passante requise de l'application

SécureMédic, on peut affirmer que l'application MédicIP requiert 0,8% moins de bande passante.

Pour un écart de moins de 0,8% de bande passante requise, on remarque un écart de 4,30% pour le temps requis. Cet écart se justifie par le temps nécessaire pour le chiffrement des données. Pour l'application SécureMédic, la vitesse de transfert moyenne est d'environ 9344 octets par seconde, comparativement à 9688 octets par seconde pour MédicIP. En prenant comme référence la vitesse de transfert de SécureMédic, on peut affirmer que l'application MédicIP est 3,55% plus performante au niveau de la vitesse de transfert. Encore une fois, cela démontre l'impact de la sécurisation sur le plan des performances.

6.4.6 Conclusion sur les conférences multimédias sécurisées

Bien que l'objectif initial visait la sécurisation du système, il faut néanmoins porter une certaine attention aux impacts qu'elle entraîne. Les résultats précédents permettent d'affirmer que la sécurisation du système occasionne des baisses de performance plus ou moins importantes selon les modules impliqués.

Il faut garder en tête que tous ces tests ont été effectués à l'aide d'ordinateurs équipés de processeurs Core 2 Duo E6750 2.66Ghz d'Intel sur un réseau local. Au moment d'écrire ce mémoire, ce type d'ordinateur est beaucoup plus puissant que les appareils mobiles sur lesquels l'application MédicIP fonctionnera dans le futur. De plus, les vitesses de transmission des réseaux locaux sont beaucoup plus élevées que celles qu'il est possible d'atteindre sur Internet.

Ceci étant dit, les impacts sur la performance nous amènent à nous questionner sur la faisabilité d'un tel système sécurisé, qui utilise en plus un modèle de conférence complètement maillé. Les tests présentés ont tous été effectués avec deux participants dans une conférence. Il avait été établi par l'équipe de développement de l'application MédicIP que le prototype pouvait supporter jusqu'à 10 participants sans connaître de problème de performance, et ce, sur un réseau local. Le comparatif global nous porte donc à croire que les impacts de performance reliés à la sécurisation de MédicIP auront comme effet de supporter un moins grand nombre de participants dans une conférence. Il suffit d'imaginer un client mobile envoyer un électrocardiogramme de manière sécurisée à cinq autres participants pour soupçonner un problème potentiel au niveau du déroulement fluide de la conférence. En effet, ces transferts impliqueraient cinq chiffrements simultanés des mêmes données, ce qui est exigeant pour un appareil mobile.

6.5 Synthèse globale

L'ensemble des analyses présentées confirment la sécurisation de l'application MédicIP au niveau de l'authentification usager, de la création de conférences ainsi que des échanges de données durant ces conférences. Les tests comparatifs démontrent une tendance qui était prévisible : la sécurisation d'une application engendre des pertes de performances.

Au niveau de la création des conférences, la perte de performance n'est pas négligeable, mais n'a pas de conséquences sur le bon fonctionnement de l'application. En effet, comme la signalisation se déroule uniquement en début de conférence, un écart de temps d'environ 40% n'est pas dramatique.

L'impact sur les performances au niveau de l'envoi des ecgML est beaucoup plus significatif que pour l'établissement des conférences. Les écarts relatifs à la durée de transmission ainsi qu'à la vitesse de transfert moyenne se situent au-delà des 85%. Malgré tout, le déroulement des conférences s'en trouve peu affecté étant donné l'envoi de manière périodique des données. C'est donc le fonctionnement interne de l'application MédicIP qui modère les impacts sur la performance, car si on ne se fiait qu'aux performances brutes, il y aurait certainement une problématique.

Les communications audios constituent les échanges les moins affectés par les pertes de performance. L'utilisation de la librairie Speex, pour la compression des données avant la transmission, permet de réduire grandement la taille des données, ce qui provoque une augmentation directe des performances. L'utilisation d'un algorithme de compression pour les données des ecgML pourrait aussi produire une amélioration des performances.

L'ensemble de ces tests a été réalisé dans les conditions idéales : processeur performant, deux seuls participants, et utilisation d'un réseau local. Une deuxième phase de tests permettrait de confirmer ou d'infirmer les craintes que les comparatifs soulèvent. Cette deuxième phase de tests se voudrait plus représentative de l'utilisation normale du système. La présence d'au moins trois participants, sur des appareils mobiles sans-fil et utilisant Internet comme canal de transmission, permettrait d'avoir l'heure juste sur le fonctionnement réel de SécureMédic.

En terminant, voici le tableau final du résumé des failles de sécurité du prototype MédicIP, mis à jour avec les différentes solutions implémentées :

Failles de sécurité identifiées	Solutions envisagées	Solutions implémentées
Communication non sécurisée lors du processus d'authentification d'un usager	Création d'une infrastructure à clés publiques permettant la validation de l'identité des entités, ainsi que la création d'un canal de communication sécurisé	Serveur d'authentification dédié, infrastructure à clés publiques, système d'archivage d'événements, protection contre certains types de déni de service, module d'authentification locale, module de protection des trusted certificates.
Signalisation SIP non sécuritaire lors de la création d'une conférence	Utilisation du protocole TLS pour effectuer les échanges SIP de manière sécuritaire	Utilisation des clés et certificats de l'infrastructure à clés publiques afin de configurer la pile logicielle M5T en mode « SIP over TLS ».
Communication audio non sécuritaire	Utilisation du protocole sécurisé SRTP	Utilisation du protocole SRTP de la pile logicielle M5T.
Échange d'électrocardiogrammes non sécuritaire	Utilisation du protocole TLS pour effectuer les échanges de données de manière sécuritaire	Utilisation d'un contexte SIP sécurisé afin de transmettre les données.
Entreposage non sécurisé du carnet d'adresses	Utiliser un carnet d'adresses centralisé	Aucune

Figure 6.3 Failles de sécurité du prototype MédicIP.

CHAPITRE 7

CONCLUSION

7.1 Synthèse finale

La réalisation, en 2007, du prototype d'application de télémédecine MédicIP, a soulevé une question d'ordre moral et technique à la fois :

> Les mécanismes et outils actuels des technologies de l'information permettent-ils de produire un environnement sécurisé pour une application de télémédecine utilisant Internet comme moyen de communication ?

Les risques encourus par l'utilisation d'une application dans le domaine de la santé demandent qu'on porte attention à cette question. Pour tenter d'y répondre, une analyse du prototype MédicIP a été effectuée afin de déterminer deux aspects importants : quelles sont les failles de sécurité présentes dans MédicIP, et quels risques représentent-elles ?

Les réponses à ces questions ont permis de répondre aux objectifs initiaux suivants :

- développer un système d'authentification usager facile d'utilisation et sécuritaire ;
- permettre des échanges sécurisés (voix et données) entre les différents utilisateurs d'une conférence multimédia ;
- entreposer de manière sécuritaire les données sensibles utilisées par l'application de télémédecine.

Pour être en mesure d'atteindre ces objectifs, une étude de l'état de l'art a été menée afin d'évaluer les technologies existantes permettant de corriger les failles de sécurité du prototype MédicIP.

Parmi les technologies phares retenues, le protocole TLS occupe une place importante. Jumelé à divers algorithmes cryptographiques, ce protocole a servi de base dans l'ensemble du projet SécureMédic. Il a, en effet, été grandement utilisé afin de mettre sur pied une infrastructure à clés publiques permettant l'authentification sécuritaire des usagers, ainsi que pour sécuriser la signalisation des conférences multimédias, en plus du transfert d'électrocardiogrammes durant celles-ci.

L'analyse des résultats des différents tests a permis d'affirmer clairement qu'il était possible de développer un système d'authentification usager simple d'utilisation et sécuritaire à l'aide d'une infrastructure à clés publiques. Les tests ont aussi démontré qu'il était possible de sécuriser des conférences multimédias tant au niveau de l'établissement de celles-ci que lors des échanges d'informations, et ce, grâce à des protocoles comme SIP/TLS et SRTP.

Le développement d'une méthode d'authentification alternative, quant à elle, démontre la faisabilité de l'entreposage sécuritaire de données sensibles utilisées par une application de télémédecine.

Du point de vue de la sécurité, le constat est clair : le projet SécureMédic a permis de corriger les failles de sécurité critiques du prototype MédicIP. Cependant, les résultats ont démontré certains dommages collatéraux causés par cette sécurisation, notamment en ce qui concerne la performance du système. En ce qui a trait au processus d'authentification usager, de même qu'à l'établissement sécuritaire de conférences multimédias, les résultats indiquent une perte de performance significative mais non critique, étant donné qu'il s'agit d'opérations ponctuelles. Par contre, pour ce qui est du transfert sécurisé des électrocadriogrammes et de la voix, on remarque la présence d'un goulot d'étranglement.

Il a été démontré par l'équipe de développement de MédicIP que le prototype initial pouvait supporter jusqu'à dix participants dans une conférence multimédia sur un réseau local, en utilisant des ordinateurs de bureau, après quoi les limitations reliées à la bande passante et à l'utilisation du processeur se font trop ressentir. Donc, dans un contexte d'utilisation réel, c'est-à-dire sur Internet, avec des appareils mobiles de plus faibles puissances et en utilisant des échanges sécurisés, il serait plausible de croire que ce nombre maximal de participants chuterait probablement de façon importante.

Ce goulot d'étranglement est en partie causé par l'utilisation du mode de conférence complètement maillé. Bien que ce type de conférence apporte un niveau de robustesse indéniable, les conférences typiques utilisant l'approche client-serveur sont néanmoins efficaces et fiables. Dans un développement futur, l'utilisation de conférences complètement maillées pourra possiblement être remise en question.

7.2 Contributions

La contribution la plus significative de ce projet aura été la mise en oeuvre d'une architecture d'authentification usager. L'intégration de plusieurs technologies comme les systèmes d'exploitation Linux, les bases de données, les infrastructures à clés publiques, les cer-

tificats numériques, les protocoles de communication sécurisés comme TLS, ainsi que la cryptographie symétrique et asymétrique ont contribué à la création d'une architecture sécuritaire et facile d'utilisation. L'intégration de mesures de protection comme le nettoyeur rajoute de plus une dimension intéressante pour ce qui est de la robustesse du système.

Le développement d'un module client conçu spécifiquement pour le serveur d'authentification a été fait de manière générique et complètement indépendante, permettant ainsi à n'importe quelle application, médicale ou non, de pouvoir utiliser le système dans sa totalité et de profiter de l'ensemble des services offerts (authentification distante et locale, système d'archivage d'événements). L'ensemble de l'infrastructure d'authentification est un projet individuel en soit, qui a été intégré au prototype MédicIP, démontrant ainsi sa capacité à être utilisé dans n'importe quel contexte.

Peut-on répondre à la question de recherche de manière affirmative ? La réponse est oui. Cela ne signifie pas pour autant que le projet global de MédicIP, incluant SécureMédic, ne soit terminé.

7.3 Travaux futurs

7.3.1 Persistence des données et sources des électrocardiogrammes

Le prototype actuel utilise des électrocardiogrammes préenregistrés sous la forme de fichiers XML. Dans le cadre d'une version finale de l'application MédicIP, il serait indispensable de créer un module qui soit le plus flexible possible, permettant d'accepter les électrocardiogrammes d'un véritable appareil médical.

Pour des raisons de sécurité et de pratique médicale, les électrocardiogrammes reçus par l'application MédicIP devraient être enregistrés localement, et ce, de manière sécuritaire.

7.3.2 Migration vers une plateforme mobile

Le prototype MédicIP peut actuellement être utilisé sur les systèmes d'exploitation Windows XP ainsi que Windows Vista. Avec la popularité grandissante des téléphones intelligents (*smart phones*) et la disponibilité des réseaux sans-fils Internet hautes vitesses, il est nécessaire que MédicIP soit migré vers un système d'exploitation fonctionnant sur les appareils mobiles et, si possible, dans un langage de programmation différent du C#

actuellement utilisé. En effet, ce langage est très lié aux plateformes Windows et est donc plus susceptible d'être limité à cet environnement, même s'il n'est pas exclusif à celui-ci.

Outre la popularité de ces appareils mobiles, il s'agit d'un aspect important sur lequel les spécialistes de la santé mettent de l'emphase. Dans le but d'avoir le plus haut taux d'acceptation possible d'un outil comme MédicIP, il est primordial que l'application ne fonctionne pas uniquement sur des ordinateurs de bureau ou des ordinateurs portables.

7.3.3 Protocole MIKEY

Dans le chapitre 5.2.2, il était question de l'implémentation du protocole SRTP. L'utilisation d'une clé incorporée à même le logiciel (*hardcode*) a été faite pour permettre l'utilisation du protocole SRTP.

Due à des contraintes de temps et d'expertise, c'est cette solution qui fait partie du projet SécureMédic final. Dans une prochaine itération du projet, il serait intéressant d'y inclure le protocole MIKEY.

MIKEY est un protocole de gestion des clés qui est utilisé dans les applications multimédias temps réel. Il est particulièrement utilisé pour le partage de clés cryptographiques permettant de sécuriser des sessions multimédias de type SRTP.

7.3.4 Service DNS sécurisé

Au chapitre 4.1.4, il était question d'une limitation causée par l'utilisation d'adresses IP statiques. Il serait intéressant, dans une itération future, d'ajouter au client SécureMédic, plus précisément dans la portion de la négociation SIP, un module permettant d'accéder de manière sécurisée à un service DNS, de manière à ce que les certificats numériques puissent avoir comme *common name* un nom d'hôte (*hostname*) au lieu d'une adresse IP statique. De cette façon, il serait possible d'utiliser un système sur un réseau où les adresses IP sont attribuées de manière dynamique.

7.3.5 Carnet d'adresses centralisé

Au chapitre 2.3.3, il était question d'une problématique où le client utilise un carnet d'adresses, qui n'est pas sécurisé, de manière locale. Bien qu'il serait possible d'utiliser un système de chiffrement pour protéger le contenu du carnet d'adresses, la solution optimale serait d'utiliser un carnet d'adresses de manière distante, afin d'avoir accès à ses contacts peu importe le client utilisé.

Cette fonctionnalité pourrait être ajoutée dans une version future du serveur de présence.

7.4 Pour terminer

L'avènement des applications de télémédecine nous permet de constater une collision entre deux mondes très présents dans notre société : la santé et les technologies de l'information. La rencontre de ces deux champs d'expertise permet de tirer le maximum de chacun, et d'ainsi offrir à la population une nouvelle gamme de services.

Comme ce mémoire le démontre, plusieurs considérations doivent être prises en compte pour produire une application de télémédecine sécuritaire et performante. Les cas du prototype MédicIP et du projet SécureMédic sont des exemples intéressants d'avancées dans le domaine. Bien qu'il soit encore trop tôt pour envisager de déployer dès maintenant le système actuel, il s'agit d'un pas dans la bonne direction.

ANNEXE A

Indice de performance du serveur d'authentification

Il est possible de mesurer la performance du serveur d'authentification d'après les paramètres suivants :

- utilisation de la mémoire ;
- temps requis par le processus d'authentification ;
- utilisation du processeur.

Les différents tests de ce chapitre ont été réalisés à l'aide du serveur d'authentification tournant sur la distribution Linux Ubuntu 9.04 x64, équipé d'un processeur Core 2 Duo E6750 d'Intel cadencé à 2.66Ghz, et muni de 4 GB de mémoire vive.

A.1 Utilisation de la mémoire

Trois types de test ont été effectués pour quantifier l'utilisation de la mémoire :

1. authentification valide ;
2. authentification échouée (mauvais mot de passe) ;
3. attaque socket passive (DoS).

Pour chacun de ces tests, dix-neuf authentifications ont été faites. Voici les résultats :

Authentification	Test 1: authentification réussie		Test 2: authentification échouée		Test 3: connection socket passive	
	Mémoire utilisée (Ko)	Augmentation de la mémoire utilisée (Ko)	Mémoire utilisée (Ko)	Augmentation de la mémoire utilisée (Ko)	Mémoire utilisée (Ko)	Augmentation de la mémoire utilisée (Ko)
0	3044		3040		3040	
1	3364	***320***	3360	***320***	3120	***80***
2	3444	80	3440	80	3224	***104***
3	3532	88	3528	88	3300	76
4	3620	88	3612	84	3372	72
5	3704	84	3700	88	3448	76
6	3788	84	3784	84	3524	76
7	3876	88	3872	88	3596	72
8	3960	84	3960	88	3668	72
9	4044	84	4044	84	3744	76
10	4128	84	4128	84	3816	72
11	4212	84	4216	88	3892	76
12	4296	84	4300	84	3968	76
13	4380	84	4384	84	4040	72
14	4468	88	4468	84	4112	72
15	4552	84	4552	84	4188	76
16	4640	88	4636	84	4260	72
17	4728	88	4720	84	4336	76
18	4812	84	4804	84	4412	76
19	4896	84	4888	84	4484	72
Moyenne 1		97,47		97,26		76
Médiane 1		84		84		76
Écart-type 1		53,93		53,98		7,18
Moyenne 2		85,11		84,89		74,12
Médiane 2		84		84		76
Écart-type 2		2,30		2,19		2,06

Figure A.1 Trois types d'authentification entre le client SécureMédic ainsi que le serveur d'authentification ont été testés afin de quantifier l'utilisation de la mémoire par le serveur d'authentification lors du processus d'authentification.

La moyenne, la médiane et l'écart-type numéro 2 sont calculés sans les valeurs en caractères gras. Ces valeurs sont considérées aberrantes, car elles ne reflètent pas le fonctionnement normal du serveur d'authentification, qui requiert quelques itérations avant de stabiliser son comportement.

Ces données ont été calculées à l'aide de l'utilitaire "top", disponible sous les systèmes Linux et Unix.

A.2 Temps requis pour les authentifications

Afin de déterminer le temps requis par le serveur pour effectuer une authentification, l'utilitaire "top" a été utilisé. Le champ "TIME+" de cet outil permet de connaître le temps processeur utilisé par l'application.

Comme le serveur d'authentification n'utilise aucune ressource processeur quand il est en attente de connexions entrantes, le temps processeur cumulatif correspond exactement au temps requis pour effectuer les authentifications.

Voici les résultats obtenus :

Authentification	Temps processeur cumulatif requis (sec)	Différence (sec)
0	0.06	
1	0.11	0.05
2	0.14	0.03
3	0.18	0.04
4	0.21	0.03
5	0.25	0.04
6	0.29	0.04
7	0.33	0.04
8	0.40	0.07
9	0.44	0.04
10	0.48	0.04
11	0.53	0.05
12	0.57	0.04
13	0.60	0.03
14	0.64	0.04
15	0.68	0.04
16	0.71	0.03
17	0.76	0.05
18	0.79	0.03
19	0.84	0.05
20	0.87	0.03
21	0.92	0.05
22	0.96	0.04
23	1.00	0.04
24	1.03	0.03
25	1.07	0.04
26	1.12	0.05
27	1.16	0.04
28	1.20	0.04
29	1.25	0.05
30	1.28	0.03
31	1.32	0.04
32	1.37	0.05
33	1.41	0.04
34	1.45	0.04
35	1.50	0.05
36	1.54	0.04
37	1.58	0.04
38	1.62	0.04
39	1.64	0.02
40	1.68	0.04
41	1.73	0.05
42	1.76	0.03
43	1.80	0.04
44	1.83	0.03
Moyenne		0.04
Médiane		0.04
Écart-type		0.01

Figure A.2 Temps requis par le serveur d'authentification pour traiter des authentifications de type usager.

A.3 Utilisation du processeur

Lorsque le serveur d'authentification est en attente de connexions entrantes, son utilisation du processeur est nulle. Lors d'une authentification, l'utilisation processeur grimpe à 100%, le temps que le serveur d'authentification traite la requête.

ANNEXE B

Résultats - Processus d'authentification à distance

Lors d'une authentification usager à distance, dont le processus est décrit au chapitre 5.1.1, plusieurs résultats ou comportements peuvent se produire en fonction du contexte de la demande d'authentification.

Cette annexe présentera les résultats pour les cas suivants :

- authentification réussie ;
- authentification échouée (mauvais certificat client) ;
- authentification échouée (mauvais *Trusted Certificate*) ;
- authentification échouée (arrêt par le nettoyeur) ;
- authentification échouée (arrêt par le compteur individuel) ;

Dans tous les résultats présentés, l'adresse IP 192.168.1.114 correspond au serveur d'authentification et les autres adresses IP correspondent à des clients MédicIP.

B.1 Authentification distante réussie

B.1.1 Configuration de la librairie OpenSSL

Afin de permettre des échanges sécurisés entre les entités, la librairie OpenSSL doit être configurée de manière à ce qu'un *handshake* TLS soit complété.

Voici la configuration utilisée pour le serveur d'authentification :

1. création d'un contexte TLS avec les méthodes "TLSv1_server_method" ;
2. assignation du jeu de chiffrement supporté (AES256-SHA) au contexte TLS ;
3. assignation du certificat serveur au contexte TLS ;
4. assignation de la clé serveur au contexte TLS ;
5. assignation du certificat root au contexte TLS (il s'agit du certificat serveur) ;
6. configuration du contexte TLS afin que le *handshake* TLS requière une authentification client (SSL_VERIFY_PEER | SSL_VERIFY_FAIL_IF_NO_PEER_CERT) ;

Voici la configuration utilisée pour le client :

1. création d'un contexte TLS avec les méthodes "TLSv1_client_method" ;
2. assignation du jeu de chiffrement supporté (AES256-SHA) au contexte TLS ;
3. assignation du certificat client au contexte TLS ;

4. assignation de la clé client au contexte TLS ;
5. assignation du certificat serveur au contexte TLS en tant que *trusted certificate*

B.1.2 Objectifs du test

L'objectif de ce test est de valider le fonctionnement sécuritaire de l'authentification à distance d'un usager, lorsque toutes les conditions nécessaires sont réunies. La configuration précédente a été utilisée durant le déroulement de ce test.

L'analyse de captures de trafic réseau et des entrées inscrites dans la base de données du serveur permettront de porter un jugement sur le déroulement de ce test.

B.1.3 Résultats

Lorsque l'authentification à distance d'un usager se produit correctement, il a été observé que l'application MédicIP démarre correctement. L'usager n'est par contre pas au fait de tout ce qui se produit à l'arrière-plan. Plusieurs paquets et messages sont échangés entre les deux entités. Voici une capture du trafic réseau se produisant lors d'une authentification réussie :

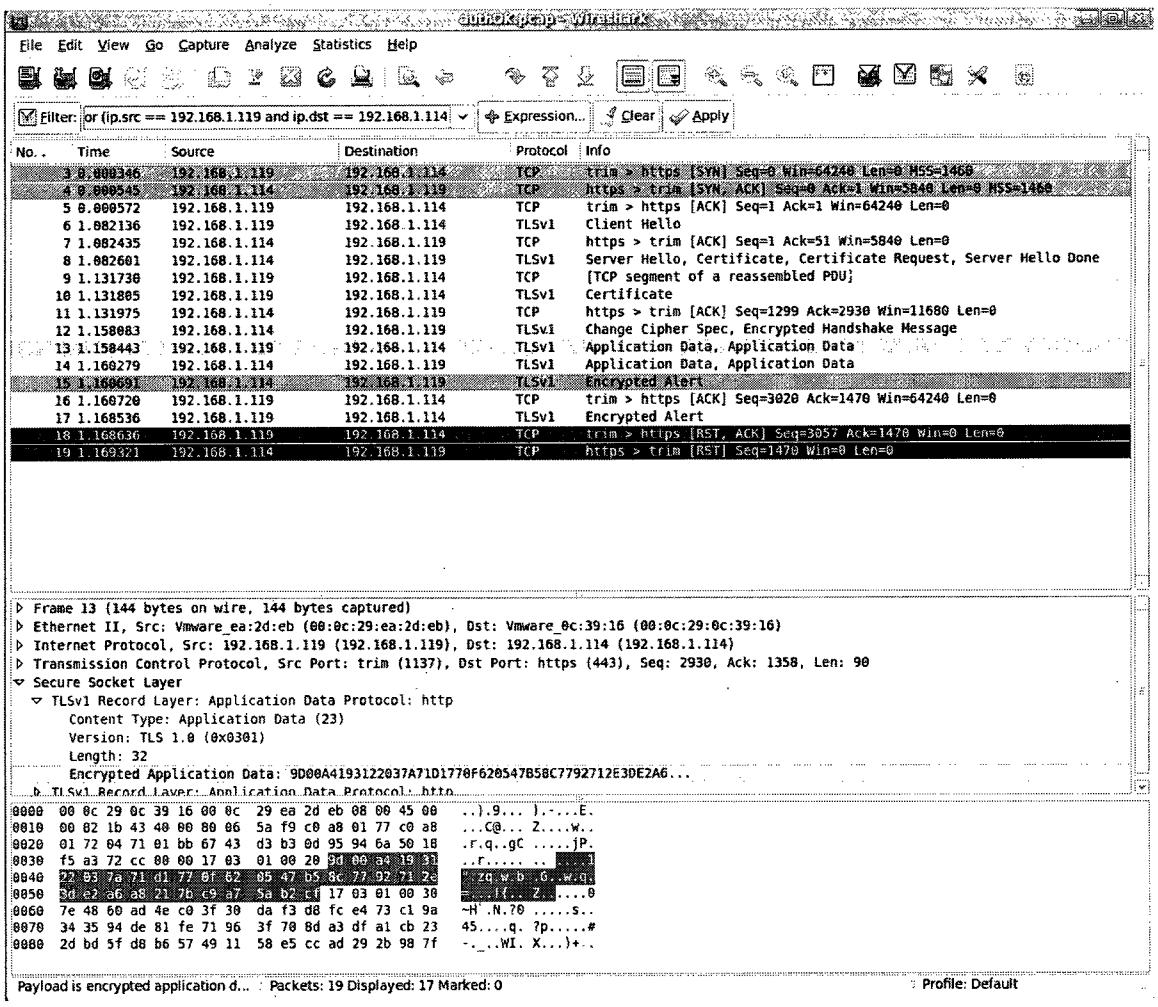


Figure B.1 Capture de trafic réseau réalisée avec Wireshark lors d'une authentification réussie entre le client MédicIP (192.168.1.119) et le serveur d'authentification (192.168.1.114). On remarque l'utilisation du protocole TLSv1 permettant la négociation d'un canal de communication sécurisé entre les deux entités, visible à la trame 13. On remarque les données chiffrées dans le champ *Encrypted Application Data* de la section *Secure Socket Layer*.

On remarque d'abord l'établissement de la connexion *socket* dans les trames 3 à 5. Par la suite, il se produit le *handshake* TLS dans les trames 6 à 12. On y distingue entre autre les échanges de certificats ainsi que les indications relatives à l'utilisation du jeu de chiffrement choisi.

Après la trame 12, le canal de communication est sécurisé, indiquant que l'authentification mutuelle entre le client et le serveur d'authentification a fonctionné.

La trame 13 correspond à l'envoi du nom d'usager et du mot de passe du client vers le serveur, qui se produit dans la deuxième phase de l'authentification d'un usager. Dans

les détails de la trame 13 (section du milieu de la capture d'écran), on constate que les données de l'application sont chiffrées (*Encrypted Application Data*).

La trame 14 correspond à la réponse de la demande d'authentification envoyée du serveur vers le client. Une fois de plus, on constate que les données sont envoyées via le protocole TLSv1, qui gère le chiffrement de celles-ci.

Les trames 15 et 17 représentent respectivement les alertes de fermeture de la connexion du serveur et du client.

Le module de base de données, présenté au chapitre 4.2.4, s'occupe de plus d'enregistrer l'événement. Dans ce scénario de test, il s'agit de l'entrée 292.

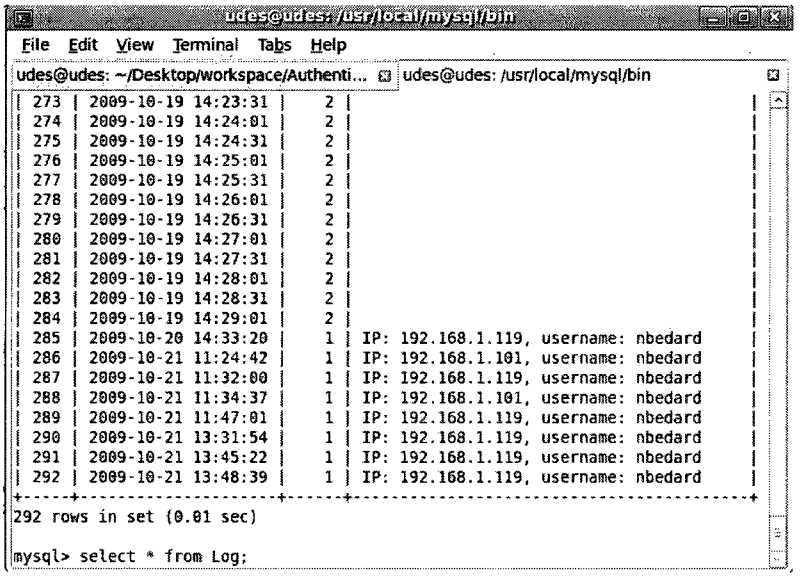


Figure B.2 Entrées de la base de données lors d'événements. La colonne du milieu contenant les chiffres 1 et 2 indique le type de l'événement (log). Par exemple, on remarque un événement de type 1 à l'entrée 292.

Les différents types d'événements sont présentés dans le tableau qui suit :

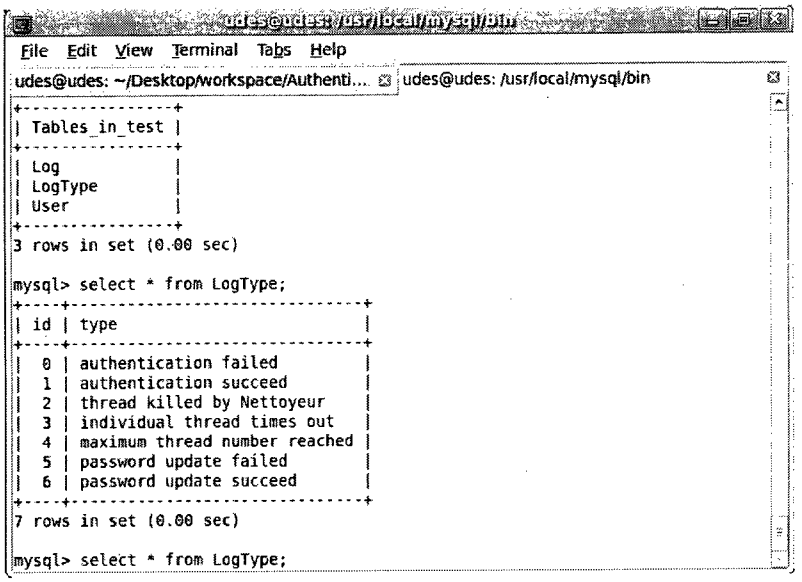


Figure B.3 Contenu de la table *LogType*. Tous les événements possèdent leur identificateur unique. Par exemple, l'événement de type 1 correspond à une authentification valide.

B.2 Authentification distante échouée - mauvais certificat client

B.2.1 Configuration de la librairie OpenSSL

La configuration utilisée pour le serveur d'authentification est identique à celle de la section précédente (B.1.1). Pour ce qui est de celle du client, le certificat et la paire de clés client ont été volontairement remplacés par des faux.

Par faux, on entend un certificat et une paire de clés valides du point de vue du format X509, mais ne faisant pas partie de l'infrastructure à clés publiques. Plus précisément, le certificat client n'est pas signé par le certificat *root*.

B.2.2 Objectifs du test

L'objectif de ce test est de valider le fonctionnement du serveur d'authentification lorsqu'une entité non autorisée tente de s'authentifier. L'architecture présentée propose que le *handshake* TLS entre le "faux" client et le serveur d'authentification échouera lorsque le client présentera son certificat dans le cadre de l'authentification mutuelle.

B.2.3 Résultats

Voici une capture du trafic réseau se produisant lors du test décrit ci-haut.

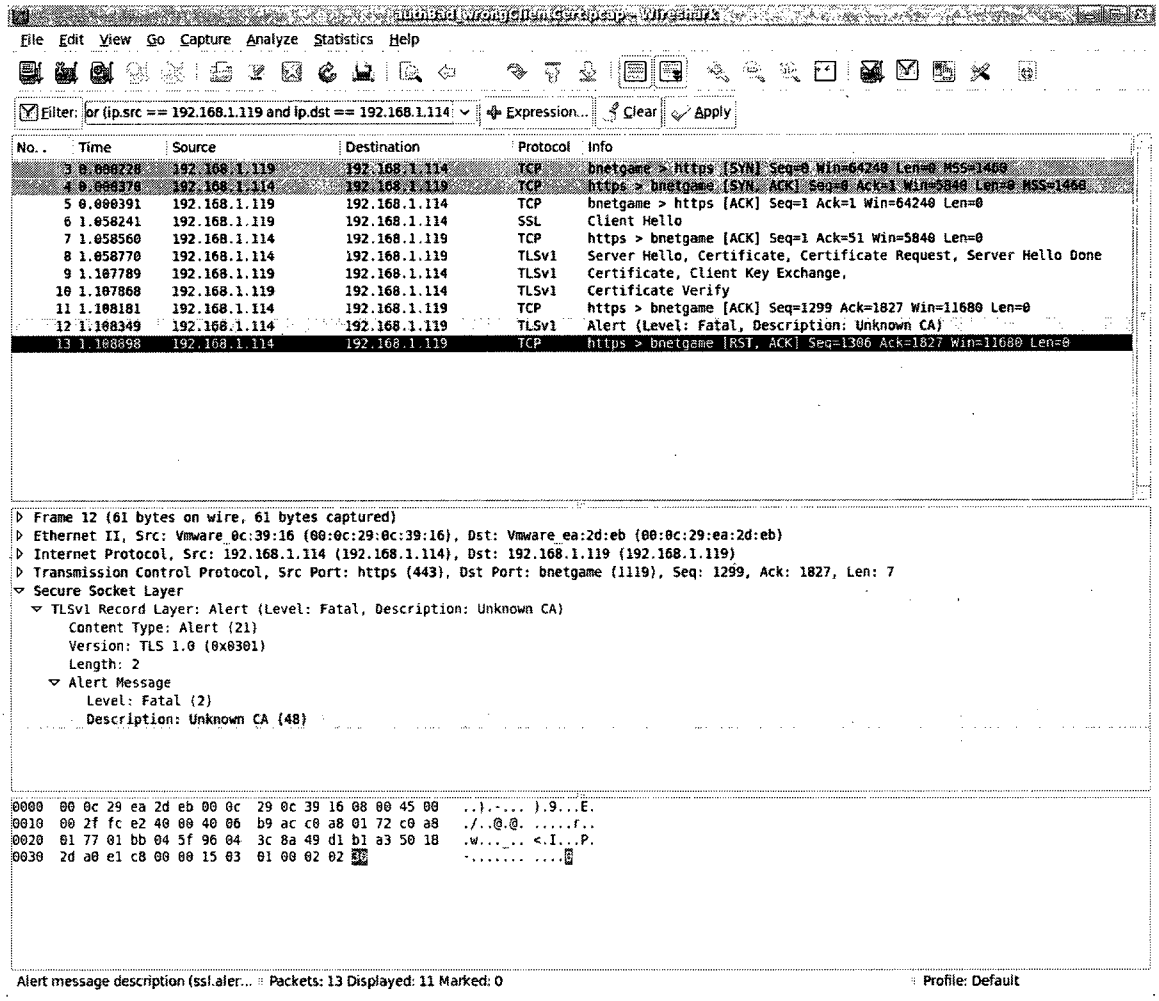


Figure B.4 Capture de trafic réseau réalisé avec Wireshark lors d'une authentification échouée due à l'utilisation d'un certificat client non autorisé.

Les trames 3 à 5 représentent encore une fois les échanges permettant l'établissement de la connexion socket entre le client et le serveur.

À partir de la trame 6, on remarque le début du *handshake* TLS, comme lors du scénario de la section précédente. Pendant l'authentification mutuelle, le serveur d'authentification effectue la validation du certificat client. Comme celui-ci n'est pas autorisé dans l'infrastructure à clés publiques, le *handshake* TLS échoue. On remarque cet échec à la trame 12, où le serveur envoie au client une alerte fatale indiquant que l'autorité de certification du certificat client est inconnue (voir détails de la trame 12 au milieu de la capture d'écran).

Au niveau du client, la réception de ce message engendre immédiatement l'affichage d'un message d'erreur à l'utilisateur. L'application MédicIP ne démarre évidemment pas lors de ce scénario.

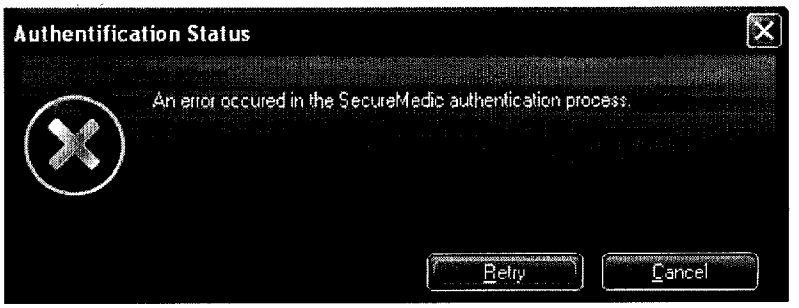


Figure B.5 Capture d'écran mentionnant à l'usager qu'il s'est produit une erreur durant le processus d'authentification.

Le module de base de données enregistre ce type de scénario de la façon suivante :

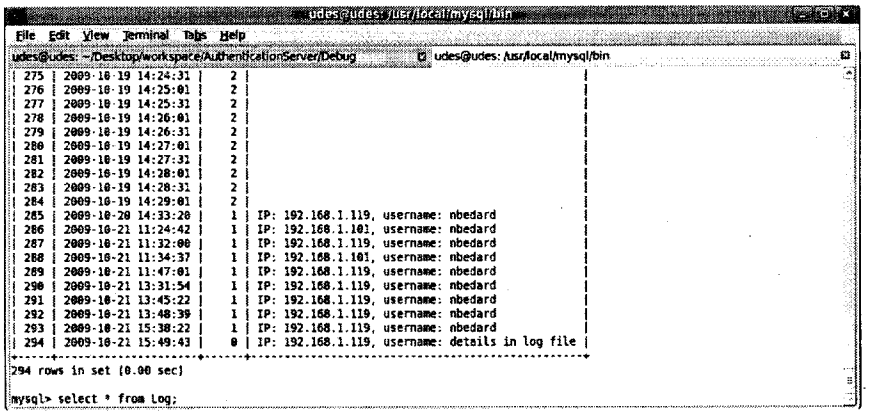


Figure B.6 Entrée dans la base de données lors d'une authentification échouée pour cause de certificat client non autorisé.

L'entrée de ce scénario correspond au numéro 294. On remarque un détail particulier dans les informations complémentaires fournies. Pour le *username*, on trouve le message *details in log file*. Le fichier de logs, géré par le module SystemLogger (décrit au chapitre 4.2.3), contient l'information suivante :

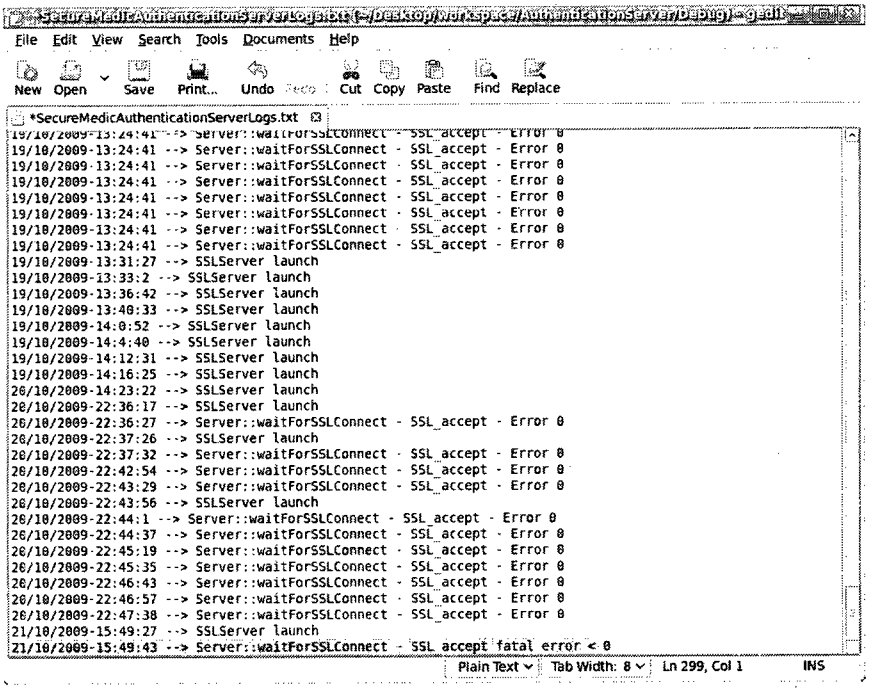


Figure B.7 Fichier de logs géré par le module SystemLogger au niveau du serveur d'authentification.

On peut se référer à l'heure et la date de l'entrée dans la base de données, soit le 21 octobre à 2009 15 :49 :43, afin de vérifier l'information contenue dans le fichier de logs. On remarque qu'il s'est produit une erreur lors de l'appel "SSL_accept", dans la fonction "waitForSSLConnect" de la classe Server. Évidemment, ce type d'information est très technique et est utile pour les développeurs. Dans ce cas-ci, on remarque qu'il s'agit d'un problème au niveau du *handshake* TLS.

B.3 Authentification distante échouée - mauvais *Trusted Certificate* client

B.3.1 Configuration de la librairie OpenSSL

La configuration utilisée pour le serveur d'authentification est identique à celle du premier test (chapitre B.1.1). Pour ce qui est de celle du client, le *trusted certificate* assigné au contexte TLS en est un arbitraire, ne faisant pas partie de l'infrastructure à clés publique.

B.3.2 Objectifs du test

L'objectif de ce test est de valider le fonctionnement de l'outil de protection du *trusted certificate*, le *TrustedCertificateProtector*, présenté au chapitre 4.3.4. Le scénario typique visé par ce test est celui où un intrus modifierait le *trusted certificate* sur l'hôte du client,

dans le but de créer un faux serveur d'authentification sur lequel le client MédicIP tenterait de s'authentifier, et ainsi, de voler des noms d'usager et des mots de passe. Afin de simuler ce scénario, le *trusted certificate* original a été remplacé par un faux.

B.3.3 Résultats

Lors de la tentative d'authentification sur le client MédicIP, le message d'erreur suivant s'est affiché :

Figure B.8 Capture d'écran mentionnant à l'usager qu'il s'est produit une erreur durant l'initialisation du module SécureMédic.

De l'information complémentaire est disponible dans le fichier de *logs* du client, géré par le module SystemLogger présenté au chapitre 4.3.3.

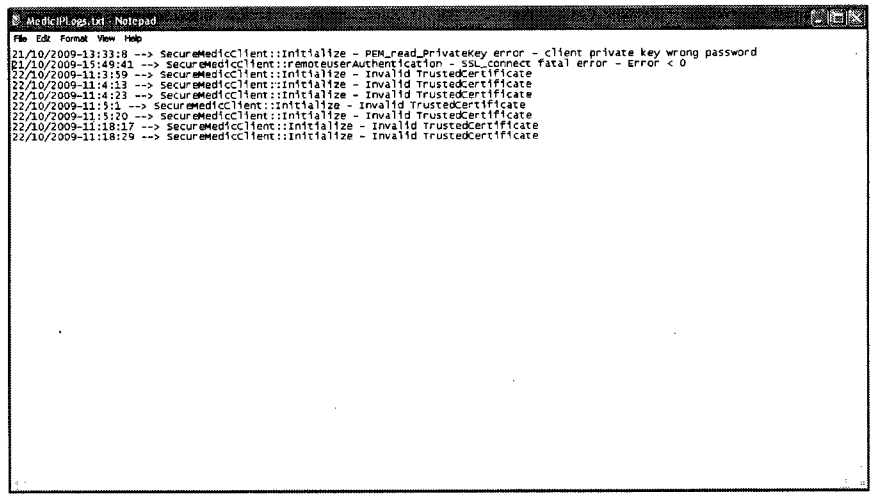


Figure B.9 Fichier de logs géré par le module SystemLogger au niveau du Client SécureMédic.

Comme la validation du *Trusted Certificate* échoue, le client n'envoie aucune requête au serveur d'authentification. Aucune capture de trafic réseau n'est donc requise pour ce test.

B.4 Authentification distante échouée - arrêt par le nettoyeur

B.4.1 Configuration de la librairie OpenSSL

Les configurations de la librairie initiale sont valides et identiques à celles du premier test (annexe B.1.1).

Le client possède par contre une modification temporaire pour permettre la réalisation de ce test. Afin de simuler une panne réseau, une surcharge du serveur d'authentification ou du client MédicIP, un délai a été introduit au niveau du client avant d'initialiser le *handshake*.

B.4.2 Objectifs du test

L'objectif de ce test est de valider le comportement du nettoyeur lors de la simulation d'une connexion socket passive, créée volontairement ou accidentellement. Le délai ajouté dans la configuration du client permettra de simuler ce type de connexion socket.

B.4.3 Résultats

Voici la capture de trafic réseau résultant de ce test :

est

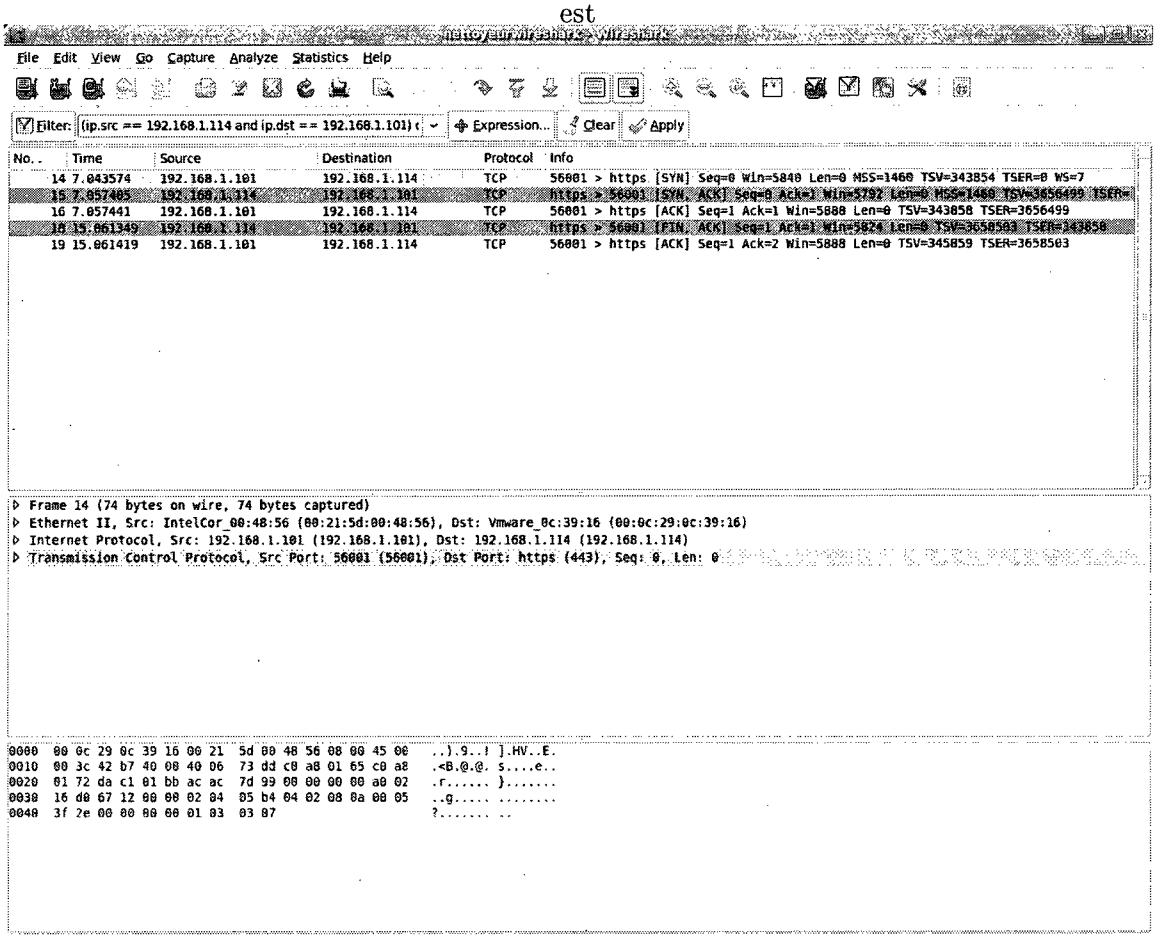


Figure B.10 Capture de trafic réseau réalisé avec Wireshark lors d'une authentification interrompue par le nettoyeur. Les trames 14 à 16 représentent l'établissement d'une connexion socket du client vers le serveur d'authentification.

Après la connexion socket, il devrait normalement se produire une connexion du client vers le serveur pour initier un *handshake* TLS. Le test vise justement le retardement de cette étape, afin que le nettoyeur interrompe le processus d'authentification.

On remarque donc l'effet du nettoyeur, à la trame 18, lors de la fermeture de la connexion.

Au niveau du client MédicIP, on remarque le message d'erreur suivant :

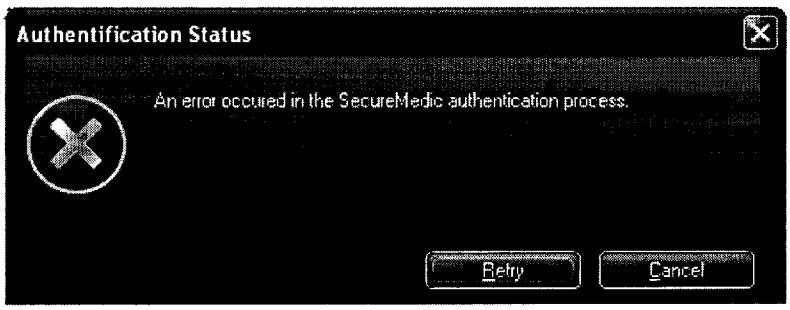


Figure B.11 Capture d'écran mentionnant à l'usager qu'il s'est produit une erreur durant le processus d'authentification.

Du côté du serveur d'authentification, on peut observer le résultat dans la base de données :

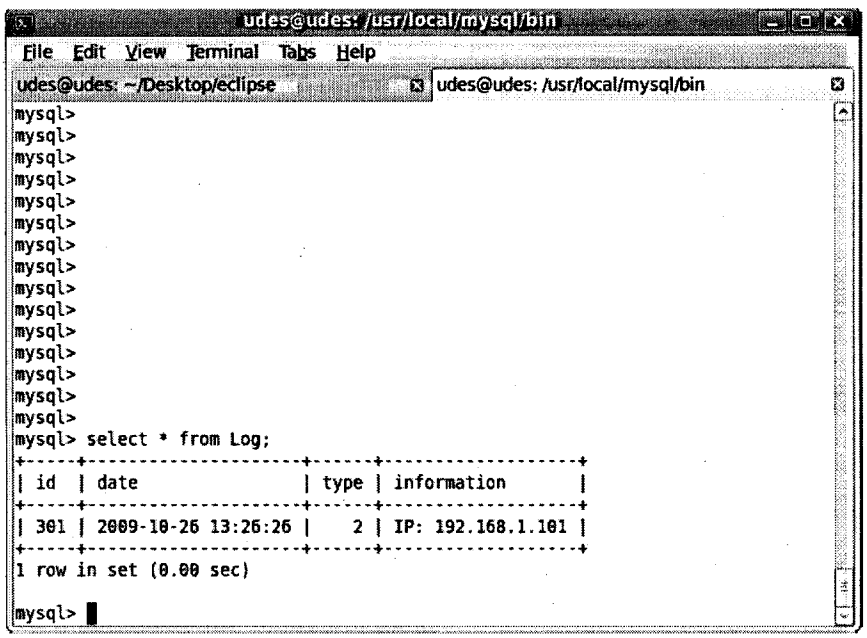


Figure B.12 Entrée dans la base de données indiquant un événement de type *thread killed by Nettoyeur.*

B.5 Authentification distante échouée - arrêt par le compteur individuel

B.5.1 Configuration de la librairie OpenSSL

Les configurations de la librairie initiale sont valides et identiques à celles du premier test (annexe B.1.1).

Le client possède par contre une modification temporaire pour permettre la réalisation de ce test. Dans le même ordre d'idées que pour le test précédent, un délai (*sleep*) a

été introduit dans le Client SécureMédic afin de simuler une surcharge du client ou du serveur, ou une panne réseau. Ce délai a été introduit avant l'envoi du mot de passe et du nom d'utilisateur de l'usager, et après l'établissement d'une connexion sécurisée suite au *handshake* TLS.

B.5.2 Objectifs du test

L'objectif de ce test est de valider le comportement du fil d'exécution **individualTimerThread**, dont le fonctionnement et le rôle sont décrits au chapitre 4.2.5.

B.5.3 Résultats

Voici la capture de trafic réseau résultant de ce test :

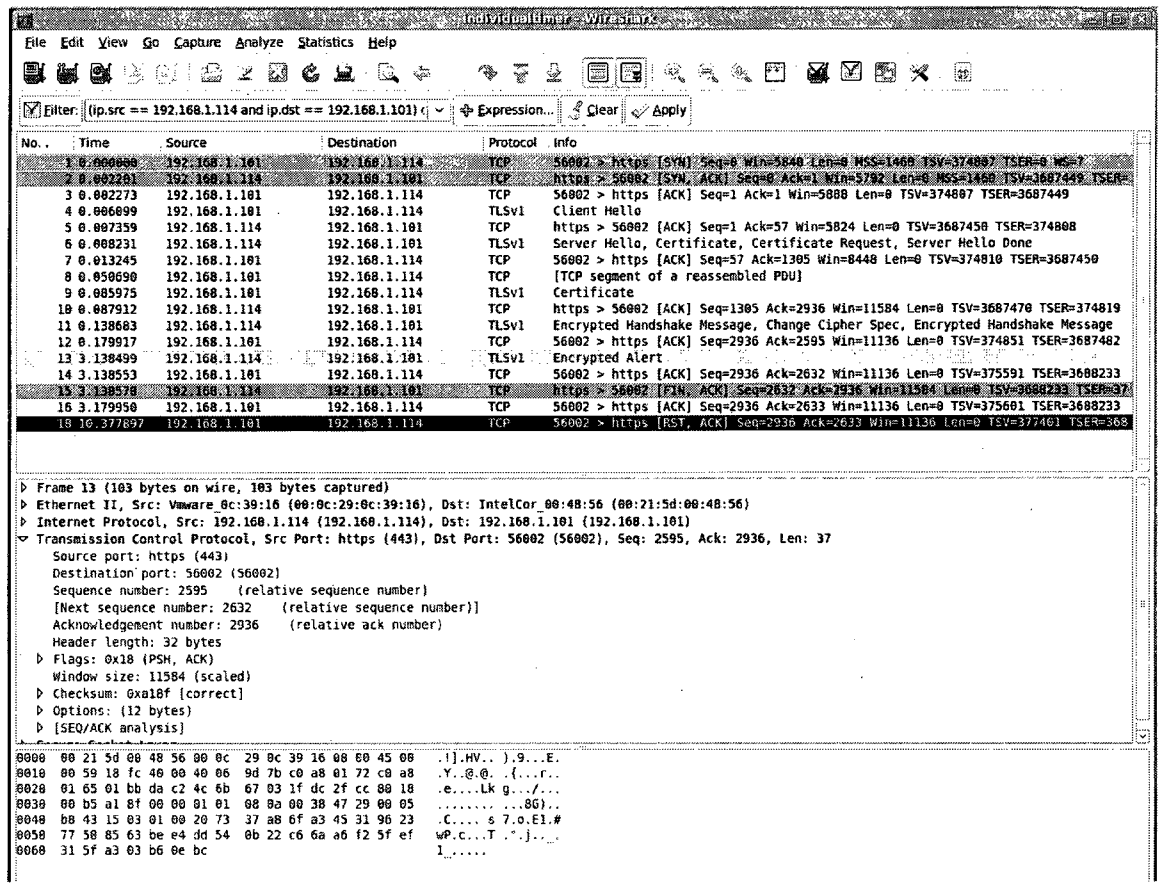


Figure B.13 Capture de trafic réseau réalisée avec Wireshark lors d'une authentification interrompue par le compteur individuel.

Les trames 1 à 3 représentent l'établissement d'une connexion socket du client vers le serveur d'authentification.

Les trames 4 à 12 représentent le *handshake* TLS permettant l'authentification mutuelle et l'établissement d'une communication sécurisée.

Après cette étape, l'envoi du mot de passe et du nom d'usager du client vers le serveur devrait normalement avoir lieu. Le test vise justement le retardement de cette étape, afin que le fil d'exécution individualTimerThread interrompe le processus d'authentification.

On remarque donc l'effet de ce fil d'exécution à la trame 13, lors de l'envoi de l'alerte chiffrée de fermeture de connexion, qui se produit par la suite dans les trames 14 à 18.

Au niveau du client MédicIP, on remarque le message d'erreur suivant :

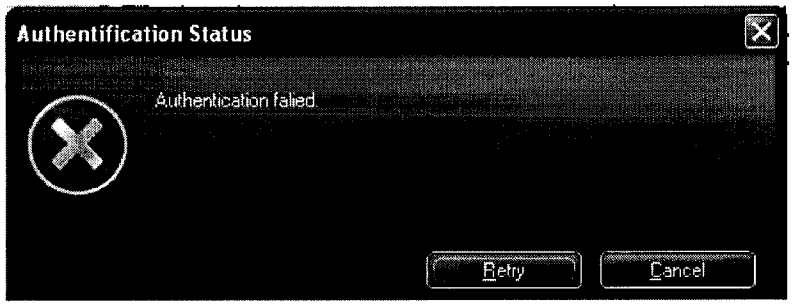


Figure B.14 Capture d'écran mentionnant à l'usager que l'authentification a échoué.

Du côté du serveur d'authentification, on peut observer le résultat dans la base de données :

```
mysql> select * from Log;
+-----+---------------------+------+------------------+
| id  | date                | type | information      |
+-----+---------------------+------+------------------+
| 298 | 2009-10-26 12:34:01 |    3 | IP: 192.168.1.101 |
+-----+---------------------+------+------------------+
1 row in set (0.00 sec)

mysql>
```

Figure B.15 Entrée dans la base de données indiquant un événement de type *individual thread times out.*

ANNEXE C

Résultats - Processus d'authentification local

Les chapitres 4.3.3 et 5.1.2 présentaient la description, le fonctionnement et les cas d'utilisation de l'authentification locale, gérés par le module *HistoryManager*.

Cette annexe présentera divers tests permettant de valider le comportement de ce processus d'authentification :

- authentification réussie ;
- authentification échouée (mauvaise clé privée) ;
- authentification échouée (historique local altéré) ;

C.1 Authentification locale réussie

Afin de valider le fonctionnement du processus d'authentification locale, deux authentifications à distance ont été effectuées à l'aide de deux usagers différents. Cette procédure permet la création de l'historique local. Voici son contenu après ces deux authentifications distantes réussies :

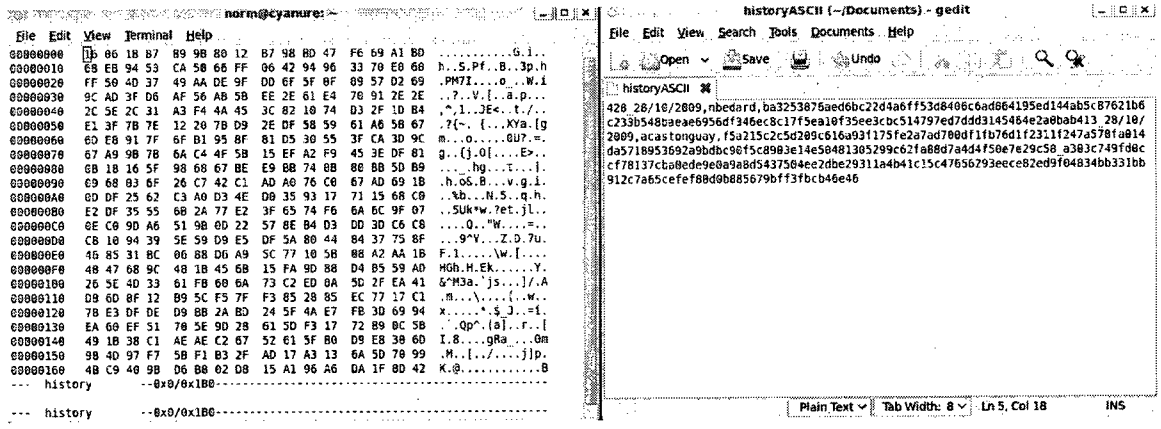


Figure C.1 Gauche : contenu de l'historique chiffré. Droite : contenu de l'historique déchiffré.

On remarque, dans la figure précédente, que le contenu de l'historique local est normalement chiffré. Un outil de test a été développé afin de déchiffrer son contenu, qui est affiché dans la portion de droite de la figure précédente.

Pour la suite du test, le serveur d'authentification a été fermé, de manière à simuler une panne. Ensuite, une authentification a été effectuée à l'aide du nom d'usager et du mot de passe de l'un des usagers précédemment authentifiés, dont les informations ont été ajoutées à l'historique local. Le module *HistoryManager* a été en mesure d'authentifier localement

l'usager, permettant ainsi à l'application MédicIP de démarrer sans la présence du serveur d'authentification.

Voici les événements enregistrés dans le fichier de *logs* de SécureMédic :

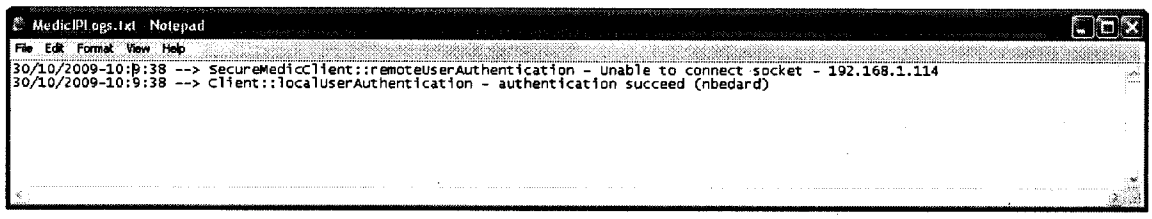


Figure C.2 Fichier de *logs* du Client SécureMédic lors d'une authentification locale réussie.

On remarque le premier événement mentionnant que la connexion avec le serveur d'authentification a échoué. Le deuxième événement mentionne l'authentification locale réussie, avec le nom d'utilisateur en question.

C.2 Authentification locale échouée - mauvaise clé privée

Le test qui suit est celui où la clé utilisée par le *HistoryManager* (chapitre 4.3.3) pour effectuer l'authentification locale est la mauvaise.

Voici le résultat affiché à l'usager :

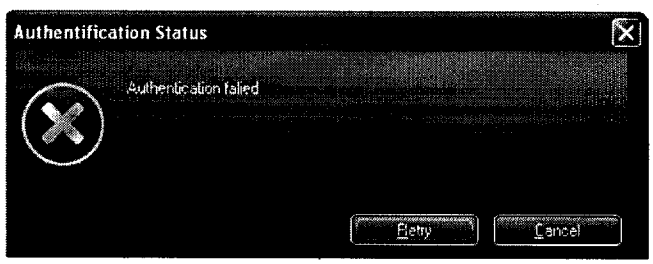


Figure C.3 Message d'erreur indiquant à l'usager que l'authentification a échoué.

Davantage de détails sont inscrits dans le fichier de *logs* de SécureMédic :

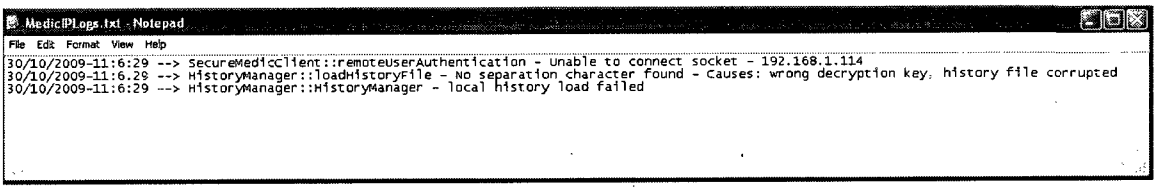


Figure C.4 Événements enregistrés dans le fichier de *logs* lors d'une authentification locale échouée due à l'utilisation d'une mauvaise clé privée.

Le premier événement indique une fois de plus l'incapacité à rejoindre le serveur d'authentification. Le deuxième événement nous renseigne sur le problème engendré par l'utilisation d'une mauvaise clé de déchiffrement. Finalement, le troisième événement découle du deuxième, où le déchiffrement de l'historique a provoqué une erreur lors de la lecture de l'historique local.

C.3 Authentification locale échouée - historique local altéré

Ce test se veut un moyen d'observer le comportement du *HistoryManager* lorsque l'historique local a été altéré, soit accidentellement, soit de façon délibérée. Pour ce faire, l'historique local a été ouvert dans un éditeur hexadécimal et son contenu a été altéré de manière aléatoire.

Lors de l'authentification usager, le même message d'erreur qu'au test précédent a été affiché à l'utilisateur.

Voici les résultats se trouvant dans le fichier de *logs* pour ce type d'événement :

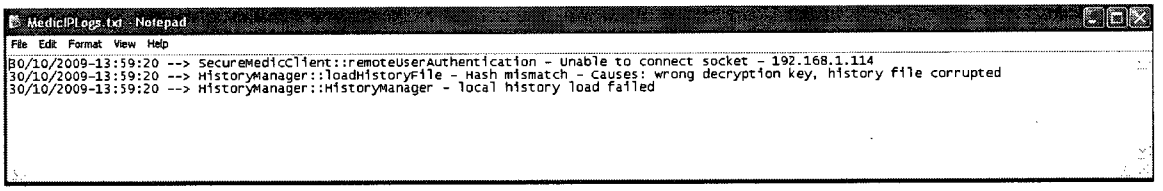


Figure C.5 Événements enregistrés dans le fichier de *logs* lors d'une authentification locale échouée, due à une altération de l'historique local.

On remarque, au deuxième événement, qu'il y a eu détection de l'altération de l'historique local, grâce au format des données utilisées. En effet, la section HASH ne concorde pas avec le hachage du fichier altéré (voir chapitre 4.3.3).

ANNEXE D

Résultats - Conférences multimédias

Cette annexe présente les résultats du processus de sécurisation des conférences multimédias du prototype MédicIP :

- établissements de conférences ;
- échanges vocaux ;
- échanges de données (électrocardiogrammes) ;

Plusieurs tests permettent de valider cette sécurisation :

- création d'une conférence multimédia réussie ;
- création d'une conférence multimédia échouée (mauvais certificat root) ;
- création d'une conférence multimédia échouée (mauvais certificat client) ;

Afin de mettre en perspective les conférences sécurisées et les conférences non sécurisées, des comparaisons sur les points suivants seront réalisées :

- temps requis et bande passante requise pour l'établissement d'une conférence ;
- temps requis et bande passante requise lors des échanges de données pendant une conférence.

D.1 Création d'une conférence multimédia réussie

D.1.1 Configuration des protocoles

La première étape lors de la création d'une conférence multimédia est la signalisation de celle-ci, à l'aide du protocole SIP. Ce protocole a été utilisé conjointement avec le protocole TLS, afin de sécuriser cette signalisation.

Comme mentionné dans le chapitre 5.2.1, l'utilisation du protocole SIP avec TLS a nécessité le recours à une librairie offerte par l'entreprise M5T. Afin de configurer le protocole TLS, les étapes suivantes ont été effectuées :

1. création d'une chaîne de certificats contenant le certificat de l'usager ainsi que le certificat root (*trusted certificate*) ;
2. création d'un contexte TLS ;
3. assignation de la chaîne de certificats au contexte TLS ;
4. assignation des jeux de chiffrement supportés au contexte TLS ;
5. assignation du *trusted certificate* au contexte TLS (dans ce cas-ci, il s'agit du certificat root) ;

6. assignation des adresses IP autorisées pour valider le *common name*, contenu dans les certificats présentés par les entités faisant partie de la conférence.

Le prototype MédicIP a été conçu de manière à utiliser le même contexte TLS que celui utilisé lors de la signalisation SIP, lors de l'envoi des électrocardiogrammes. La configuration précédente pour utiliser SIP avec TLS est donc suffisante pour sécuriser la signalisation SIP, ainsi que l'envoi des électrocardiogrammes.

La sécurisation des communications vocales repose sur l'utilisation du protocole SRTP. Dans le cadre de ce test, une clé maître (*master key*) intégrée à même le code logiciel a été utilisée afin de permettre la dérivation de clés, et ainsi permettre le chiffrement des données.

D.1.2 Objectifs du test

L'objectif de ce test est de valider la sécurisation du protocole SIP, soit l'utilisation du protocole "SIP over TLS". La configuration précédente a été utilisée afin de réaliser ce test.

D.1.3 Résultats

Premièrement, voici une capture de trafic réseau réalisée lors de l'établissement d'une conférence multimédia avec l'application MédicIP originale :

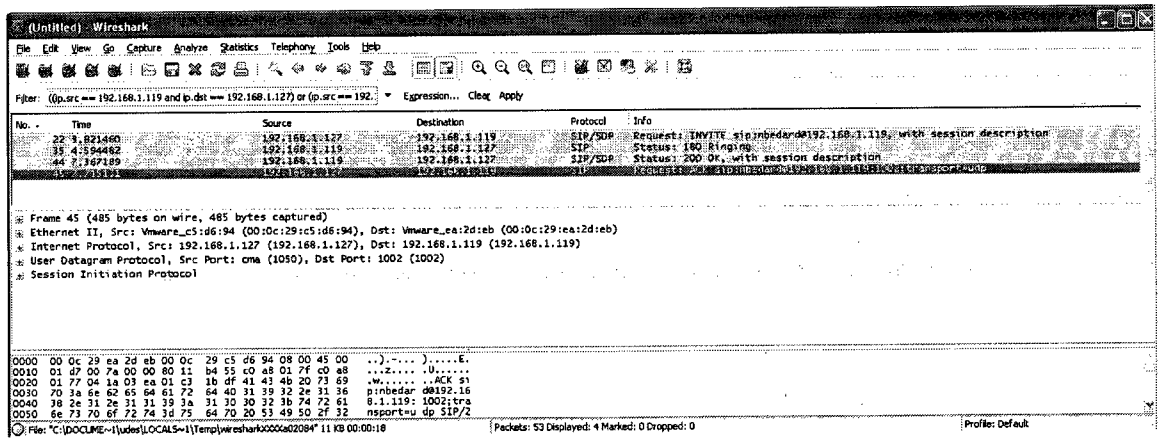


Figure D.1 Capture de trafic réseau lors de l'établissement d'une conférence multimédia avec l'application MédicIP originale.

Voici maintenant une capture de trafic réseau réalisée lors de l'établissement d'une conférence multimédia avec l'application MédicIP sécurisée par le projet SécureMédic :

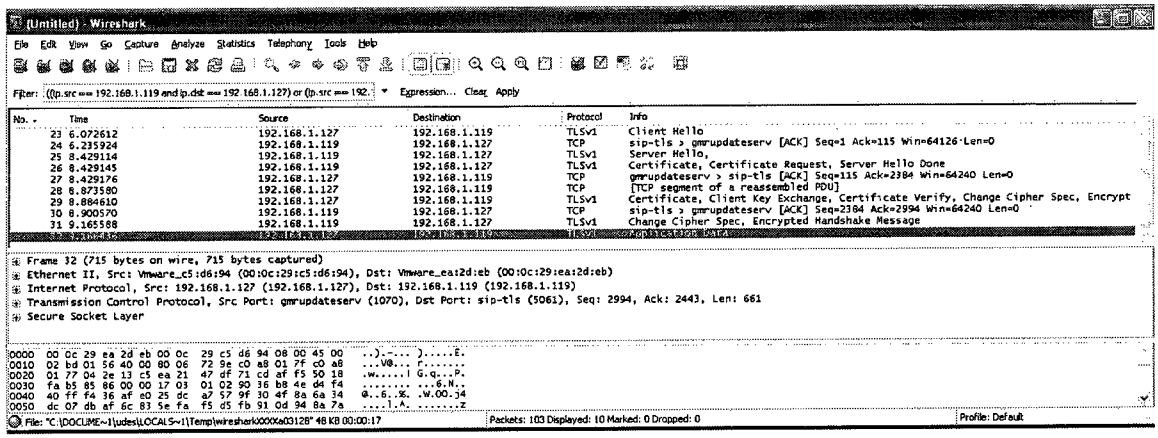


Figure D.2 Capture de trafic réseau lors de l'établissement d'une conférence multimédia avec l'application MédicIP sécurisée.

Lors de l'établissement d'une conférence avec l'application MédicIP, on remarque la signalisation de la conférence à l'aide du protocole SIP. Pour ce qui est de SécureMédic, on remarque la présence d'un *handshake* TLS, permettant de sécuriser les échanges via des messages de type *Application Data*.

D.2 Création d'une conférence multimédia échouée - mauvais certificat root

Pour ce test, la configuration des protocoles est la même qu'à la section précédente (section D.1). Par contre, le *root certificate* utilisé est un certificat ne faisant pas partie de l'infrastructure à clés publiques.

Puisqu'en temps normal l'application MédicIP ne démarre pas lors de l'utilisation d'un tel certificat (voir chapitre B.3), une modification temporaire a été apportée à l'application MédicIP afin que l'authentification usager ne soit pas effectuée. Ainsi, le module gérant les conférences multimédias peut être testé de manière isolée.

D.2.1 Objectifs du test

L'objectif est de valider le comportement du module gérant les conférences multimédias lorsqu'un certificat *root* non autorisé est utilisé. On s'attend à ce qu'il se produise une erreur et que la conférence ne puisse débuter.

En fait, deux hypothèses peuvent être avancées à ce stade. Premièrement, il est possible que la création de la chaîne de certificats (voir annexe D.1.1) produise une erreur, étant donné que le certificat usager n'est pas signé par le faux certificat root. Deuxièmement, si une erreur n'est pas générée durant la création de la chaîne de certficats, il est fort probable que le *handshake* TLS échoue, dû à l'invalidité du certificat root. Dans les deux cas, la conférence ne devrait tout simplement pas démarrer.

D.2.2 Résultats

Voici maintenant la capture de trafic réseau reliée à ce test :

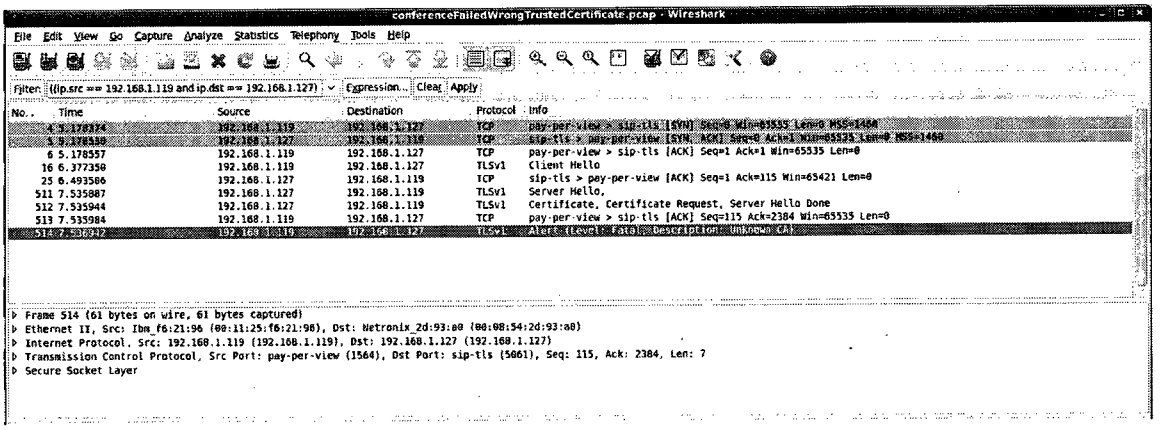


Figure D.3 Capture de trafic réseau lors de l'échec de l'établissement d'une conférence multimédia, dû à la présence du *trusted certificate* non autorisé, créant ainsi une erreur lors du *handshake* TLS.

Comme lors du test précédent, on observe les échanges reliés au *handshake* TLS. La trame 514 affiche une alerte fatale, indiquant l'échec du *handshake* TLS et par le fait même, l'échec de la signalisation SIP.

D.3 Création d'une conférence multimédia échouée - mauvais certificat client

Pour ce test, la configuration des protocoles est la même que dans les deux sections précédentes (sections D.1 et D.2). Cependant, dans ce test-ci, le certificat client qui est utilisé n'est pas signé par le certificat root.

Comme lors du test précédent, une modification temporaire du client a été faite afin qu'il n'y ait pas de validation de ce certificat, encore une fois, pour s'assurer de tester le module gérant les conférences multimédias de manière indépendante.

D.3.1 Objectifs du test

L'objectif de ce test est de valider le comportement du module gérant les conférences multimédias lorsque le certificat client utilisé n'est pas signé par le certificat root. On s'attend à ce qu'il se produise une erreur et que la conférence ne puisse débuter.

Deux hypothèses peuvent être avancées avant d'effectuer ce test. Premièrement, il est possible que la création de la chaîne de certificats (voir annexe D.1.1) produise une erreur, étant donné que le faux certificat usager n'est pas signé par le certificat root. Deuxièmement, si une erreur n'est pas générée durant la création de la chaîne de certificats, il est fort

probable que le *handshake* TLS échoue, puisque la validation du certificat client échouera. Dans les deux cas, la conférence ne devrait tout simplement pas démarrer.

D.3.2 Résultats

Voici maintenant la capture de trafic réseau reliée à ce test :

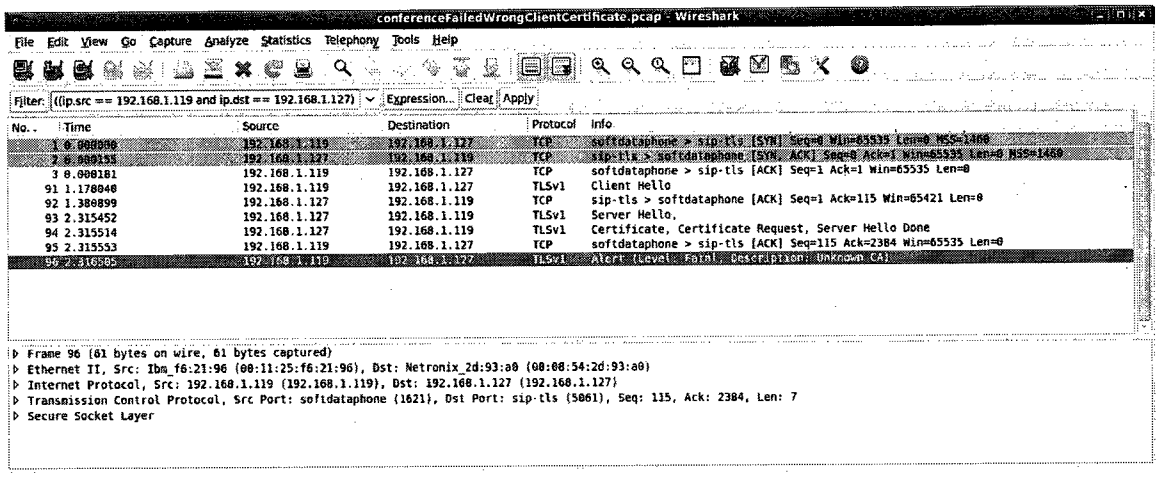


Figure D.4 Capture de trafic réseau lors de l'échec de l'établissement d'une conférence multimédia dû à la présence d'un certificat client non autorisé. La trame 96 affiche une alerte fatale, indiquant l'échec du *handshake* TLS et, par le fait même, l'échec de la signalisation SIP.

D.4 Conférence sécurisée VS conférence non sécurisée

Cette section permettra de mettre en perspective l'ancien module de communications multimédias de MédicIP avec celui produit par le projet SécureMédic. Les résultats et statistiques suivants ont été produits à l'aide de deux ordinateurs branchés sur un réseau local (LAN).

D.4.1 Temps requis pour l'établissement d'une conférence

Afin de quantifier le temps requis pour l'établissement d'une conférence, certaines bases doivent être communex et servir de référence pour la réalisation des tests. Dans ce cas-ci, l'objectif était de comparer le temps requis pour établir une conférence pour l'application MédicIP originale ainsi que pour l'application MédicIP sécurisée par le projet SécureMédic.

Pour chacune des plateformes, quinze établissements de conférence ont été effectués. Les temps calculés ont été mesurés à l'aide de l'outil *wireshark*. Pour ce faire, il a été établi comme référence que le temps requis pour établir une conférence est le temps séparant le premier paquet réseau envoyé du premier paquet audio envoyé.

Voici les résultats pour ces tests :

Conférence	MédicIP (sec)	SécureMédic (sec)
1	3.96	6.63
2	3.96	7.72
3	5.15	7.70
4	4.11	7.11
5	3.71	6.46
6	4.93	7.52
7	3.59	8.32
8	5.08	7.32
9	4.43	8.90
10	4.26	7.30
11	5.01	7.30
12	3.47	6.84
13	5.92	8.54
14	4.83	7.79
15	4.62	7.31
Moyenne	4.47	7.52
Médiane	4.43	7.32
Écart-type	0.69	0.68

Figure D.5 Temps requis pour la création d'une conférence.

D.4.2 Charge réseau utilisée pour l'établissement d'une conférence

Afin de quantifier la charge réseau utilisée pour l'établissement d'une conférence, certaines bases doivent être communes et servir de référence pour la réalisation des tests. Dans ce cas-ci, l'objectif était de comparer la bande passante réseau utilisée pour établir une conférence pour l'application MédicIP originale ainsi que pour l'application MédicIP sécurisée par le projet SécureMédic.

Pour chacune des plateformes, quinze établissements de conférence ont été effectués. Les charges réseau calculées ont été mesurées à l'aide de l'outil *wireshark*. Pour ce faire, il a été établi comme référence que la charge réseau requise pour établir une conférence est la taille totale des paquets échangés entre les clients comprise entre le premier paquet et le paquet précédent le premier paquet audio.

Voici les résultats pour ces tests :

Conférence	MédicIP (octets)	SécureMédic (octets)
1	8367	14816
2	7446	14756
3	8148	14936
4	7274	15160
5	7569	14271
6	8388	15116
7	8632	13948
8	7689	14062
9	7625	15340
10	8752	14762
11	7325	14800
12	7754	13986
13	8330	14816
14	8557	14996
15	9122	13948
Moyenne	8065.2	14647.53
Médiane	8148	14800
Écart-type	576.91	476

Figure D.6 Bande passante requise pour la création d'une conférence.

D.4.3 Échanges d'ecgML

Afin de comparer les performances lors du transfert d'électrocardiogrammes, certaines bases doivent être communes et servir de référence pour la réalisation des tests. Dans ce cas-ci, l'objectif était de comparer les temps requis pour le transfert des électrocardiogrammes ainsi que la bande passante requise durant ces transferts.

Pour chacune des plateformes, six tests ont été effectués. Chacun des tests se déroulait de la même façon, soit le transfert de cinq électrocardiogrammes (fichiers ecgML). Dans les résultats qui suivent, la durée de transfert de chacun des ecgML ainsi que le temps cumulatif total des ecgML 1 à 5 ont été calculés individuellement.

De plus, le temps total de la conférence pour chacun des tests a été calculé. Ces valeurs représentent le temps entre le moment où on lance le transfert d'ecgML, et celui où le transfert du cinquième et dernier ecgML est complété.

Finalement, la bande passante requise pour chacun des tests a été calculée.

Voici maintenant l'ensemble des résultats :

SecureMédic

	Durée des fichiers ecgML (sec)					
	#1	#2	#3	#4	#5	#1 à #5
Test 1	0.87	0.66	1.10	0.92	1.40	4.95
Test 2	1.59	0.69	0.57	0.46	1.64	4.95
Test 3	0.66	0.64	0.69	0.54	0.82	3.35
Test 4	0.69	0.64	1.67	0.63	1.81	5.44
Test 5	1.10	0.67	0.74	0.55	0.52	3.58
Test 6	0.69	0.82	0.68	0.85	0.80	3.84
Moyenne	0.93	0.69	0.91	0.66	1.17	4.35
Médiane	0.78	0.67	0.72	0.59	1.11	4.40
Écart-type	0.36	0.07	0.41	0.16	0.52	0.87

Durée totale de la conférence (sec)	Taille totale de la conférence (octets)
52.40	194950
55.35	194571
55.65	195116
52.55	193583
52.22	195099
56.30	195170
54.08	194748
53.95	195025
1.66	611

MédicIP

	Durée des fichiers ecgML (sec)					
	#1	#2	#3	#4	#5	#1 à #5
Test 1	0.10	0.03	0.04	0.03	0.04	0.24
Test 2	0.05	0.03	0.04	0.04	0.03	0.19
Test 3	0.39	0.03	0.73	0.05	0.69	1.89
Test 4	0.05	0.22	0.04	0.18	0.05	0.54
Test 5	0.03	0.04	0.04	0.04	0.04	0.19
Test 6	0.04	0.03	0.03	0.03	0.05	0.18
Moyenne	0.11	0.06	0.15	0.06	0.15	0.54
Médiane	0.05	0.03	0.04	0.04	0.05	0.22
Écart-type	0.14	0.08	0.26	0.06	0.26	0.68

Durée totale de la conférence (sec)	Taille totale de la conférence (octets)
49.99	167998
51.60	166703
55.39	172428
50.87	173589
55.36	178060
51.65	172468
52.48	171874
51.63	172448
2.32	4090

Figure D.7 Durée et bande passante requises pour le transfert de cinq ecgML. La portion gauche des tableaux détaille le temps requis pour l'envoi de chacun des ecgML ainsi que le temps cumulatif total requis pour l'envoi des cinq ecgML. La portion de droite indique la durée totale des conférences ainsi que la taille totale des données échangées pour chacun des tests. On remarque que SecureMédic prend un temps cumulatif beaucoup plus long pour l'envoi des cinq ecgML, mais produit une durée totale et une taille totale de conférence similaires.

D.4.4 Échanges audios

Afin de comparer les performances lors de communications vocales, certaines bases doivent être communes et servir de référence pour la réalisation des tests. Dans ce cas-ci, l'objectif était de comparer les temps requis pour le transfert de données audios ainsi que la bande passante requise durant ces transferts.

Pour chacune des plateformes, cinq tests ont été effectués. Chacun des tests se déroulait de la même façon, soit le transfert des données d'une séquence audio enregistrée à même le code logiciel de SécureMédic. L'utilisation d'un microphone pour effectuer ces tests n'aurait pas permis d'effectuer de comparatifs significatifs. Dans les résultats qui suivent, le temps total de la transmission audio ainsi que la bande passante requise ont été calculés.

Voici maintenant l'ensemble des résultats :

	SécureMédic		MédicIP	
Conférence	Temps requis (sec)	Taille (octets)	Temps requis (sec)	Taille (octets)
1	3.23	30310	3.04	30152
2	3.14	30370	3.21	30046
3	3.50	30430	3.06	30152
4	3.13	30370	3.11	30152
5	3.26	30370	3.12	30152
Moyenne	3.25	30370	3.11	30131
Médiane	3.23	30370	3.11	30152
Écart-type	0.15	42	0.07	47

Figure D.8 Durée et bande passante requises pour le transfert d'une séquence audio préenregistrée. Les cinq tests comparatifs permettent de confirmer que le processus de sécurisation génère des pertes de performance peu significatives au niveau du temps requis et de la bande passante requise pour l'envoi d'une séquence audio.

ANNEXE E
Conférence ISMICT 2009

Lors du *Third International Symposium on Medical Information & Communication Techology*, le projet SécureMédic fut présenté. Voici l'article de conférence présenté :

Description of the MedicIP Communication Platform with Emphasis on Data Security Issues

Normand Bédard[a], Alain Houle[a], Guy Lépine[b], Frédéric Mailhot[a]

[a] *Université de Sherbrooke ,Sherbrooke (Qc), Canada*
[b] *M5T, Sherbrooke (Qc), Canada*

Abstract

MedicIP, a system for communicating by voice and exchanging data between medical personnel during an emergency situation has been designed and implemented. The security aspects of the system are reviewed, including authentication of participants, data integrity and access control, robust resistance to external attacks. The performance of the various protection elements is evaluated.

Keywords: *Patient Care Team, Computer Security, Secure Multimedia Conference*

Introduction

Effective communication between ambulance crew and medical personnel at a hospital emergency department is known to have a very positive impact on both patient care quality and efficiency of the emergency operations [1]. As an example, making it possible for the ambulance crew to share real-time electrocardiograms (ECGs) through multimedia conferencing with an emergency physician and a cardiologist can surely help in decision making for patient transportation and can also contribute to make sure a medical team is properly prepared to receive the patient [2-3]. We present the MedicIP communication platform, which is actually being developed at Université de Sherbrooke. In particular, our interest lies in the security aspects of such a distributed communication platform. First, we describe the architecture of the MedicIP platform itself. We then present solutions for security issues that have been identified. The MedicIP vision: multimedia communication over IP anytime, anywhere.

The MedicIP project began in 2005 with the idea of building a communication system able to support simultaneous voice communication and real-time ECG transmission from an ambulance to a hospital. The MedicIP platform allows more than point-to-point communication: it also enables multimedia conferencing for up to ten parties [4]. In a typical case scenario, an ambulance crew establishes a communication link with an emergency physician at a first-line hospital. Depending on the analysis of the physiological signals received from the ambulance, the emergency physician has multiple options: 1) he can decide to prepare his team to receive the patient; 2) he can invite another physician, potentially a specialist from another institution, into the conference to get his opinion; 3) he can decide to reroute the ambulance to a more appropriate institution given this institution is able to receive the patient.

The MedicIP platform is an IP (Internet Protocol) -based communication platform. Since the beginning of the project, we did not consider the physical radio links required for ambulance communication and assumed that any involved party has access to an IP network. Future work will concentrate on the radio communication aspects as many new technologies are now emerging (e.g. APCO P25 standard [5]).

The MedicIP signaling functions, required to establish communication sessions between parties, are implemented using the SIP protocol (RFC 3261) [6]. A full-mesh conferencing protocol was implemented over the SIP middleware [7]. The full-mesh conferencing scheme is not the most bandwidth efficient one as $N(N-1)/2$ data streams are required in opposition to only N data streams for the more conventional client-server conferencing scheme. However, as it is not very likely that more than ten parties would be involved in the kind of emergency situation we are aiming, bandwidth efficiency is not our first concern. The important issue is the reliability of the system because of the critical nature of the MedicIP task. The full-mesh conferencing scheme is then an excellent candidate as it does not have a single point of failure. Being a distributed scheme, any party can leave an ongoing conference without leaving other parties in the dark. This is not the case with the conventional client-server scheme where the central server must be always reachable and available.

With respect to the type of information that can be communicated using the MedicIP platform, we began by considering the simultaneous transmission of voice and real-time ECG. Voice signals are encoded according to conventional voice-over-IP (VoIP) standards [8]. We selected ecgML [9] as a representation format for ECGs. Based on the XML technology, ecgML is intrinsically compatible with SIP technology. Data streams are transported using the RTP protocol (RFC

3550) [6]. Since we combine different types of information for simultaneous transmission, we are exploiting multimedia communication over IP (MMoIP).

In January 2007, a team of six undergraduate computer engineering students received the mandate to redesign the MedicIP platform and to incorporate a sophisticated graphical user interface (GUI) (*Figure 1*). That work, made under the umbrella of a final-year design project in our computer engineering curriculum at Université de Sherbrooke, also included interaction of the MedicIP platform with a presence server used to locate physicians and specialists [10].

Since January 2008, the development of the MedicIP platform has been focused on security issues, under the project name SecureMedic. The remaining of this paper reports our work on these issues.

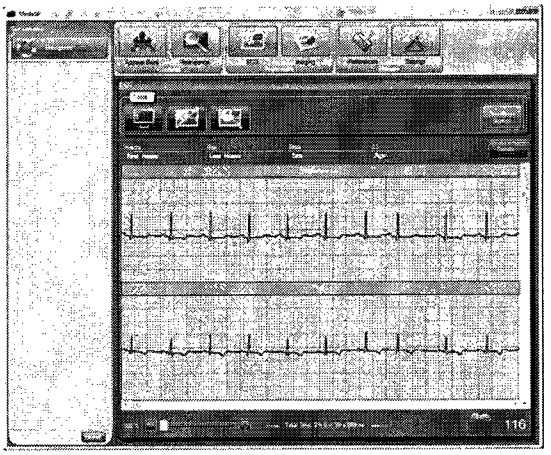


Figure 1- The MedicIP GUI

Methods

MedicIP is intended as a communication tool both between distant physicians and also between emergency personnel and ambulance crews on their way to the emergency departments. It is a Windows-based software client application that runs on traditional computers. The software system consists of approximately 45,000 lines of C# code, which has been extensively tested and validated by a team of physicians.

User authentication

The SecureMedic project completely redesigns the authentication process of the MedicIP prototype. The initial objectives and goals of the SecureMedic authentication architecture answer problems identified in the MedicIP prototype:

- Have a transparent mutual authentication between the server software and the client software. This avoids unauthorized malicious entities to form part of a communication.
- Have secure communications between entities during an authentication.
- Have a robust and reliable centralized authentication server.
- Have an efficient way for users to authenticate themselves.
- Have flexible authentication alternatives if there are network problems or if the authentication server is unavailable.
- Allow users to securely receive their credentials after a successful authentication.

The SecureMedic architecture is a PKI (Public Key Infrastructure). Its role is to provide services that allow the authentication of entity by another. Two major elements are present in the SecureMedic PKI:

- Private keys
- Public keys, embedded into X.509v3 digital certificates

The system is based on 2048 bits RSA keys and the SecureMedic architecture uses three kind of digital certificates:

- Generic server certificate (root). This certificate holds the server software credential. The private key associated to the generic server certificate's public key is hardcoded into the server software. Doing so lowers the burden of certificate management.
- Generic client certificate. This certificate, signed by the root certificate, holds the client software credentials. The private key associated with the generic client certificate's public key is hardcoded in the client software.
- User certificates. Each user credentials are held within a unique certificate, signed by the root certificate. Initially, the authentication server has all the client's certificates and associated private keys.

The next sequence diagram shows how the user authentication and the mutual authentication between client and authentication server works.

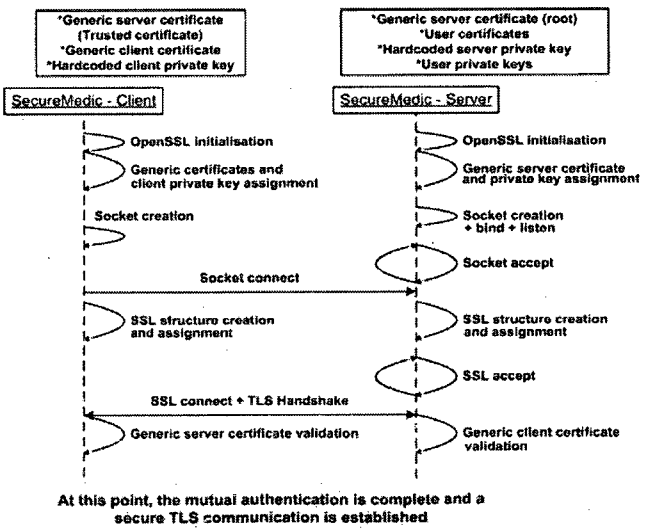


Figure 2 - Device mutual authentication

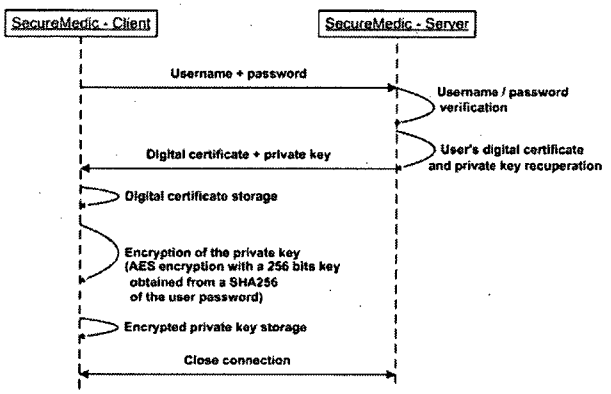


Figure 3 - User authentication

For performance reasons, the server software is multithreaded. Each socket connection creates a new thread that is responsible for user authentication and mutual device authentication. The authentication server also includes security protections against some types of Denial of Service (DoS) attacks. Another thread, called the "Cleaner", is responsible for cleaning up inactive socket connection threads, possibly caused by a DoS. This thread can be seen as a garbage collector for the connection list.

The authentication server contains a list which can be filled with a maximum number of threads. If the list is full, there is no socket connection listening. Otherwise, socket connections are possible and a thread can be created and added to the list.

The "Cleaner" thread has an internal timer which depends on the number of threads in the list; the value of the timeout is inversely proportional to the number of connections in the list. If there is a small number of connections in the list, there is no disadvantage to let a long timeout running. However, a large number of connections in the list indicates higher traffic on the network, causing congestion. This is why the "Cleaner" thread is advantageous to cleanup the list more rapidly. When the timer expires, the "Cleaner" kills and removes the oldest connection from the list and restarts its timer.

Connections stay in that list until they complete a TLS handshake (see Figure 2). The handshake allows the client and the authentication server to authenticate the peer and agree to a symmetric keys used for data [11]. Upon the successful completion of the TLS handshake (which includes entity authentication), the connection is removed from the list and the user authentication takes place. Every thread is associated with a timer which ensures none of the threads will consume computer resources forever if the authentication is not complete. The "Cleaner" thread does not manage threads that are removed from the list.

This method protects the system against malicious socket connections which could try to overload the authentication server. Indeed, if the list is full, no socket will be made created, and no thread will be created, meaning that no computer resources are used by those connection attempts. Although malicious connections are possible when the list is not empty, they will be killed by the "Cleaner" thread if they are inactive or will be killed by the authentication server if they fail the TLS handshake. Since there is no complete solution against a DoS [12] attack, this method is not immune to the authentication server refusing legitimate connections. It's a trade-off between security, robustness and reliability. Our method assures that a legitimate authentication request present in the list will be processed within a predefined time period.

To summarize, the list is used as a buffer protection, which allows the authentication server to filter the legitimate authentication requests. It is also called a perimeter defense.

Privacy of information exchanges

A MMoIP communication exchange is done over a TCP/IP network. This type of network is intrinsically insecure. Therefore, additional protocols need to be used in order to provide communication security over this insecure network.

First, the SIP signalling must be protected. The MedicIP software client uses SIP, which in turn supports TLS (RFC5246) [6], the ubiquitous protocol for securing communications over TCP (RFC793) [6]. This protocol provides:

- Server authentication (the entity which will receive and process the SIP requests)
- Optional client authentication. In the process of securing the MedicIP application, client authentication using X.509v3 certificates has been chosen
- Data privacy, using a negotiated symmetric-key algorithm
- Data integrity, using an also negotiated HMAC algorithm

Note, however, that SIP over TLS only secures the conference signalling between the parties. The actual multimedia data exchanged need to be protected by other means. This is where SRTP/SRTCP comes into play. When initiating a multimedia session with SIP, the session description (i.e. the devices' multimedia capabilities) is specified using SDP (RFC4566) [6] and negotiated as per RFC3264 [6]. The negotiation results in a chosen codec, ptime, IP address and port for the exchange to take place, usually using RTP.

However, RTP is insecure and additional protocols need to be used to protect the multimedia data. Therefore, a secure version of RTP, SRTP (RFC3711) [6], is put in place, providing a symmetric-key algorithm to protect the

multimedia communication. The key needs to be exchanged by other means than SRTP. Usually, the key exchange will be done using MIKEY (RFC3830) [6], inserted in the SDP exchange. The SDP exchange being a payload of SIP packets, the latter being secured with TLS, there is no way an attacker could eavesdrop on the keying material.

Therefore, SIP over TLS is used to protect the conference signalling, in conjunction with SRTP/SRTCP to protect the multimedia exchanges.

Security of stored data

Secure data storage was not an initial objective of the MedicIP development team. Both the client and authentication server lack security measures to ensure data privacy and integrity. SecureMedic fixes a number of those problems on the authentication server side, in particular data previously stored in unprotected XML files.

First of all, the authentication server has been completely redesigned in C++, on OpenBSD (www.openbsd.org), considered by many experts as one of the most secure operating system.

A local database, whose access is authorized only with the required credentials, contains all the information needed by the system (user accounts, digital certificates and private keys filenames, user privileges, hashed passwords, etc.).

Users' digital certificate files and private keys indicate specific file access restrictions, which are managed by the operating system. In addition to the data protection, all the hard drive is encrypted with the TrueCrypt (www.truecrypt.org) which allows on-the-fly encryption and decryption of needed files by the authentication server.

On the client side, MedicIP was storing two types of data: address book and ecgML files. No file restrictions protected that data, therefore anybody who had access to the computer could read, modify or delete it. With SecureMedic, all that confidential information is the AES symmetric key cryptography algorithm (AES is a standard adopted by the U.S. Government [13]). The AES key is generated on demand using the user's private key with a Password-Based Key Derivation Function (PBKDF2) that is part of RSA Laboratories' Public-Key Cryptography Standards (PKCS) series, specifically PKCS #5 V2.0 [14].

Discussion

SecureMedic uses existing security concepts or products with the aim of fixing security vulnerabilities of an existing prototype.

This means that the project has limits introduced by the technological choices made. In the case of the authentication scheme using the PKI's concept, there are some restrictions. Indeed, the architecture developed does not offer all the functionalities of a complete Internet X.509 public key infrastructure certificate management protocol (RFC 2510) [6]. The development of a complete PKI architecture could itself be a project, and will possibly be added to SecureMedic in the future.

As discussed before, we introduced a protection against a flood DoS attack that could overload the authentication server with authentication requests. Many other types of DoS attacks exist, like infrastructure DoS, service DoS, network DoS, application DoS, etc. SecureMedic does not claim to be able to counter advanced and sophisticated DoS attacks.

One of our objectives was to have flexible authentication alternatives if there are network problems or if the authentication server is unavailable. Two solutions are currently being designed:

- Relayed authentication
- Local authentication

For the relayed authentication, it is possible that a client is unable to reach the authentication server, but is able to reach another client that has been already authenticated. With the use of the digital certificate that signed the client devices (two client devices can do mutual authentication), a protocol will be designed to allow a client to use another authenticated client as a relay to "talk" to the authentication server and authenticate himself.
As a last resort, a client device will be able to authenticate a user locally with a local history of successful authentications. This history will be encrypted with the client generic private key and hash functions will ensure that no modifications have been made on the history. Because the user's digital certificate and private key are stored on a client device (see Figure 3), the client software will be able to use it for other uses (discussed in the above privacy of information exchanges section). Since the prototype is used in a highly critical situation involving patients' lives, it is important to have this trade-off between security and accessibility, resulting in a functional system even if the central authentication server is unavailable. A 99.99% server uptime rate involves a 52 minutes downtime period over a year, which indicates the necessity of that trade-off.

Conclusion

The MedicIP prototype fills a need in the traditional way patients are transported to a hospital, by allowing ambulance crew and emergency personnel to efficiently interact earlier in the process.

Because this product uses IP network and Internet as a communication medium, the security aspects of the project needed to be addressed.

SecureMedic focuses on some important security aspects: secure authentication process, secure information exchanges and secure data storage. Since computer systems are never completely secure, future development is planned on SecureMedic to minimize the risks associated with a networked environment.

Acknowledgements

We thank Université de Sherbrooke's Department of Electrical Engineering and Computer Engineering, which sponsored the team of six undergraduate students who worked on the latest version of the MedicIP platform. The members of that team, Benoit Beauchemin, Francis Beaulé, Marc Boissonneault, Jean-François Desjean-Gauthier, Guillaume Dubeau and Francis Ruel, were essential to the design and realization of the latest prototype and have all our gratitude.

For information, please contact

Alain Houle: alain.houle@usherbrooke.ca

Normand Bédard: normand.bedard@usherbrooke.ca

Frédéric Mailhot : frederic.mailhot@usherbrooke.ca

Guy Lépine: glepine@m5t.com

References

[1] Rowlands, A., "An evaluation of pre-hospital communication between ambulances and an accident and emergency department", Journal of Telemedicine and Telecare, (2003), 35-37.

[2] McNamara et al., "Effect of door-to-balloon time on mortality in patients with ST-segment elevation myocardial infarction", J.Am.Coll.Cardiol., 2006, vol.47, pp.2180-2186.

[3] Swor et al., "Prehospital 12-Lead ECG: Efficacy or Effectiveness?", Prehospital Emergency Care, July/September 2006, vol.10, no.3, pp.374-377.

[4] Dorais-Joncas, A., Elleuch, W. and Houle, A.C.,"A Conference Mechanism for the Simultaneous Transmission of Voice and Medical Information using SIP", 19th IEEE Canada Conference on Electrical Engineering and Computer Engineering, 2006, Ottawa, Ontario, Canada.

[5] TIA/EIA 102.BAAA Project 25 – FDMA Common Air Interface – New Technology Standards Project – Digital Radio Technical Standards, 1998.

[6] Internet Engineering Task Force, www.ietf.org/rfc.html

[7] Lennox, J. et Schulzrinne, H., "A protocol for reliable decentralized conferencing", NOSSDAV'03 : Proceedings of the 13th International Workshop on Networks and Operating Systems Support for Digital Audio and Video, New York, NY, United State of America, 72-81, ACM Press.

[8] ITU-T G.729, "G.729 : Coding of speech at 8 kbit/s using conjugate-structure algebraic-code-excited linear prediction (CS-ACELP)", 2007.

[9] H. Wang, F. Azuaje, B. Jung, N. Black, "A markup language for electrocardiogram data acquisition and analysis (ecgML)," BMC Medical Informatics and Decision Making, May 2003.

[10] Friedlander, J., Loganathan, K., Murphy, R., Pattabhiraman, R.V. and Vemuri, K.V., "Are you there? Reflections on presence server architectures", Bell Labs Technical Journal, 10(2006), 77-82.

[11] Rescola E. SSL and TLS Designing and Building Secure Systems. 2000; 449p.

[12] C. Peikari, A. Chuvakin. Security Warrior. 2004

[13] CNSS Policy No.15. National Policy on the Use of the Advance Encryption Standard (AES) to Protect National Security Systems and National Security Information. June 2003.

[14] RSA Laboratories, "PKCS #5: Password-Based Cryptography Standard", www.rsa.com/rsalabs/node.asp?id=2127

LISTE DES RÉFÉRENCES

Adams, C. et Ferrel, S. (1999) Internet x.509 public key infrastructure certificate management protocols - rfc2510. Http ://tools.ietf.org/html/rfc2510.

Ali, A. M. et Montgomery, M. A. (2005) Secure internet access and the role of the network smart card. *Proceedings of the Fourth IASTED International Conference on Communications, Internet, and Information Technology, CIIT 2005*, p. 7.

Baugher, M., McGrew, D., M.Naslund, Carrara, E. et Norrman, K. (2004) The secure real-time transport protocol (srtp) - rfc3711. Http ://www.ietf.org/rfc/rfc3711.txt.

Beauchemin, B., Beaulé, F., Boissonneault, M., Gauthier, J. D., Dubeau, G. et Ruel, F. (2007a) Présentation de l'audit final de l'équipe médicip.

Beauchemin, B., Beaulé, F., Boissonneault, M., Gauthier, J. D., Dubeau, G. et Ruel, F. (2007b) Rapport final médicip - collaboration multimédia sur protocole ip en mileu médical.

Campbell, J., Kleeman, D. et Ma, W. (2007) The good and not so good of enforcing password composition rules. *Information Security Journal : A Global PErspective*, volume 16, p. 6.

Dierks, T. et Rescola, E. (2008) The transport layer security (tls) protocol version 1.2 - rfc5246. Http ://www.ietf.org/rfc/rfc5246.txt.

Dorais-Joncas, A. (mai 2007) Mémoire de maîtrise : Un mécanisme de conférence basé sur le protocole sip pour la transmission simultanée de voix et de données médicales.

Information Sciences Institute, U. o. S. C. (1981a) Internet protocol darpa internet program protocol specification - rfc 791. Http ://www.ietf.org/rfc/rfc0791.txt.

Information Sciences Institute, U. o. S. C. (1981b) Transmission control protocol darpa internet program protocol specification - rfc 793. Http ://www.ietf.org/rfc/rfc0793.txt.

International Organization for Standardization (ISO) (2007) Iso 7816-x - smart card standards. Www.iso.org.

Jain, A., Bolle, R. et Pankanti, S. (1999) *Biometrics : Personal Identification in Networked Society*. Kluwer Academic Publishers.

Kaliski, B. (2003) Twirl and rsa key size. Http ://www.rsa.com/rsalabs/node.asp ?id=2004.

Kleinjung, T., Aoki, K., Lenstra, J. F. A. K. et Tome, E. (2010) Factorization of a 768-bit rsa modulus. Http ://eprint.iacr.org/2010/006.pdf.

M5T (2008) Http ://www.m5t.com/.

Marechal, S. (2007) État de l'art sur le cassage de mots de passe. SSTIC 2007 - Sumposium sur la Sécurité des Technologies de l'information et des Communications.

McDowell, M., Rafail, J. et Hernan, S. (2004) Choosing and protection passwords. Http ://www.us-cert.gov/cas/tips/ST04-002.html.

Noore, A. (2000) Highly robust biometric smart card design. *IEEE Transaction on Consumer Electronics*, volume 46, p. 5.

Nystrom, M. et Kaliksi, B. (2000) Pksc 10 : Certification request syntax specification - rfc2986. Http ://tools.ietf.org/html/rfc2986.

Rescorla, E. (2000) *SSL and TLS Designing and Building Secure System*. Addison-Wesley, 499 p.

Rosenberg, J., Schulzrinne, H., Camarillo, G. et Johnson, A. (2002) Sip : Session initiation protocol - rfc 3261. Http ://www.ietf.org/rfc/rfc3261.txt.

Saffron, C. et Ring, D. (1995) Protection of confidentiality in the computer-based patient records. *M.D. Computing*, volume 12, p. 187–192.

Salah, F. B. (2008) Mémoire de maîtrise : Service de présence dans un contexte médical.

Schulzrinne, H., Casner, S. et Frederick, R. (2003) Rtp : A transport protocol for real-time applications - rfc 3550. Http ://www.ietf.org/rfc/rfc3550.txt.

Timms, M. (1990) The pros and cons of proximity[access control]. *Security Management*, volume 47.

Wang, H., Azuaje, F., Jung, B. et Black, N. (2003) A markup language for electrocardiogram data acquisition and analysis (ecgml). *BMC Medical Informatics and Decision Making*.

Zygowicz, W. (Mars 2008) Electronic ems : Navigating the maze of paperless pcrs. Conférence EMS Today 2008, Baltimore.