Laboratoire de Transmissions de données
Rue de la Prairie 4
CH-1202 Genève
Switzerland

e i g

Windows 2000 Identification

Informations

Révision	*1.1.1*
Date	*04.12.2001*
Etudiant	*Mario Pasquali*
Professeur	*Gérald Litzistorf*
Fichier	*Windows 2000 Identification_1.1.1.doc*
Mots clés	*GINA, CAPI, CSP*

Description

Ce document explique les phases du développement de la chaîne complète d'identification et d'authentification sous Windows 2000 (GINA, CryptoAPI, CSP).

Enoncé

Histoire des révisions

Revision	Date	Comments
1.0.1	24.10.2001	Premier rendu du mémoire
1.0.2	14.11.2001	Deuxième rendu du mémoire Apport des améliorations suggérées par M. Litzistorf. Finalisation de la partie GINA. Amélioration de la partie CSP. Ajout du chapitre Windows 2000 Smart Card Logon.
1.0.3	28.11.2001	Troisième rendu du mémoire Apport des améliorations suggérées par M. Litzistorf. Apport des améliorations suggérées par L. Guinnard. Complété partie sur installation du Smart Card Logon. Ajout de la partie sur UsbGina. Ajout de la partie sur développement CryptoAPI. Mise à jours des annexes. Finalisation du document.
1.1.1	04.12.2001	Rendu final du mémoire

Conventions typographiques

Le texte courant utilise la police Verdana.

Les extraits de code et les noms des éléments logiciels utilisent la police `Courier`.

Les termes anglophones sont exprimés en *italique* et sont expliqués dans le glossaire.

Remerciements

A M. Litzistorf pour sa rigueur et son intérêt

A Laurent Guinnard pour sa disponibilité et ses conseils

A mes parents et mon amie pour le soutien moral qu'ils m'ont fourni

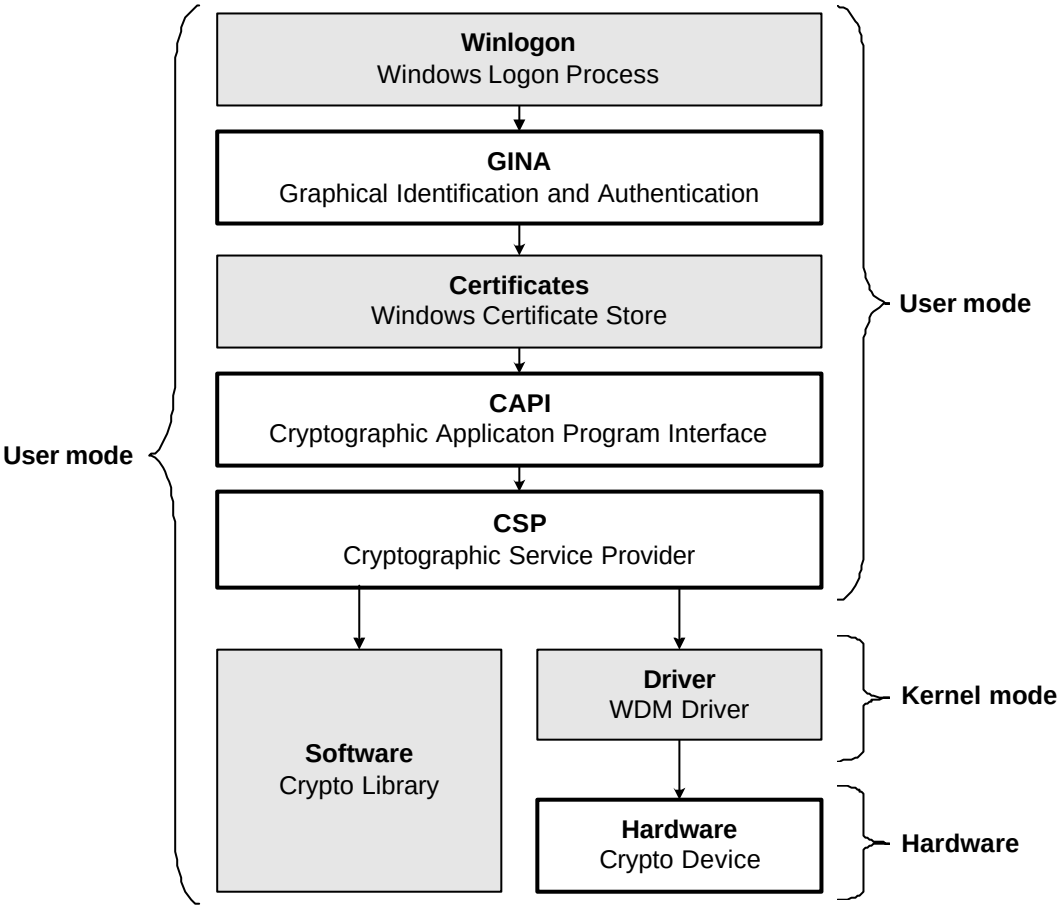
A mes collègues de laboratoire pour l'ambiance décontractée qu'ils ont su faire régner

Table des matières

I. Introduction

1. Objectif

L'objectif de ce travail de diplôme est d'obtenir une vision globale de tous les composants qui interagissent lors de l'identification et de l'authentification dans un système Windows 2000. La figure suivante représente ces différents composants sous la forme d'un empilement de couches logicielles et matérielles :



Les rectangles mis en évidence avec le bord en gras correspondent aux composants pour lesquels un développement logiciel a été effectué durant ce travail.

Voici une brève description des chapitres de ce document :

- Le chapitre III nommé **Windows 2000 Smart Card Logon** explique d'une façon détaillée l'installation d'une authentification basée sur une *smart card* dans une infrastructure à clé publique sous Windows 2000.

- Le chapitre IV décrit la problématique rencontrée lors du **Développement d'une DLL GINA Winlogon**. Cette dernière doit permettre à un utilisateur de s'authentifier sur le domaine local d'une station de travail Windows 2000 avec un jeton d'accès.

- Le chapitre V renseigne sur le **Développement d'un CSP CryptoAPI**. Ce dernier doit implémenter les fonctionnalités nécessaires à la génération d'un certificat numérique.

Vous trouverez une vue d'ensemble à ces technologies dans le chapitre II.

2. Annexes

Ref.	Titre
Annexe 1	Installation et liens intéressant du Platform SDK
Annexe 2	Création d'un projet MFC DLL avec Visual C++ 6.0
Annexe 3	Enregistrement d'une DLL à l'aide de 'regsvr32.exe'
Annexe 4	Trace d'une requête de certificat avec CspLog
Annexe 5	Trace d'une authentification avec GinaLog
Annexe 6	Code source de la DLL GINA de traçage (GinaLog)
Annexe 7	Code source de l'implémentation minimale de GINA (GinaLib)
Annexe 8	Code source de l'implémentation USB de GINA (UsbGina)
Annexe 9	Code source du CSP de traçage (CspLog)
Annexe 10	Code source de l'implémentation logicielle du CSP (CspLib)
Annexe 11	Commandes OpenSSL utilisées dans CspLib
Annexe 12	Guide de l'application didactique DidacSSL
Annexe 13	Code source de l'application didactique CryptoAPI (DidacSSL)

3. Acronymes et glossaire

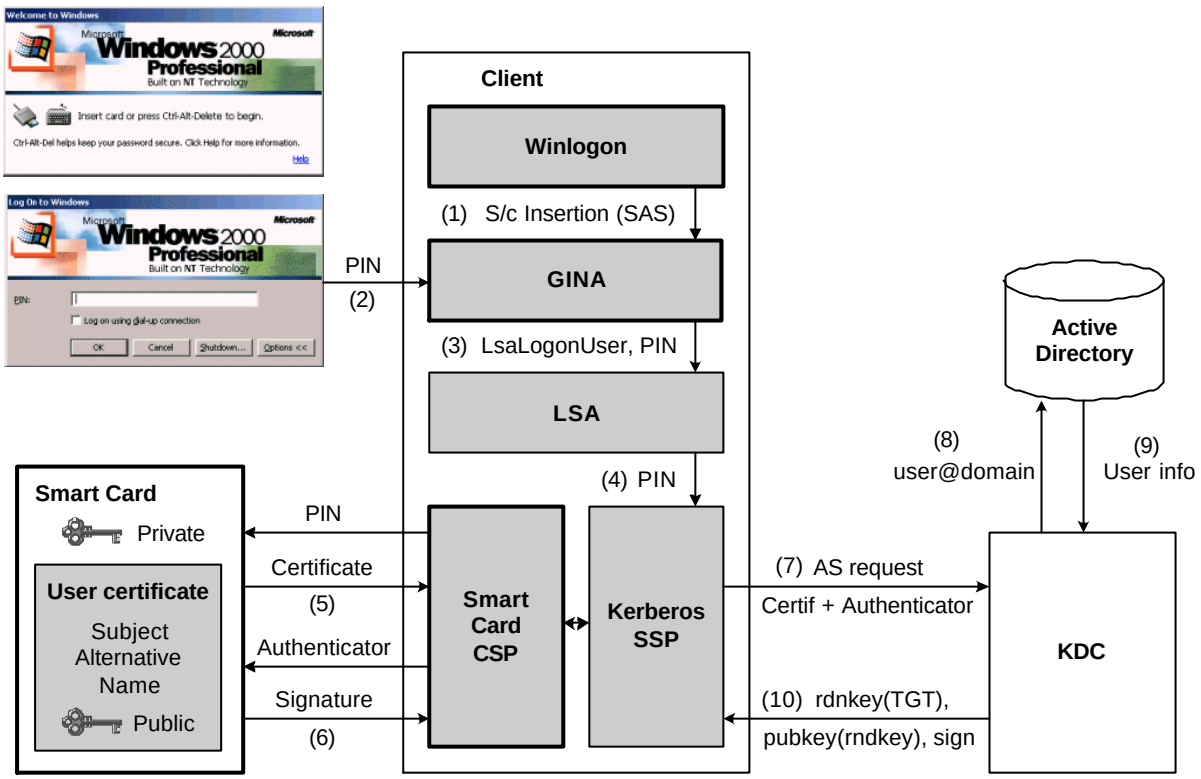
Terme	Définition
API	Voir Application Program Interface
Application Program Interface	Jeu de fonctions offertes par un système à une application
Checked	Version de Windows utilisée pour le test. Elle possède notamment tous les symboles de debug
CryptoAPI	Interface cryptographique de programmation développée par Microsoft
Cryptographic Service Provider	Interface entre CryptoAPI et un module logiciel ou matériel
Cryptoki	Norme PKCS#11. Définit l'interface entre un module matériel de cryptographie appelé token et une application
CSP	Voir Cryptographic Service Provider
CSPDK	CSP Developer's Kit
DLL	Voir Dynamic Link Library
Dynamic Link Library	Librairie contenant du code exécutable lié dynamiquement à une application
EFS	Voir Encrypted File System
Encrypted File System	Système de fichier chiffré. Permet d'obtenir une sécurité maximale pour les données critiques stockées sur un élément de stockage.
Entry Point	Voir Point d'entrée
Firmware	Voir micrologiciel
GINA	Voir Graphical Identification and Authentication
Graphical Identification and Authentication	DLL utilisée par le système d'exploitation à travers Winlogon pour gérer l'identification et l'authentification des utilisateurs
Handle	Valeur numérique obtenue par le système

Terme	Définition
	d'exploitation et qui identifie de façon unique un objet noyau
Hash	Fonction cryptographique qui fait correspondre une valeur de longueur fixe à un flux de données
Keyblob	Structure de données représentant une clé cryptographique
Keyset	Conteneur de clés cryptographiques. Endroit où les clés seront stockées dans un CSP
MFC	Voir Microsoft Fundation Classes
Micrologiciel	Logiciel pour microcontrôleur
Microsoft Fundation Classes	Classes C++ permettant d'accéder à toutes les API de Windows à travers le langage C++
MSDN Library	Documentation destinée aux développeurs utilisant les technologies Microsoft
Personal Identification Number	Numéro d'identification personnel. Ce numéro sert à identifier un utilisateur sur une carte à puce
PIN	Voir Personal Identification Number
PKCS	Voir Public Key Cryptography Standards
PKI	Voir Public Key Infrastructure
Point d'entrée	Fonctions exportées par une DLL
Public Key Cryptography Standards	Standards de cryptographie à clé publique définit par RSA
Public Key Infrastructure	Environnement où les utilisateurs peuvent accéder à divers services en utilisant une infrastructure à clé publique
Rivest-Shamir-Adelman	Fondateurs de l'entreprise RSA Security
RSA	Voir Rivest-Shamir-Adelman
SAS	Voir Secure Attention Sequence
Secure Attention Sequence	Séquence de touches qui débute le processus d'ouverture ou de fermeture de session. Par défaut, la séquence est Ctrl-Alt-Del
Simple Template Library	Librairie standard C++ qui offre diverses templates de listes, maps, strings, vecteurs, …
Smart card	Composant électronique de la taille d'une carte de crédit intégrant un processeur et de la mémoire. Aussi appelée carte à puce
STL	Voir Simple Template Library
Template	Classe C++ générique possédant un type non-définit à la création. Utilisée par exemple pour créer des listes chaînées génériques
Thread	Processus créé par un autre processus s'exécutant dans le même espace mémoire
Universal Serial Bus	Bus universel de transmission de données. Permet de connecter aisément des périphériques à un ordinateur personnel
USB	Voir Universal Serial Bus
Vendor request	Requête spécifique effectuée à un périphérique USB sur l'endpoint 0. Très simple à implémenter mais pas très performant lors de transferts de données importants
Winlogon	Processus d'identification de Windows NT. Utilise GINA pour l'interaction avec l'utilisateur

II. Tour d'horizon

1. Introduction

Le schéma suivant propose une représentation temporelle de l'authentification Windows 2000 dans une infrastructure à clé publique avec une carte à puce :



Les rectangles mis en évidence en gras correspondent aux composants qui seront traités plus en détail dans ce document. La suite de ce document détaille cette structure ainsi que les abréviations utilisées.

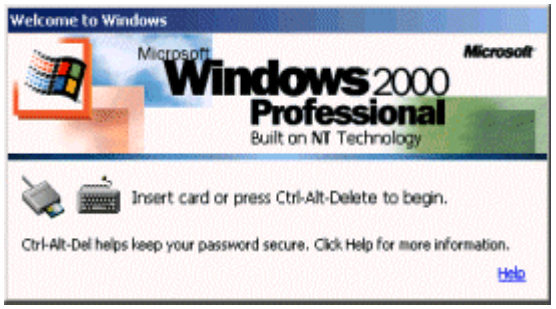
Un schéma similaire représentant les couches logicielles est disponible au chapitre I.

2. Winlogon et GINA

L'accès sécurisé à une ressource telle qu'un répertoire ou une imprimante est une problématique très importante des réseaux informatiques. Pour être évolutif et souple, un système d'exploitation doit offrir une solution permettant d'authentifier un utilisateur avec de nouveaux systèmes de reconnaissance ou pour accéder à un nouveau type de réseau.

Windows offre cette souplesse grâce au composant **Winlogon**. Ce dernier exécute un processus sécurisé qui gère l'identification et l'authentification sous Windows NT 3.5 et supérieur. Winlogon est le premier processus exécuté par le système lors du démarrage. Il enregistre alors le *SAS* (*Secure Attention Sequence*) par défaut (**Ctrl-Alt-Del**). Cela implique qu'aucun autre processus ne pourra intercepter cet événement.

Pour gérer l'**interaction avec l'utilisateur**, Winlogon utilise une DLL **remplaçable** nommée **GINA** (Graphical Identification aNd Authentication). GINA implémente la politique de sécurité pour l'authentification. Les boîtes de dialogues suivantes sont gérées par la DLL GINA livrée en standard avec Windows. Elles apparaissent lorsque le système est muni d'un lecteur de cartes à puces pour l'identification.



Les développeurs potentiels de composants GINA spécifiques sont par exemple les entreprises commercialisant des systèmes d'authentification. Si ces derniers doivent être utilisés dans Windows pour l'authentification, il est nécessaire de développer une DLL GINA permettant de communiquer avec le système.

Le chapitre IV présente plus de détails sur le développement d'un composant GINA.

3. Microsoft CrypoAPI

La sécurité des systèmes informatiques devient chaque jour plus critique. Un système d'exploitation doit intégrer des services cryptographiques de confidentialité, d'intégrité et d'authentification, nécessaires tant pour le développeur que pour l'utilisateur.

Microsoft CryptoAPI met ces services à disposition dans Windows. Il permet aux développeurs de sécuriser leur application d'une manière simple, puissante et évolutive. Il permet aussi aux développeurs de périphériques cryptographiques d'interfacer leurs produits avec les applications de manière transparente.

L'avantage de cette transparence est double. D'une part, une application sécurisée pourra être utilisée avec n'importe quel périphérique cryptographique installé sur le système. Inversement, un nouveau périphérique cryptographique est directement interfaçable avec une application sécurisée utilisant CryptoAPI.

CryptoAPI définit un jeu de fonctions permettant d'effectuer des opérations cryptographiques telles que le chiffrement/déchiffrement des données, la création et la gestion de clés symétriques et asymétriques, la génération de signatures numériques et leur vérification, etc.

Les exemples de périphériques qui peuvent être interfacés à CryptoAPI sont nombreux : générateurs de nombres aléatoires, accélérateurs cryptographiques, cartes à puces, etc. Chaque périphérique particulier doit posséder un composant logiciel associé.

Le présent document ne traite pas le développement d'une application basée sur CryptoAPI. Mon travail de semestre 2001 apporte plus d'informations sur le sujet. Une application didactique nommée DidacSSL a également été développée durant cette période. Le guide de cette dernière se trouve dans l'annexe 12 et son code source dans l'annexe 13.

Le chapitre V présente plus d'informations sur le développement d'un fournisseur de services cryptographiques CryptoAPI.

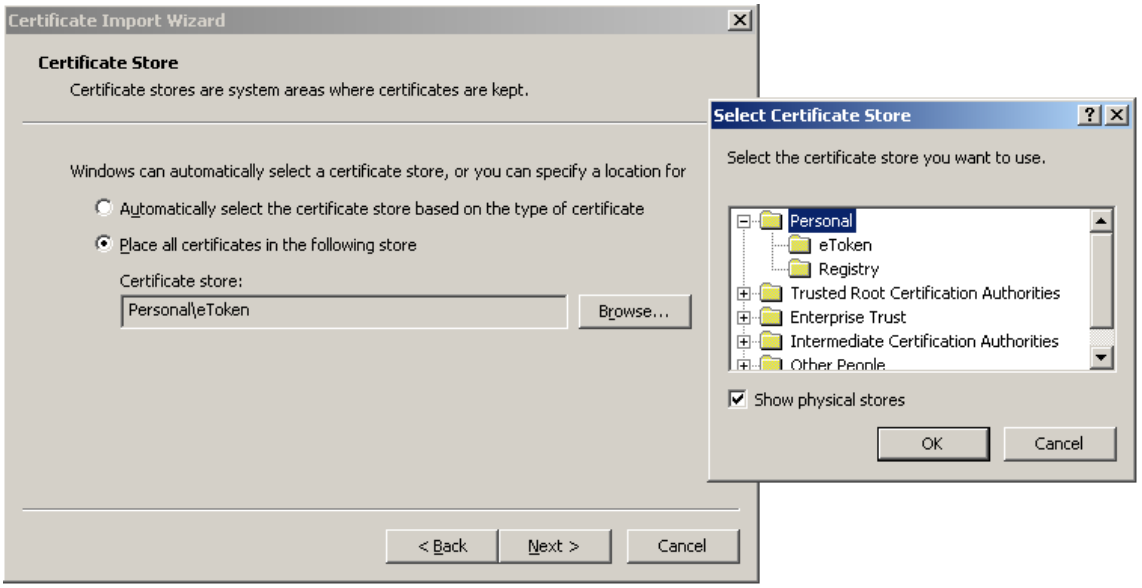
4. Certificate Store

Dans une infrastructure à clé publique, lorsqu'un utilisateur veut signer un document, il doit fournit ses informations d'identification ainsi que sa clé privée. Son certificat fournit les informations d'identifications, mais la clé privée doit être récupérée d'une autre manière.

Le composant Certificate Store a été développé pour réduire cette complexité. En utilisant cette technologie, un utilisateur qui veut obtenir un certificat doit spécifier des informations sur le CSP et sur le nom du conteneur de certificat à utiliser. Le certificat est stocké dans le conteneur spécifié et associé avec les autres éléments. Lorsqu'un utilisateur veut signer un document, il lui suffit d'identifier le certificat à utiliser. Le programme effectuant la signature sera alors capable de récupérer le certificat ainsi que la clé privée associée au certificat.

Plusieurs *stores* (magasins) sont présents sur un système Windows 2000 par défaut. Ces *stores* permettent de stocker les certificats (Personal, Trusted Root, Enterprise Trust, Intermediate Certification Authorities, Other People), les listes de révocation et les listes de confiance. Des *stores* spécifiques peuvent être ajoutés pour stocker des certificats dans la mémoire d'une carte à puce.

Lors de l'importation d'un certificat dans Internet Explorer (Tools > Internet Options > Contents > Certificates > Import), le système demande quel *store* utiliser et permet de naviger dans les *stores* disponnibles.



Plus d'informations sur cette technologie sont présentes dans le Platform SDK. Les liens sur les pages intéressantes sont listés dans l'annexe 1.

III. Windows 2000 Smart Card Logon

1. Introduction

Depuis sa version 2000, Windows supporte l'authentification dans une infrastructure à clé publique avec une carte à puce. Le kit 'eToken Enterprise' d'Aladdin est utilisé dans ce travail pour mettre en place cet environnement.

Le kit d'Aladdin offre les services suivants :

- Authentification Windows 2000
- Signature d'emails avec Microsoft Outlook
- Chiffrement d'emails avec Microsoft Outlook

Pour mettre en place le service d'authentification Windows 2000, l'installation des machines suivantes est nécessaire :

- Domain Controller
- Certification Authority
- Enrollment Station

La configuration de test choisie regroupe tous ces éléments dans une seule machine, mais la séparation de ces composants sur des machines différentes est aussi possible.

2. eToken d'Aladdin

Le eToken d'Aladdin est un périphérique USB permettant de stocker des certificats et d'effectuer des opérations cryptographiques. Du côté logiciel, ce composant peut être considéré comme un lecteur de carte à puce ayant une carte à puce intégrée et inamovible.

Ce composant existe en deux versions:

- La version **R2** permet de stocker des paires de clés asymétriques et des certificats. Toutes les opérations cryptographiques sont effectuées en logiciel sur la machine. Cette implémentation est de type *mixed* (voir chapitre V.8).

- La version **Pro** possède un processeur cryptographique qui permet d'effectuer toutes les opérations cryptographiques en matériel. Cette implémentation est de type *hardware* (voir chapitre V.8).

Aladdin fournit tous les modules logiciels permettant de mettre en place une infrastructure à clé publique (en anglais *Public Key Infrastructure, PKI*) dans un domaine Windows 2000.

Rainbow fournit une solution similaire à celle d'Aladdin, mais la documentation et le support sont de moins bonne facture. Pour ces raisons, seul le kit d'Aladdin va être testé et mis en œuvre.

Ces périphériques USB sont assimilés à des cartes à puces dans les fonctionnalités, mais leur sécurité physique n'est pas aussi bonne. Le document suivant explique des failles au niveau matériel et logiciel de ces composants :
http://www.atstake.com/research/reports/usb_hardware_token.pdf

3. Stockage des mots de passes dans Windows 2000

3.1 Introduction

Les mots de passe font actuellement partie des points les plus critiques pour la sécurité d'un système informatique. Les utilisateurs choisissent le plus souvent des mots de passe faciles à découvrir comme une suite de lettre, des mots courants ou le nom de leur chien. Pour augmenter la sécurité d'un réseau, un administrateur peut créer des règles concernant la complexité des mots de passe. Par exemple, il peut exiger que tout mot de passe contienne au moins six caractères dont au moins deux chiffres. Il peut aussi forcer les utilisateurs à changer leurs mots de passe tous les deux mois par exemple.

Divers outils permettent de pirater des mots de passe, le plus connu et médiatisé d'entre eux est sûrement L0phtCrack 2.5 (LC3). Les mots de passe peuvent être récupérés sur une machine Windows NT ou Windows 2000, mais aussi lus sur un réseau informatique en reniflant (*sniffer*) les paquets SMB. Vous trouverez cet outil en téléchargement à l'adresse suivante : *http://www.atstake.com/research/lc3/download.html.*

La suite du chapitre explique comment les mots de passe sont stockés dans les systèmes Windows NT et 2000.

3.2 Windows NT

Windows NT stocke des *hashs*, et non les mots de passe eux-même en clair. Un *hash* est une opération mathématique univoque permettant de faire correspondre une valeur de longueur fixe à des données de longueur variables. Cette valeur est stockée dans la base de données **SAM** du contrôleur de domaine. Cette dernière peut être trouvée dans le répertoire 'winnt\system32\config'. En utilisant les privilèges de l'administrateur, L0phtCrack peut récupérer les *hashs* dans la SAM et retrouver les mots de passe en exploitant les faiblesses de l'algorithme de hachage de NT.

Pour retrouver un mot de passe, L0phtCrack effectue en premier lieu une attaque par dictionnaire en itérant une liste de mots communs. Il hache chaque mot et compare le résultat avec la valeur récupérée dans la SAM. Si les valeurs sont identiques, L0phtCrack a trouvé le mot de passe. Si l'itération du dictionnaire est infructueuse, le programme va effectuer une attaque hybride en reprenant le dictionnaire et en ajoutant quelques caractères au début et à la fin de chaque mot. Finalement, si le mot de passe n'a toujours pas été trouvé, L0phtCrack va procéder à une attaque de force brute (*brut force attack*) en testant chaque combinaison possible de caractères.

Pour des raisons de compatibilité, NT utilise l'algorithme de hachage de *LAN Manager*. Ce dernier met les mots de passe en majuscules avant d'effectuer le *hash* et découpe les mots de passe trop longs en blocs de sept caractères ce qui facilite d'autant plus les attaques.

3.3 Windows 2000

La sécurité des mots de passe dans Windows 2000 a été nettement améliorée. Il utilise **Syskey** qui chiffre les *hashs* avant le stockage. Dans un Windows 2000 Professionnel, les comptes utilisateurs et les mots de passe associés sont toujours stockés dans la SAM. Par contre, un contrôleur de domaine Windows 2000 ne l'utilise plus au profit d'*Active Directory.* Cette dernière technologie rend l'opération plus compliquée, mais toujours possible car la dernière version de L0phtCrack permet d'y récupérer tout de même les *hashs*.

Pour ce faire, il utilise une technique appelée 'DLL injection'. Habituellement, les hashs sont récupérés par le processus lsa.exe et utilisés par la DLL msv1_0. Dans le cas de L0phtCrack, cette dernière DLL est remplacée par une autre nommée samdump.dll. Elle utilise les mêmes API internes que msv1_0.dll pour accéder aux hashs des mots de passe. Cela signifie qu'elle peut obtenir les hashs sans mettre en œuvre la complexité rencontrée pour les extraire et de les déchiffrer. La DLL n'a pas besoin de connaître les algorithmes ni les clés utilisées. Ce principe est basé sur le programme PwdDump2. Plus d'informations ainsi que le code source sont disponibles à l'adresse suivante :
http://www.webspan.net/~tas/pwdump2

4. Divers types de logon

Une carte à puce peut être utilisée pour authentifier un domaine Windows 2000 de trois manières différentes :

- **Interactive Logon**
  Utilise *Active Directory*, le protocole Kerberos version 5 et les certificats

- **Client authentication**
  Authentifie un utilisateur comparant son certificat avec un compte stocké dans *Active Directory* sans passer par Kerberos

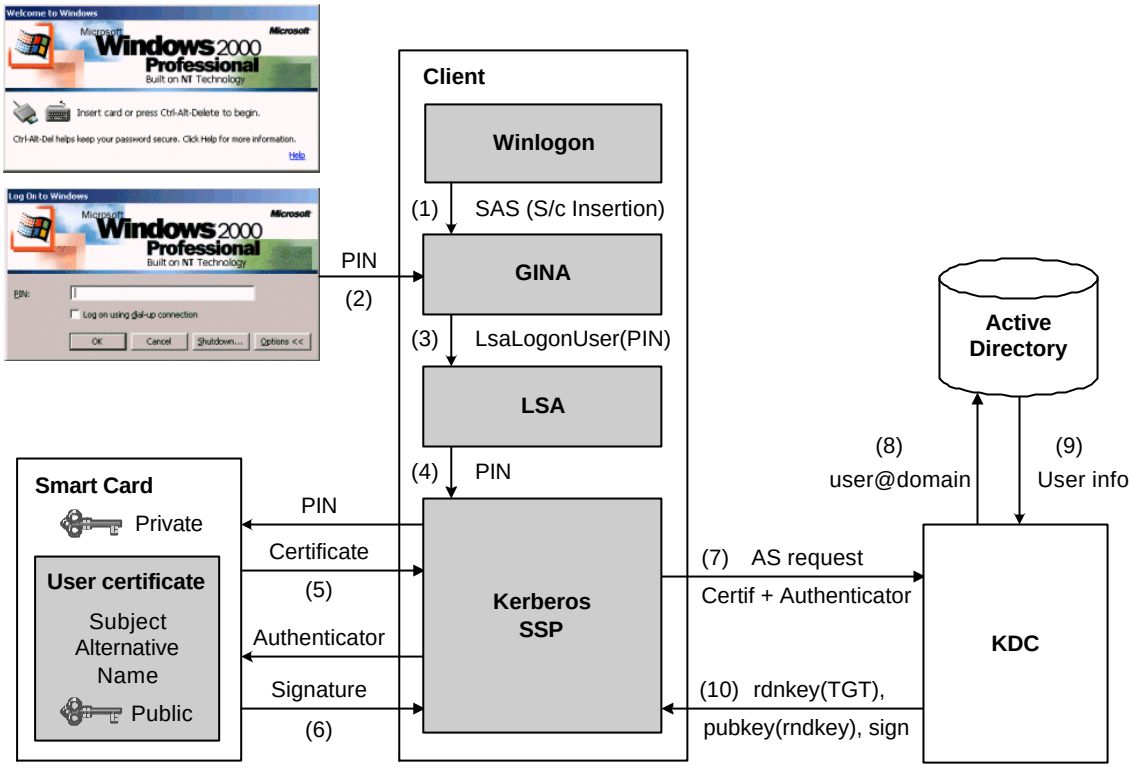
- **Remote Logon**
  Authentifie un utilisateur distant sur un compte *Active Directory* en utilisant son certificat et les protocoles *EAP* (*Extensible Authentification Protocol*) et *TLS* (*Transport Layer Security*)

L'*Interactive Logon* est celui qui est utilisé par défaut dans Windows 2000 lorsque l'ordinateur appartient à un domaine.

5. Interactive logon et Kerberos 5 avec l'extension PKINIT

La procédure de logon interactive utilise le protocole Kerberos 5 avec l'extension PKINIT. Cette dernière implémente un échange AS initial avec des clés asymétriques au lieu de clés symétriques pour être utilisé dans une PKI. Vous trouverez plus d'informations sur Kerberos dans le projet de semestre 2001 de Souchon.

Voici un schéma illustrant les diverses étapes composant l'extension PKINIT ainsi que l'explication verbale relative :



1. **Obtention du PIN**
   Le logon interactif utilisant une carte à puce commence lors de son insertion (1). Cette opération, appelée SAS (*Secure Attention Sequence*), correspond à la séquence Ctrl-Alt-Del d'un logon basé sur un mot de passe. Après l'insertion de la carte à puce, Winlogon appelle MSGINA qui invite l'utilisateur à entrer son PIN (*Personal Identification Number*) pour s'authentifier auprès de la carte à puce (2). Notez qu'il n'est pas nécessaire d'entrer un nom de domaine car l'utilisateur est identifié avec un **UPN** (*Unified Principal Name*, ex: user@domain).

2. **Requête de logon**
   MSGINA envoie le PIN à son LSA (*Local Security Authority*) grâce à la fonction LsaLogonUser (3). Ces informations vont ensuite être redirigées au client SSP Kerberos de la machine (4). Ce dernier peut ainsi récupérer le certificat de l'utilisateur dans la carte à puce (5). Le SSP génère un authentifieur (*authenticator)* qui contient essentiellement l'heure de la machine (*timestamp*). Cet authentifieur est signé avec la clé privée associée au certificat pour permettre au KDC d'authentifier l'utilisateur (6).

   Le SSP envoie ensuite une requête AS (*Authentication Service*) au KDC du contrôleur de domaine (7) pour obtenir un TGT (*Ticket Granting Ticket*). Le certificat est inclus dans les champs de pré-authentification de la requête AS ainsi que l'authentifieur.

3. **Vérification du certificat et de la signature**
   Pour répondre à la requête AS, le KDC doit vérifier qu'il peut faire confiance aux CA contenues dans le chemin de certification (*Certification Path*). Le KDC retourne une erreur si le certificat root n'est pas crédible (*trusted*), s'il ne trouve pas les certificats des parents ou si la liste de révocation ne peut être déterminée.

   Après cette vérification, le KDC utilise CryptoAPI pour vérifier la signature numérique de l'authentifieur avec la clé publique contenue dans le certificat. Le KDC doit encore valider le *timestamp* contenu dans l'authentifieur pour s'assurer que ce n'est pas une attaque de réplication.

4. **Consultation du compte et TGT**
   Après avoir vérifié que l'utilisateur est bien celui qu'il prétend être, le KDC demande les informations de compte au contrôleur de domaine correspondant au champ *Subject Alternative Name* du certificat (8). Les informations retournées (9) vont être utilisées pour construire un TGT. Le KDC chiffre le TGT avec une clé symétrique générée aléatoirement. Cette clé est elle-même chiffrée avec la clé publique de l'utilisateur et est ensuite inclue dans le champ de pré-authentification de la réponse. Le KDC signe cette réponse avec sa clé privée pour que le client puisse l'authentifier (10).

   Le client peut ensuite vérifier le chemin de certification du certificat du KDC, authentifier la signature de la réponse, déchiffrer la clé symétrique aléatoire avec sa clé privée et déchiffrer le TGT. Le protocole Kerberos standard est utilisé pour la suite des opérations.

6. Installation du contrôleur de domaine (Domain Controller)

Vous trouverez toutes les informations nécessaires pour l'installation du contrôleur de domaine dans le mémoire de diplôme 2001 de Souchon (Annexe 1).

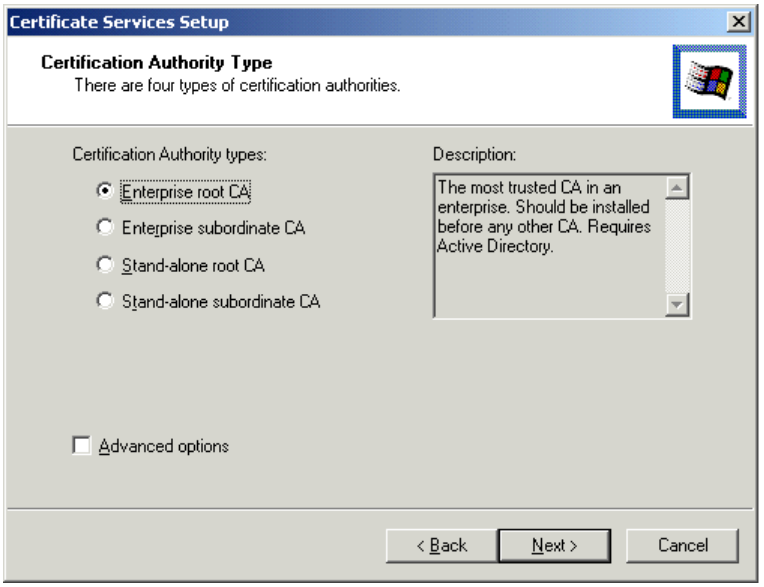
7. Installation de l'autorité de certification (Certification Authority)

Pour mettre en place un **Smart Card Logon** dans un environnement Windows 2000, il est nécessaire d'installer une autorité de certification qui délivrera les certificats pour les cartes à puces.

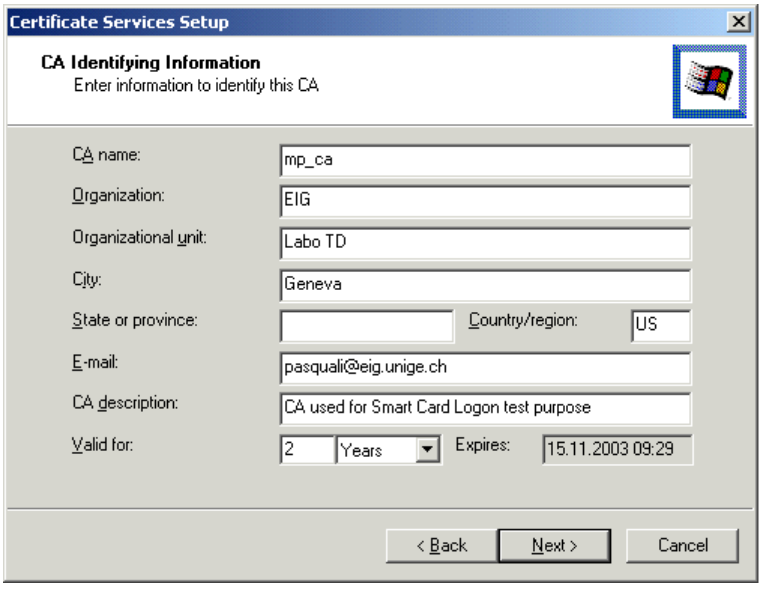
Avant de commencer, assurez-vous que le **serveur Web IIS** est présent sur la machine. Si ce n'est pas le cas, l'extension pour IIS ajoutant le support de requêtes de certificats ne sera pas installée.

1. **Installation de la CA**
   Pour installer l'autorité de certification de Microsoft, allez dans :
   'Start > Settings > Control Pannel > Add/Remove Programs > Add/Remove Windows Components > Certificate Services' et ajouter ce composant.

2. **Certification Authority Type**
   Pour le Smart Card Logon, il est nécessaire d'installer une Enterprise root ou subordinate CA pour qu'elle fasse partit du domaine. L'**Enterprise root CA** a été choisie pour la suite.



3. **CA Identifying Information**
   C'est ici que le nom de la CA va être spécifié. Voici les informations qui ont été entrées lors de cette installation :

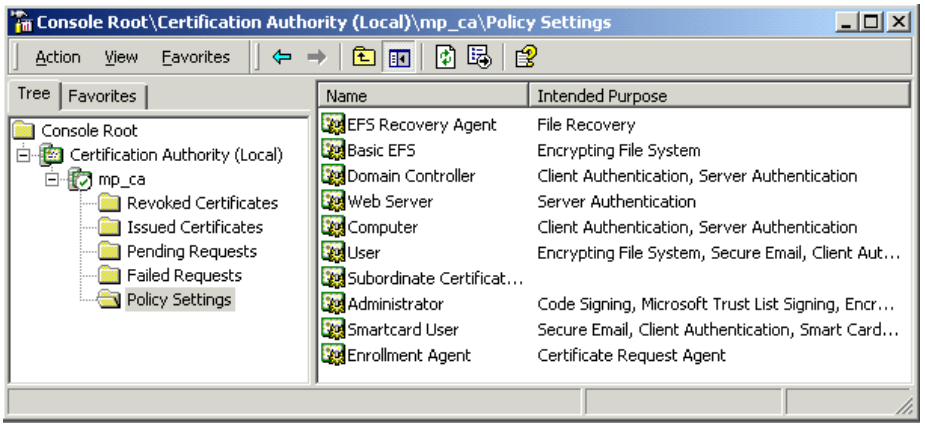


4. Utilisez les options par défaut pour les autres étapes. Acceptez d'arrêter IIS lorsque le programme vous le demande.

8. Configuration de l'autorité de certification

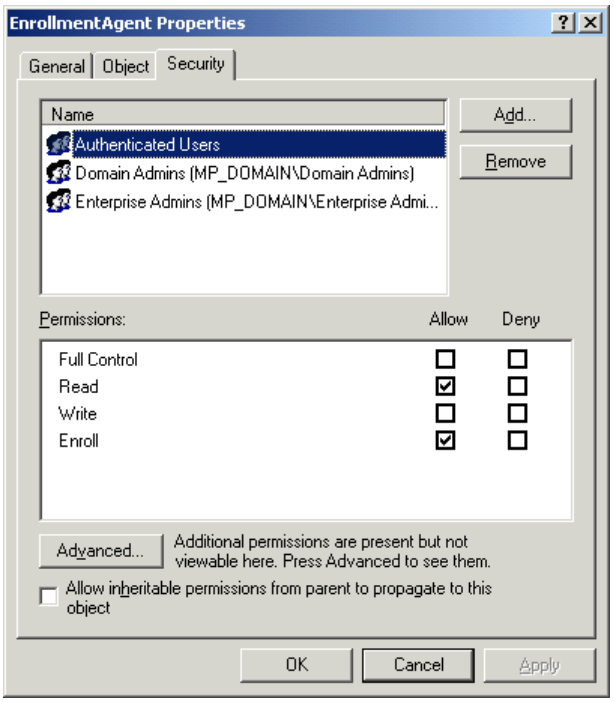
1. Exécutez MMC et ajouter le snap-in **Certification Authority**.

2. Dans Policy Settings, faire **New > Certificate to issue** et ajouter **Enrollment Agent** et **Smart Card User**.



3. Ajouter le snap-in **Active Directory Sites and Services** dans MMC. Si les services ne sont pas affichés, activer l'option dans View > Show Services Nodes.

4. Aller dans Services > Public Key Services > Certificates Templates. Afficher les propriétés de **EnrollmentAgent**, puis Security. Vérifier que tous les groupes possèdent les permissions **Read** et **Enroll**. Effectuer les mêmes opérations pour **MachineEnrollmentAgent** et **SmartCardUser**.

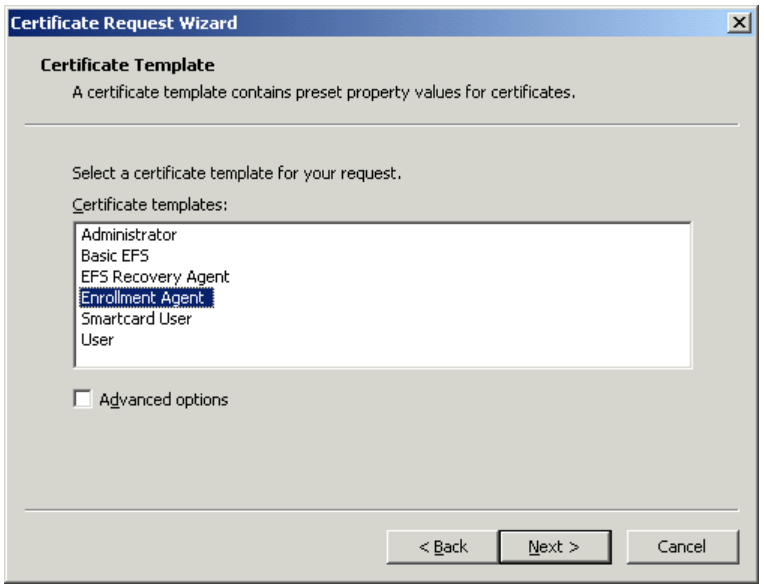


9. Installation de la station d'enregistrement (Enrollment Station)

Dans un réseau d'entreprise où la sécurité est de mise, l'agent d'enregistrement doit être installé sur une machine dédiée à cette tâche. Dans notre cas, la solution la plus simple est d'installer cet agent sur la même station que la CA. Les opérations suivantes peuvent être effectuées depuis le compte administrateur.

1. Exécutez MMC et ajouter le snap-in **Certificates** en sélectionnant **My User Account**.

2. Allez dans Personnal > Certificates, faire All Tasks puis Request New Certificate. Sélectionnez la template **Enrollment Agent**.

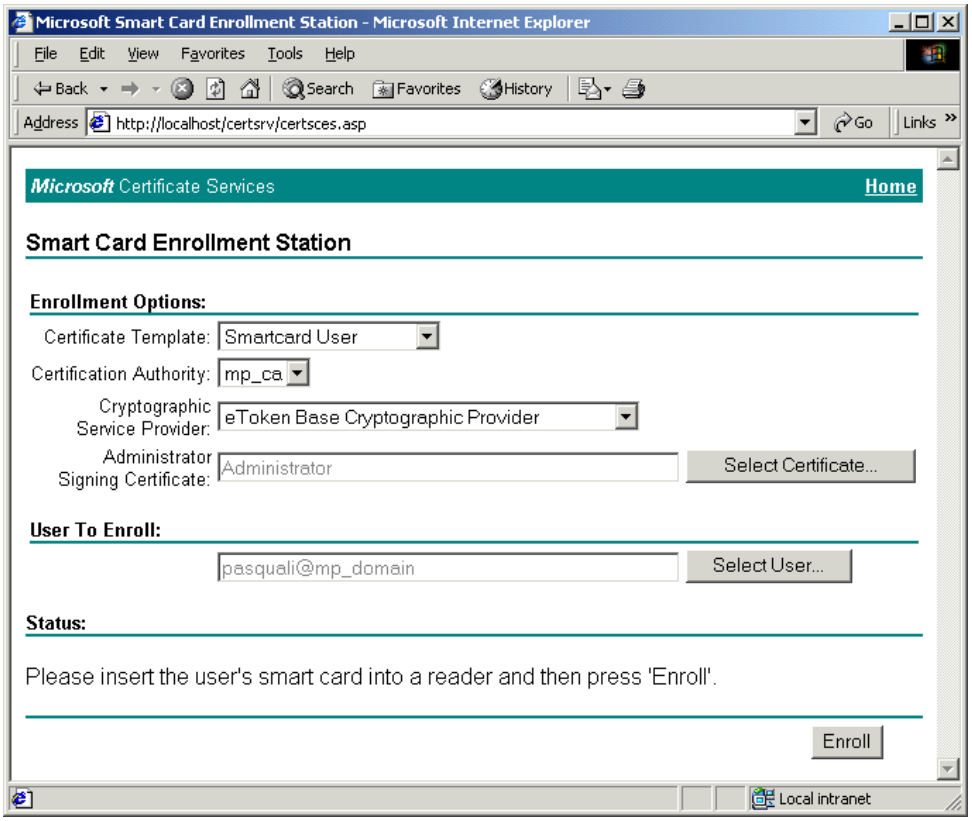


3. Cliquez sur **Install Certificate**.

10. Enregistrement d'un utilisateur de smart card

Avant d'effectuer ces opérations, il est nécessaire d'installer le **SDK eToken d'Alladin**. Si aucun utilisateur n'a été créé dans votre domaine, il est préférable d'en ajouter un qui sera utilisé pour l'enregistrement lors de la génération du certificat.

1. Lancer Internet Explorer et entrez l'adresse **http://localhost/certsrv** puis Request a Certificate > Advanced Request > Request a certificate for a smart card. Dans Certificate Template mettre **smart card user**, sélectionnez le **CSP eToken** ainsi que l'utilisateur de votre choix.



2. Cliquez sur **Enroll** et attendez que le certificat soit délivré.

11. Configuration de la machine de logon

La machine qui va être utilisée pour tester le Smart Card Logon doit être installée avec Windows 2000 et doit faire partie du domaine précédemment installé. De plus, le **eToken SDK** doit aussi être installé. Assurez-vous que l'eToken que vous allez utiliser pour le Smart Card Logon a été au moins connecté une fois sur l'ordinateur et que sa LED s'allume. Si ce n'est pas le cas, il y a un problème au niveau du driver qu'il faut corriger avant de poursuivre l'installation.

Il est ensuite nécessaire de redémarrer la machine et d'insérer la carte à puce possédant le certificat utilisateur délivré dans les étapes précédentes. La session utilisateur doit s'ouvrir après quelques secondes (env. 60s avec la version Pro car les opérations cryptographiques sont effectuées en matériel).

12. Ressources utiles

- **Microsoft Smart Card Logon White paper**
  http:// www.microsoft.com/windows2000/docs/sclogonwp.doc
  Bon document expliquant les mécanismes de logon sous Windows 2000

- **Troubleshooting Windows 2000 PKI Deployment and Smart Card Logon**
  http://www.microsoft.com/windows2000/docs/smrtcrdtrbl.doc
  Bon document donnant quelques réponses à des problèmes courants

- **Microsoft Windows 2000 Kerberos Authentication White paper**
  http://www.microsoft.com/windows2000/docs/kerberos.doc
  Très bon document sur Kerberos et son extension PKINIT

- **Deploying Smart Card**
  http://www.microsoft.com/technet/security/smrtcard/smartc09.asp
  Très bon document sur l'installation d'un Smart Card Logon

- **Integration Guide for Windows 2000 Smart Card Network Logon**
  ftp://ftp.ealaddin.com/pub/etoken/enterprise/eToken_W2K_Smartcard_Logon_2-0.zip
  Très bon document d'Aladin sur l'installation d'un Smart Card Logon

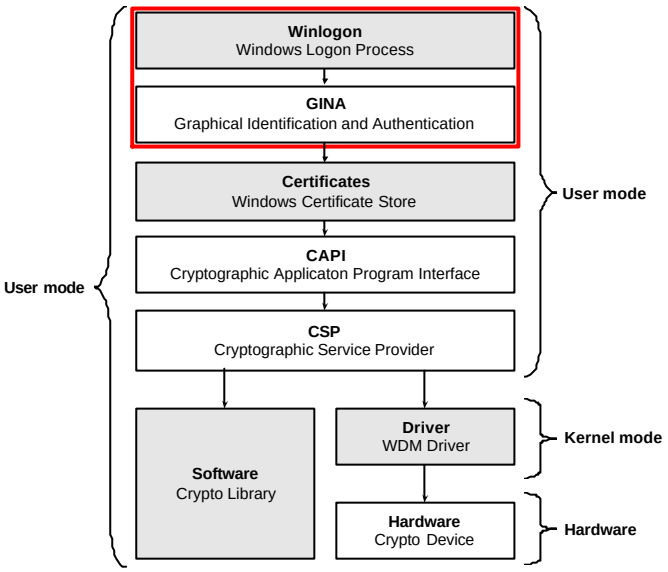
IV. Développement d'une DLL GINA Winlogon

1. Objectif

L'objectif de cette partie est de développer une DLL GINA permettant d'authentifier un utilisateur dans le domaine local d'un système Windows 2000. Les données utilisateurs sont entrées dans une boîte de dialogue, puis passées au sous-système d'authentification.

Dans une deuxième étape, la DLL développée sera modifiée pour être utilisée avec du matériel d'authentification spécifique (un module connecté sur USB).

2. Vue d'ensemble



L'architecture de Winlogon et basée sur trois composants :

- L'exécutable **Winlogon.exe**.

- Une DLL **d'identification et d'authentification graphique** (GINA – Graphical Identification and Authentication).

- Plusieurs DLL de **fournisseurs de réseau**. Ces fournisseurs permettent au système d'interagir avec des réseaux non-compatibles avec Windows. Ce point ne sera pas traité dans ce document. Vous trouverez de plus amples informations dans l'annexe 1 qui indique des liens intéressant du Platform SDK.

Winlogon est le premier processus qui est exécuté par le système lors du démarrage de Windows. Il accomplit les tâches suivantes lors de sa création :

- Il enregistre auprès du système le *SAS* (*Secure Attention Sequence*) par défaut (**Ctrl-Alt-Del**). Le noyau appelle Winlogon à chaque fois que cette séquence est entrée par un utilisateur. Cela implique qu'aucune autre application ne peut prendre le contrôle de la machine lorsqu'un SAS est entré, ni intercepter ce dernier.

- Il crée une station de travail de base possédant une souris, un clavier et un écran. Sur cette station, il crée un bureau pour l'utilisateur, un bureau système et un bureau pour le l'économiseur d'écran. Le bureau système est un bureau sécurisé où Winlogon exécute ses tâches et affiche les boîtes de dialogues GINA.

Une fois que l'utilisateur a entré son nom et son mot de passe, Winlogon envoie ces informations à son *LSA* (*Local Security Authority Server*) qui va authentifier le mot de passe. Il génère ensuite un jeton d'accès que Winlogon va utiliser pour créer le *shell* de l'utilisateur.

GINA est une DLL Windows standard possédant des points d'entrée spécifiques définis dans le Platform SDK. Il existe différentes versions de GINA qui ont évolué avec les versions des systèmes d'exploitation.

Version de GINA	Version de Windows correspondante
1.0	Windows NT 3.5.0
1.1	Windows NT 3.5.1
1.2	Windows NT 4.0
1.3	Windows 2000
1.4	Windows XP

Les tâches suivantes sont gérées par GINA :

- **Reconnaissance des SAS**
  Une DLL GINA de remplacement peut avoir ses propres SAS et est donc responsable de les reconnaître. La notification d'un SAS au système est générée grâce à la fonction `WlxSASNotify`. Si aucun SAS spécifique n'est utilisé, celui par défaut de Winlogon reste valable (Ctrl-Alt-Del) et est directement géré par le système d'exploitation.

- **Interface utilisateur**
  GINA est responsable de toute interaction avec l'utilisateur lors de l'identification et l'authentification. La DLL doit donc afficher une interface utilisateur lorsqu'elle veut rassembler les informations nécessaires au logon. Le composant GINA standard affiche la boîte de dialogue suivante que nous connaissons bien :

- **Création du shell**
  Lorsque l'identification auprès du système est acceptée, GINA est chargée restaurer l'environnement utilisateur (tel que les connexions réseau), le profil (tel que les couleurs, polices, économiseurs d'écrans) et d'exécuter des scripts de logon.

Par défaut, le processus Winlogon utilise la DLL GINA standard fournie avec Windows. Cette DLL se trouve à l'emplacement 'c:\winnt\system32\msgina.dll'. Pour la remplacer par une DLL GINA spécifique, il est nécessaire d'indiquer à Winlogon son emplacement à l'aide d'une clé dans la base de registre. Le point IV.11 donne plus d'informations à ce sujet.

3. Authenticité d'une DLL GINA

Contrairement aux CSP (voir point V.4), GINA ne possède pas de mécanisme permettant de vérifier l'authenticité de sa DLL. Cela signifie qu'il suffit d'avoir accès à la base de registre d'un système Windows pour échanger la DLL standard avec une autre DLL qui pourrait par exemple sauvegarder les mots de passe de tous les utilisateurs dans un fichier à l'intention d'une personne malintentionnée.

La sécurité de GINA n'est assurée que par le système de fichier de Windows. Il faut être conscient que si une personne a accès à une machine et qu'il peut la démarrer à partir d'une disquette, il sera aussi capable de remplacer la DLL GINA par sa DLL de remplacement. Une solution à ce problème est d'utiliser une partition *EFS* (*Encrypted File System*) qui empêche la lecture et l'écriture des fichiers depuis un autre système d'exploitation.

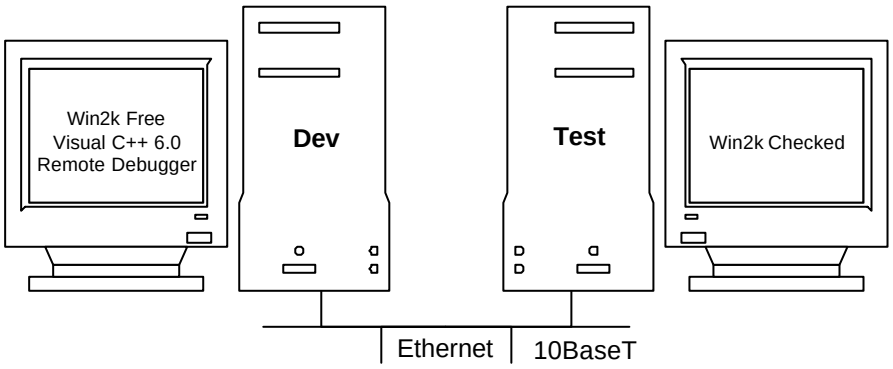
4. Environnement de test

4.1 Introduction

Le déverminage d'une DLL GINA ne peut pas être fait avec des outils habituels, comme par exemple le débogueur de Visual C++, car cette DLL est la première à être chargée dans le système d'exploitation, avant même le débogueur. Il faut donc mettre en place un environnement de test adéquat.

4.2 Remote debugger

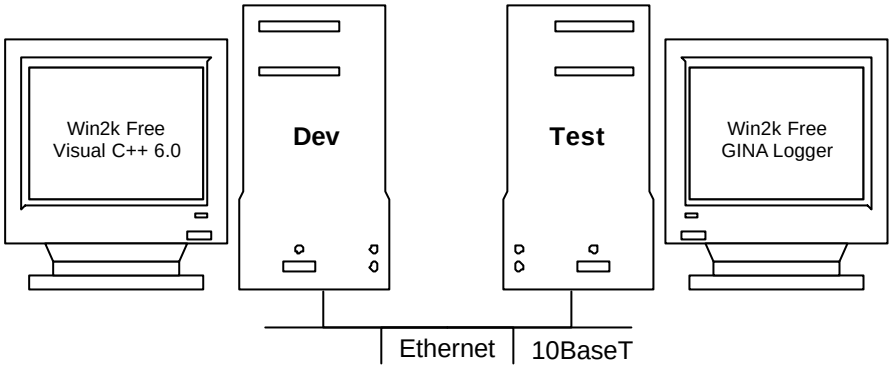
Microsoft préconise d'utiliser deux machines, la première dédiée au test et l'autre au développement. L'installation d'une version *checked* de Windows sur la machine de test permet de générer des traces détaillées du Winlogon ainsi que de déboguer à distance depuis la machine de développement.



Les inconvénients de cette solution sont multiples. Premièrement, la version *checked* de Windows 2000 est beaucoup plus lente que la version normale, appelée *free*. Cela se traduit par des temps de démarrage beaucoup plus longs qu'à l'habitude. De plus, elle est complexe à mettre en œuvre car les outils fournis manquent de support. Pour ces raisons, cette solution n'a pas été choisie pour ce développement.

4.3 Logger

Une autre solution plus simple est de générer un fichier log à partir de la DLL GINA. Il est ainsi possible de suivre exactement à l'évolution de la DLL.



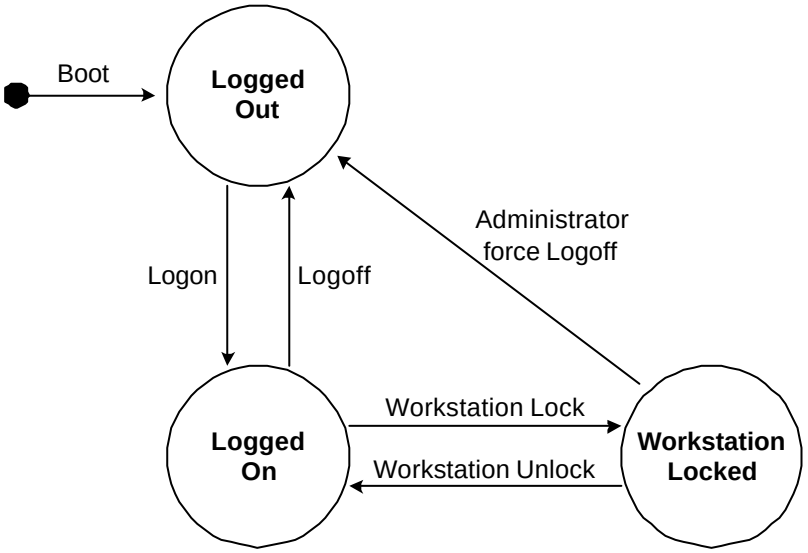
L'échange d'une DLL entre la machine de développement et la machine de test se fait par un répertoire partagé. Un problème subsiste toutefois car on ne peut pas remplacer une DLL qui est utilisée par le système. Il faut donc utiliser deux noms de DLL différents qui vont être utilisés alternativement. Une méthode très pratique pour copier rapidement une DLL dans le répertoire system32 consiste à ajouter un raccourci vers ce répertoire dans le menu 'Send to' de Windows.

5. Création d'un projet GINA avec Visual C++

Vous trouverez le détail la création d'un projet DLL MFC avec l'environnement Visual C++ dans l'annexe 2.

6. Machine d'état de Winlogon

La machine d'état de Winlogon possède trois états distincts. A un moment donné, Winlogon ne peut être que dans un de ces états état. Le diagramme qui suit représente cette machine d'état et les transitions associées :



Au démarrage du système d'exploitation, Winlogon se met dans l'état **Logged Out**. Une fois que l'utilisateur a fourni son nom et son mot de passe et qu'il a été authentifié auprès du système, Winlogon passe dans l'état **Logged On**. Si l'utilisateur verrouille manuellement sa station de travail ou que l'écran de veille s'enclenche, Winlogon se met dans l'état **Workstation Locked**.

7. Gestion des événements SAS par Winlogon

Winlogon peut recevoir des événements SAS depuis plusieurs sources :

- Le SAS par défaut (Ctrl-Alt-Del) est détecté par le noyau,
- Les SAS des cartes à puces sont détectés par le gestionnaire de ressources,
- Les SAS personnalisés sont détectés par la DLL GINA de remplacement.

Pour répondre à un événement SAS, Winlogon doit déterminer l'état courant du bureau et doit ensuite appeler le point d'entrée GINA correspondant :

- **WlxLoggedOutSas** lorsque Winlogon est dans l'état *Logged Out*.
- **WlxLoggedOnSas** lorsque Winlogon est dans l'état *Logged On*.
- **WlxWkstaLockSas** lorsque Winlogon est dans l'état *Workstation Locked*.

Ces fonctions qui vont provoquer les transitions dans la machine d'état de Winlogon.

8. Remplacement de la DLL GINA par défaut

Pour remplacer la DLL GINA par défaut par une autre DLL, il faut ajouter une valeur dans la base de registre qui indique à Winlogon le nom de la DLL de remplacement. Cette valeur est la suivante :

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon]
"GinaDLL"="mygina.dll"

Il est préférable que l'installation soit faite par la DLL elle-même. Le programme regsvr32.exe permet d'exécuter des points d'entrée d'une DLL à cet effet. L'annexe 3 contient plus d'informations sur le sujet.

9. Création d'une DLL de traçage

La première phase du développement consiste à déterminer l'ordre selon lequel les différents points d'entrée sont appelés ainsi que le nombre minimum de points d'entrée à implémenter pour ouvrir une session.

Le principe de la DLL de traçage est expliqué au point V.9 dans le chapitre CSP. Vous trouverez le code source de GinaLog dans l'annexe 6 pour plus de détails sur l'implémentation.

Lorsqu'une DLL de traçage a été développée, on peut remplacer la DLL par défaut par celle-ci en ajoutant une clé dans la base de registre comme indiqué au point IV.8. L'appel de chaque point d'entrée est alors enregistré dans un fichier log.

10. Implémentation des points d'entrée d'une DLL GINA

La trace de la DLL GINA de base lors de l'ouverture d'une session a montré que les points d'entrée suivants doivent être implémentés :

- **WlxNegotiate**
  Ce point d'entrée permet à Winlogon et à GINA de s'assurer qu'ils se supportent mutuellement. Si la version de Winlogon n'est pas supportée par GINA, une bonne idée est de désinstaller la DLL pour pouvoir redémarrer normalement (voir GINALIB1 dans code source du projet GinaLib, annexe 7).

- **WlxInitialize**
  Le contexte d'une DLL GINA est passé par Winlogon à chaque point d'entrée. Un pointeur sur un espace mémoire est créé lors de *WlxInitialize* (voir GINALIB2). Ce contexte contient des données collectées durant tout le processus de logon qui peuvent être utiles pour la suite des opérations.

- **WlxDisplaySASNotice**
  L'utilité de ce point d'entrée est d'indiquer à l'utilisateur ce qu'il doit faire pour obtenir le dialogue de login (SAS). Une version simplifiée pourrait directement envoyer à Winlogon le SAS par défaut.

- **WlxLoggedOutSAS**
  C'est le point d'entrée le plus important de la phase de logon. Il est chargé d'afficher un dialogue à l'utilisateur l'invitant à entrer son nom et son mot de passe. Avec ces informations, il est possible d'appeler la fonction *LogonUser* (voir GINALIB3) pour authentifier la validité des informations.

- **ActivateUserShell**
  C'est ici que l'environnement utilisateur est chargé. Pour cela, il faut appeler la fonction *CreateProcessAsUser* en ayant pris le soin d'avoir modifié l'utilisateur courant avec la fonction *ImpersonateLoggedOnUser* (voir GINALIB4).

Ces points d'entrée ont été implémentés avec succès (voir code source). La DLL GINA permet d'authentifier un utilisateur dans le domaine local.

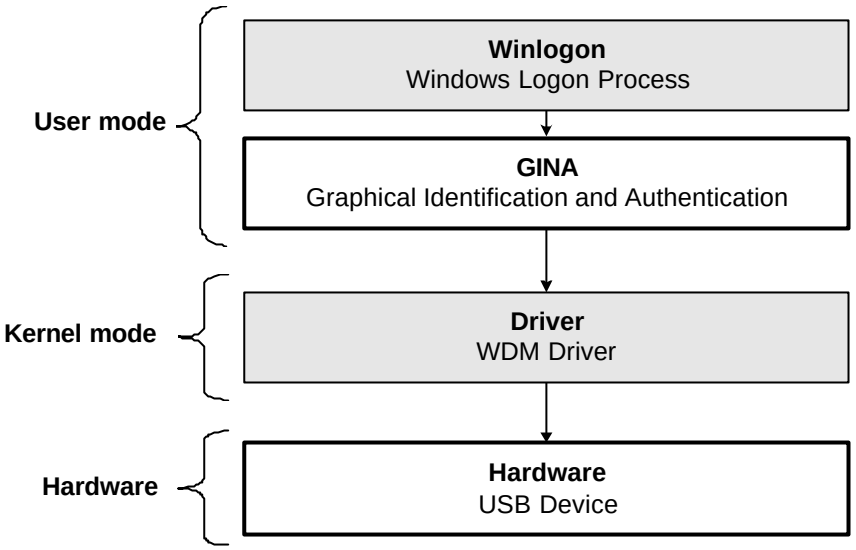
11. Implémentation d'une DLL GINA associée à du matériel spécifique

11.1 Introduction

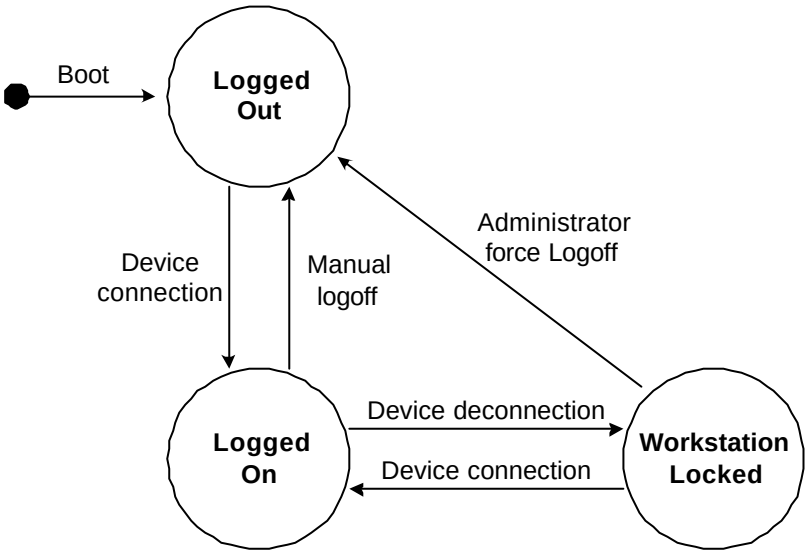
L'étape suivante du développement est d'interfacer la DLL GINA réalisée avec du matériel spécifique, comme par exemple :

- Une carte d'authentification (cartes à puces, carte sans-fil, etc.)
- Une authentification biologique (empreintes digitales ou de l'iris, etc.)

La figure suivante représente l'empilement de couches intervenant lors de cette étape :



Pour simuler l'un de ces dispositifs, une carte USB va être utilisée. Lors de sa connexion, GINA va récupérer les informations utilisateurs et les authentifier auprès du système. Lors de sa déconnexion, la station de travail va être verrouillée jusqu'à la reconnexion de la carte. Voici le diagramme d'état de Winlogon appliqué à cette implémentation :



Aucun mécanisme de chiffrement n'a été implémenté lors des transmissions USB. Lors d'une implémentation avec un périphérique d'authentification évolué, il sera nécessaire de mettre en œuvre un mécanisme de chiffrement adéquat pour augmenter la sécurité du système.

La suite du chapitre va décrire les différentes parties du développement.

11.2 Partie matérielle

Une carte didactique USB a été réalisée dans le cadre du laboratoire de transmissions de données lors d'un travail précédent. Cette carte est essentiellement composée du microcontrôleur Cypress EZ-USB de type Intel 8051, d'un affichage LCD et d'une mémoire non-volatile de type EEPROM utilisée pour stocker le code du micrologiciel. Une partie de cette mémoire va être aussi utilisée pour stocker le nom de l'utilisateur et son mot de passe.

Un micrologiciel universel avait également été développé et permettait d'accéder à tous les composants présents sur la carte à travers des *vendor requests* du bus USB. Ce micrologiciel est utilisé pour accéder à la mémoire EEPROM depuis GINA.

11.3 Pilote USB

Le pilote USB standard de Cypress est utilisé pour interfacer la carte USB avec le système d'exploitation. La dernière version de ce pilote est supportée par Windows 2000.

11.4 Librairie d'accès

Une librairie d'accès au périphérique USB a été écrite pour réaliser les tâches suivantes :

- Vérification de la présence du périphérique
- Lecture et écriture du nom de l'utilisateur
- Lecture et écriture du mot de passe

Cette librairie par être communément utilisée par l'application d'inscription et GINA.

11.5 Application d'inscription

Il est nécessaire de pouvoir stocker simplement un nom d'utilisateur et un mot de passe dans la mémoire de la carte en passant par le bus USB. Pour cela, une petite application a été développée en utilisant la librairie d'accès précédemment décrite.

11.6 Partie GINA

La DLL GINA est chargée de détecter les connexions et les déconnexions du périphérique USB et de générer les SAS personnalisés. Pour cela, il est nécessaire de créer un thread lors de l'initialisation de la DLL qui va s'occuper de cette tâche en parallèle aux autres fonctions (voir USBGINA1 dans code source). Lors de la connexion d'un périphérique, un événement est généré pour permettre à la boîte de dialogue en cours de se fermer par elle-même (vois USBGINA2). Ceci est nécessaire car les boîtes de dialogues utilisées sont bloquantes.

Quatre boîtes de dialogues sont affichées à l'utilisateur :

- **CNoticeDialog** invite l'utilisateur à s'authentifier auprès du système. Cette boîte sera fermée grâce à l'événement généré lors de la connexion.

- **CLogonDialog** affiche un message de bienvenu à l'utilisateur et lui demande de patienter durant le chargement de sa configuration personnelle.

- **COptionsDialog** est affiché à l'utilisateur lorsque le SAS par défaut (Ctrl-Alt-Del) est entré et que l'utilisateur est authentifié. Cette boîte permet d'afficher le gestionnaire de tâches et d'effectuer d'autres opérations utiles.

- **CLockedDialog** est affiché lorsque la station de travail est bloquée. Ce dialogue indique quel utilisateur a bloqué la station. Cette boîte disparaît lorsque l'utilisateur insert son périphérique. Une vérification supplémentaire est nécessaire pour vérifier que le périphérique connecté possède effectivement les informations correctes (voir USBGINA3).

Voici à quoi ressemble la boîte de dialogue CNoticeDialog :

L'annexe 8 qui contient le code source de cette DLL GINA.

12. Points d'entrée de GINA

Une DLL GINA doit implémenter tous les points d'entrées définis dans le Platform SDK en fonction de la version du système d'exploitation désiré. Ces points d'entrée permettent de gérer toute l'interaction avec Winlogon.

La liste suivante décrit tous les points d'entrée d'une DLL GINA version 1.3 (Windows 2000) :

Points d'entrée	Description
Init	
`WlxNegotiate`	Permet à la DLL GINA de vérifier qu'elle supporte la version de Winlogon installée
`WlxInitialize`	Winlogon appelle cette fonction pour chaque station présente sur l'ordinateur. C'est ici que le contexte est retourné par GINA
Logon	
`WlxDisplaySASNotice`	Affiche un message à l'utilisateur avant de s'identifier. Dans la DLL de base, cette fenêtre demande à l'utilisateur de taper Ctrl-Alt-Del
`WlxLoggedOutSAS`	Cette fonction est appelée lorsqu'un SAS est détecté et qu'aucun utilisateur n'est identifié. Cette fonction appelle *LogonUser* pour indiquer les informations de l'utilisateur au système
`WlxActivateUserShell`	Winlogon appelle cette fonction après qu'un utilisateur ait été identifié auprès du système pour activer son environnement
Logged on	
`WlxLoggedOnSAS`	Cette fonction est appelée lorsqu'un SAS est détecté pendant qu'un utilisateur est identifié et que la machine n'est pas bloquée. Un dialogue est affiché et permet d'éteindre la machine, de la bloquer ou de se déconnecter
`WlxDisplayLockedNotice`	Winlogon appelle cette fonction lorsque la machine est dans un état bloqué
`WlxWkstaLockedSAS`	Cette fonction est appelée lorsqu'un SAS est détecté et que la machine est bloquée
`WlxScreenSaverNotify`	Cette fonction est appelée pour indiquer à GINA que l'écran de veille va être affiché. GINA peut à ce moment indiquer à Winlogon si un mot de passe est nécessaire pour en sortir
`WlxStartApplication`	Winlogon appelle cette fonction lorsqu'il doit

Points d'entrée	Description
	exécuter une application dans le contexte de l'utilisateur
WlxIsLockOk	Winlogon appelle cette fonction avant d'essayer de bloquer la machine
WlxIsLogoffOk	Winlogon appelle cette fonction lorsque l'utilisateur commence un logoff
Logoff	
WlxLogoff	Winlogon appelle cette fonction pour indiquer à GINA que l'utilisateur va être déconnecté
WlxShutdown	Winlogon appelle cette fonction avant d'éteindre la machine
User Messages	
WlxDisplayStatusMessage	Winlogon appelle cette fonction pour afficher un message à l'utilisateur
WlxGetStatusMessage	Winlogon appelle cette fonction pour obtenir le status du message qui est affiché
WlxRemoveStatusMessage	Winlogon appelle cette fonction pour faire disparaître un message précédemment affiché
Network Provider	
WlxNetworkProviderLoad	Cette fonction est utilisée pour mettre les informations de l'utilisateur dans une structure qui va être passée aux *network providers*

13. Etat actuel du développement

La DLL dans l'état actuel permet d'authentifier un utilisateur sur une machine Windows 2000. Les fonctionnalités suivantes n'ont pas été implémentées :

- La DLL ne permet l'authentification que sur le domaine local de l'ordinateur. Pour pouvoir s'authentifier sur un autre domaine, il faut énumérer les domaines disponibles avec une API de Winlogon et les afficher à l'utilisateur dans la boîte de dialogue créée dans *WlxLoggedOutSAS*. Il faut ensuite passer le nom de domaine à la fonction LogonUser.

- Diverses options sont paramétrables dans Winlogon par une modification des flags dans la base de registre. Une DLL GINA de remplacement se doit de tenir compte de ces options. Le document 'The Essentials of Replacing MSGINA.DLL' vous apportera plus de détails sur ces différentes options. Le lien est disponible dans les ressources utiles au point IV.15.

De plus, un exemple d'implémentation matérielle est fournit avec UsbGina, annexe 8. Il serait intéressant de réutiliser cette base pour intégrer la DLL avec un périphérique d'authentification existant.

14. Perspectives futures

La DLL GINA dans l'état actuel est une bonne base pour une entreprise désirant intégrer son périphérique d'authentification dans un système Windows 2000.

D'un point de vue didactique, il serait intéressant de continuer le développement pour créer une PKI sur la base du *Smart Card Logon*. Pour cela, il faudrait approfondir les connaissances dans *Certificate Services* pour accéder à une authorité de certification depuis GINA et authentifier un utilisateur grâce à un certificat stocké dans un jeton d'accès comme une carte à puce.

Un autre travail intéressant pourrait être d'utiliser le SDK (*Software Development Kit*) offert par Aladdin pour interfacer un eToken avec cette DLL GINA.

15. Pré-requis pour continuer le développement

Pour continuer ce développement, il est préférable posséder :

- De bonnes connaissances du langage C++
- De bonnes connaissances de l'environnement Visual C++ et des MFC
- Une bonne compréhension du code source et de ce chapitre

La configuration nécessaire au développement est décrite au point IV.4. Les programmes suivants doivent être installés sur la machine de développement :

- Microsoft Visual C++ 6.0
- MSDN Library
- Platform SDK (voir annexe 1 pour installation et liens)

La machine de test doit être suffisamment performante pour démarrer rapidement rapidement car les redémarrages sont fréquents lors d'un tel développement. Une installation minimale de Windows est aussi conseillée pour cela.

16. Ressources utiles

Très peu de ressources sont disponibles sur Internet pour aider le développement de DLL GINA. Lors du début du travail de diplôme, un groupe Yahoo dédié à GINA a été crée en collaboration avec Paul Evans. Ce dernier est chargé de développer une DLL GINA sur mesure pour la société DS (www.ds.co.uk). Ce groupe Yahoo ne contient actuellement pas beaucoup d'informations mais grandit rapidement et pourrait devenir un jour un intéressant site de référence pour le développement GINA.

Les ressources utilisées durant le travail de diplôme sont les suivantes :

- **Platform SDK August 2001**
  Contient les documents de référence sur GINA. Tous les points d'entrée y sont définis avec une description détaillée. Fournit dans le MSDN Universal.
- **The Essentials of Replacing MSGINA.DLL**
  http://www.microsoft.com/windows2000/docs/msgina.doc
  Le seul document de Microsoft décrivant le développement d'une DLL GINA. Toutes les options de la base de registre y sont détaillées.

- **GINA Yahoo Group**
  http://groups.yahoo.com/group/msgina
  Divers documents et parties de code. Diverses fonctions non-documentée y sont expliquées. Encore un peu jeune mais tend à se développer.

- **Microsoft Security Newsgroup**
  http://msdn.microsoft.com/newsgroups/loadframes.asp?newsgroup=microsoft.public.platformsdk.security
  Le seul newsgroup officiel où l'on peut poser des questions sur GINA et sur CryptoAPI.

- **NISGINA Source Code**
  http://nisgina.deakin.edu.au
  Développement d'une DLL GINA pour s'authentifier sur un domaine NIS par l'université de Deakin, très intéressant.

17. Conclusion

Le développement d'une DLL GINA est rendu complexe à cause des points suivants :

- GINA est une DLL chargée au démarrage de Windows 2000 ce qui implique un redémarrage inévitable pour le test d'une nouvelle version. Au début du développement, la DLL n'est pas encore capable d'authentifier un utilisateur dans le système ce qui implique encore un redémarrage supplémentaire. Cela diminue beaucoup le rendement lors du développement initial.

- La documentation fournie par le Platform SDK est parfois trop succincte et ne suffit pas toujours à elle toute seule. D'autres ressources sont alors nécessaires pour avancer dans le développement (voir point IV.15).

La réalisation de cette DLL a été grandement facilitée par le choix de développer un composant GINA qui ne supporte que les fonctionnalités minimales pour authentifier un utilisateur sur le domaine local. Elle donne satisfaction avec les fonctionnalités précitées.

V. Développement d'un CSP CryptoAPI

1. Objectif

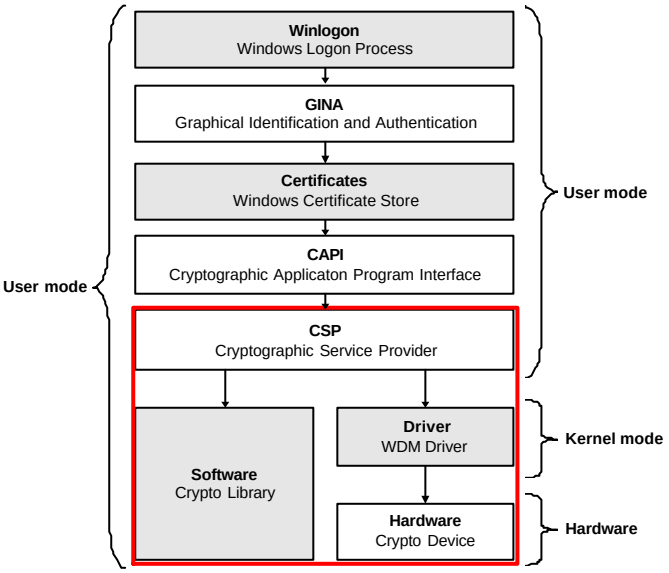
Le premier objectif de cette étape est de se familiariser avec le développement d'un CSP. Pour cela, une DLL permettant d'obtenir une trace de n'importe quel CSP installé dans le système sera développé.

L'objectif suivant est de développer un CSP permettant de générer un certificat. Voici les fonctionnalités que doit posséder le CSP pour cet objectif :

- Gestion des conteneurs de clés
- Génération de clés asymétriques de type RSA
- Exportation d'une clé publique
- Hachage de données
- Signature d'un hash avec la clé privée

Les opérations cryptographiques seront implémentées en logiciel grâce à OpenSSL. Ce dernier est un outil permettant d'effectuer de nombreuses opérations relatives à la sécurité informatique, allant de la génération de clés cryptographiques jusqu'à la gestion de certificats numériques.

2. Vue d'ensemble



3. Environnement de test

Le développement d'un CSP ne requière pas de configuration de test complexe. L'écriture du code et le déverminage peuvent être effectués sur une même machine.

Deux options sont possibles pour tester le fonctionnement d'un CSP :

- Le système d'exploitation utilise un CSP lors de la création d'un certificat. Il est alors possible de l'utiliser pour vérifier que le CSP fonctionne
- L'autre solution est d'écrire une application spécifique pour effectuer les tests

Cette dernière option est la plus appropriée pour commencer le développement car il est ainsi possible de tester les fonctions implémentées étape par étape. Lorsque le développement du CSP est dans un stade plus avancé, il est alors possible de vérifier la compatibilité avec le système d'exploitation ou d'autres logiciels spécifiques comme Microsoft Outlook.

4. Authenticité d'un CSP

Les CSP sont des composants qui jouent un rôle important sur toute la sécurité du système d'exploitation. Il est donc nécessaire de mettre en place des mécanismes qui empêchent qu'une personne malintentionnée puisse remplacer un CSP système par un CSP malicieux. Pour cela, chaque CSP doit posséder une signature numérique directement générée par Microsoft.

Avant Windows 2000, les signatures des CSP étaient stockées dans la base de registre. Il arrivait parfois que la DLL et la signature correspondante se désynchronisent rendant le CSP inutilisable. Ce problème a été fixé dans Windows 2000 en stockant directement la signature dans les ressources de la DLL.

Lorsqu'un CSP est demandé par une application avec la fonction `CryptAcquireContext`, CryptoAPI va vérifier sa signature. Si ce test échoue, la fonction retournera l'erreur `NTE_BAD_SIGNATURE` et il sera impossible de l'utiliser.

5. Installation du CSP Developer's Kit

Lors de la phase de développement, il faut bien entendu pouvoir outrepasser cette protection. Microsoft fournit pour cela le CSPDK qui contient une DLL **advapi32.dll** de remplacement. C'est elle qui est chargée de la vérification de l'intégrité d'un CSP.

Pour télécharger le CSPDK :
http://download.microsoft.com/download/win2000pro/Utility/V2.0/W98NT42KMe/EN-US/cspdk.exe

Les fichiers utiles du CSPDK sont les suivants :

- **cspSign.exe** permet de signer un CSP durant le développement
- **advapi32.dll** autorise le système à accepter un CSP signé avec cspSign
- **cspdk.h** contient toutes les déclarations nécessaires à l'écriture d'un CSP

Pour installer le fichier advapi32.dll sous Windows 2000, procédez comme suit :

- Choisissez le bon fichier **advapi32.dl_** correspondant à votre service pack
- Exécutez la commande **expand –r advapi32.dl_ advapi32.dll**
- Redémarrez en mode console (F8 au démarrage, puis *Recovery Console*)
- Renommez le fichier **c:\winnt\system32\advapi32.dll** en **.old**
- Copiez le nouveau advapi32.dll dans le répertoire **c:\winnt\system32**
- Redémarrez

Le système est maintenant prêt pour commencer le développement d'un CSP.

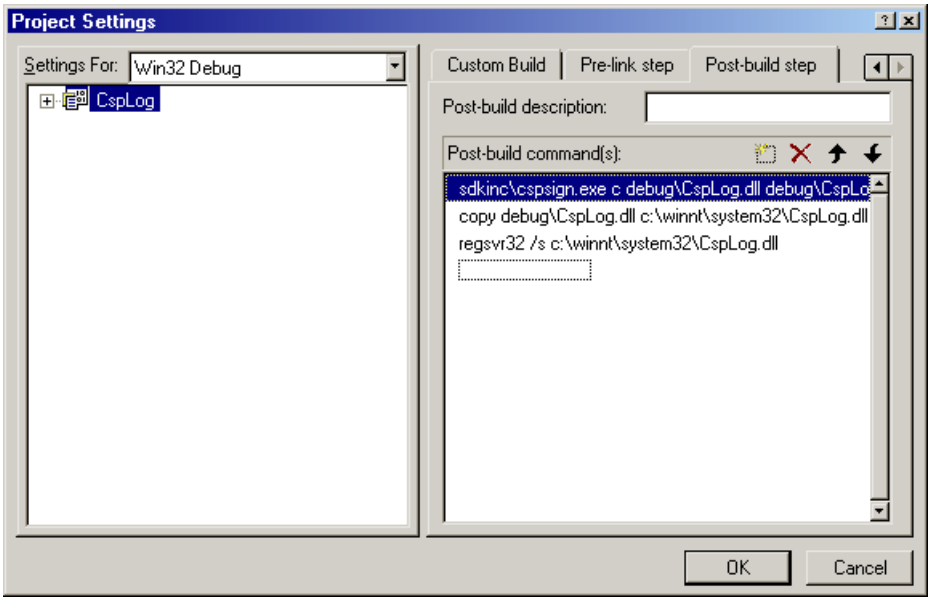
6. Création d'un projet CSP avec Visual C++

Vous trouverez le détail la création d'un projet DLL MFC avec l'environnement Visual C++ dans l'annexe 2.

Pour faciliter le développement, il est judicieux de laisser le soin à Visual C++ d'effectuer certaines opérations. Les opérations suivantes sont nécessaires à chaque compilation :

- Signature du CSP (voir point V.4)
- Copie de la DLL dans le répertoire 'c:\winnt\system32'
- Enregistrement de la DLL (voir point V.9)

Pour que Visual C++ exécute ces opérations à la fin de chaque compilation, il faut aller dans le menu 'Project > Settings > Post-build steps'.



Avant de lancer la compilation, il est nécessaire qu'aucun logiciel n'accède au CSP en cours de développement sans quoi l'opération de copie va échouer.

7. Conteneurs de clés cryptographiques

Deux types de clés existent dans un CSP :

- Les **clés de sessions** sont des clés symétriques qui sont utilisées pour chiffrer une grande quantité de données. Ces clés sont **volatiles**, cela signifie qu'elles ne sont pas stockées entre deux sessions.

- Les **clés d'échange et de signature** sont des paires de clés asymétriques de type RSA. Ces clés sont **non-volatiles**, elles sont résidentes dans un CSP et seront présentes entre diverses sessions.

Un conteneur de clés est une sorte de répertoire où les paires de clés asymétriques sont stockées. Il existe deux types de clés RSA différentes :

- **AT_KEYEXCHANGE**
  Cette paire de clé va être utilisée pour chiffrer et échanger des clés de session. C'est aussi cette paire qui va être exportée lors de la création d'un certificat.

- **AT_SIGNATURE**
  Cette paire est utilisée pour créer et vérifier des signatures numériques.

Le *handle* de ces clés peut être récupéré via la fonction `CPGetUserKey`.

`CPAcquireContext` est le premier point d'entrée appelé par une application. Mis à part son utilisation normale qui consiste à obtenir un *provider*, il peut aussi servir pour gérer des conteneurs de clés en spécifiant les *flags* adéquats.

Un utilisateur peut posséder plusieurs conteneurs de clés. Le paramètre `pszContainer` de `CPAcquireContext` permet de les différencier. Si ce paramètre est `NULL`, le conteneur par défaut de l'utilisateur est utilisé.

Lorsque l'on désire créer un nouveau conteneur, il faut utiliser le flag `CRYPT_NEWKEYSET` avec un `pszContainer` non existant. C'est ce qui se passe lors de la création d'un nouveau certificat par exemple. Il faut ensuite demander la création des paires de clés avec la fonction `CryptGenKey`.

Il est aussi possible de supprimer un conteneur grâce au flag `CRYPT_DELETEKEYSET`.

8. Types d'implémentations d'un CSP

Comme indiqué au point II.3, les fonctions cryptographiques d'un CSP peuvent être implémentées en logiciel, en matériel ou parfois mixtes. Il est possible d'obtenir le type d'implémentation d'un CSP grâce à la fonction `CryptGetProvParam` avec le paramètre `PP_IMPTYPE`.

Les types d'implémentation définis sont les suivants :

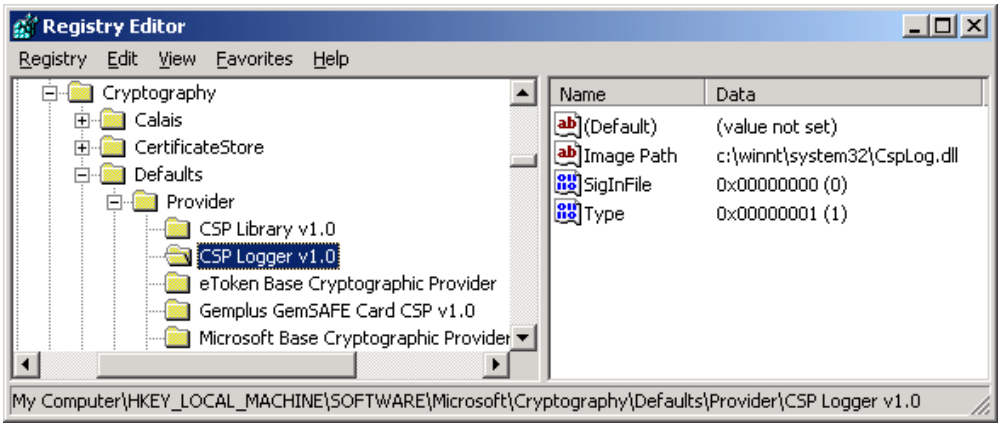
- `CRYPT_IMPL_HARDWARE`
- `CRYPT_IMPL_SOFTWARE`
- `CRYPT_IMPL_MIXED`
- `CRYPT_IMPL_UNKNOWN`

L'implémentation *Mixed* est souvent utilisée lorsque les périphériques doivent être très bon marché. Dans ce cas, les clés sont générées en logiciel et sont stockées dans une mémoire non-volatile du module cryptographique. Un bon exemple est le eToken R2 d'Aladdin. Vous trouverez plus d'informations à ce sujet dans le chapitre III.2.

9. Installation d'un CSP dans le système d'exploitation

Un CSP est installé dans le système d'exploitation en créant une clé dans la base de registre. Le chemin de la clé à créer est le suivant :
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\Defaults\Provider

La copie d'écran suivante représente le programme *Registry Editor* (regedit.exe). Ce programme permet de manipuler aisément la base de registre.



Le nom de la clé correspond au nom du CSP. Il sera ensuite utilisé pour accéder directement au CSP avec la fonction `CPCryptAcquireContext`, paramètre `pszProvider`. Les valeurs suivantes doivent être spécifiées dans la clé :

- **Image Path**
  Cette valeur spécifie l'emplacement de la DLL du CSP.

- **SigInFile**
  Cette valeur indique au système d'exploitation que la signature du CSP est stockée dans les ressources de la DLL. Cette notion a été introduite avec Windows 2000 pour éviter le problème cité au point V.4.

- **Type**
  Plusieurs types de CSP sont définit dans le fichier 'wincrypt.h'. On trouve entre-autre le type `PROV_RSA_FULL` qui indique que le CSP supporte toutes les fonctions cryptographiques. Il est possible de demander au système d'énumérer tous les CSP d'un certain type pour effectuer une tâche bien particulière.

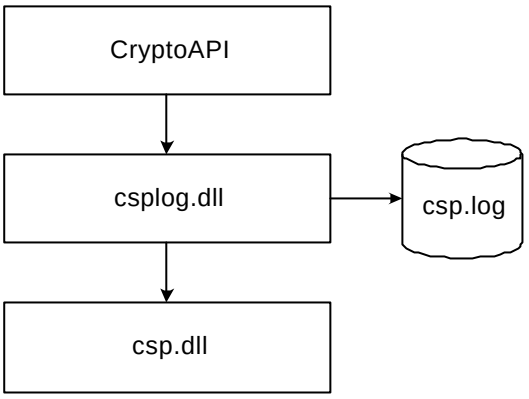
Cette procédure d'enregistrement dans la base de registre est très importante et il est préférable qu'elle soit gérée par le CSP lui-même. Le programme regsvr32.exe permet d'exécuter des points d'entrée d'une DLL à cet effet. Vous trouverez plus de détails sur ce sujet dans l'annexe 3.

10. Création d'un CSP de traçage

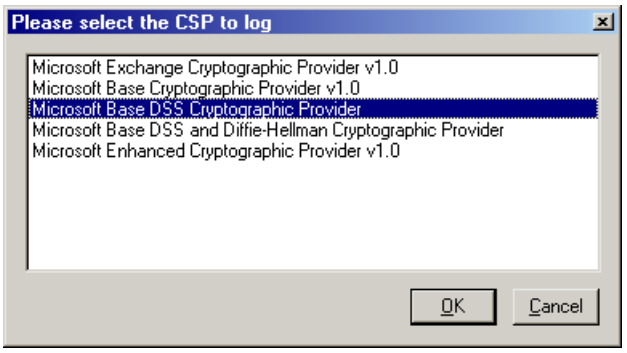
La quantité de points d'entrée d'un CSP et leur complexité rend impossible le développement d'un CSP complet dans le cadre d'un travail de diplôme.

Le but du CSP développé est d'être capable de générer une paire de clé asymétrique qui sera ensuite associée à un certificat. L'idéal est de n'implémenter que les points d'entrée nécessaires à cette opération. Pour cela, un CSP de traçage a été développé.

Le principe est très simple : pour chaque point d'entrée du CSP, une trace dans un fichier log est effectuée et le point d'entrée correspondant d'un autre CSP est appelé.



Lors du premier point d'entrée qu'est `CPAcquireContext`, le CSP de traçage va afficher une boîte de dialogue à l'utilisateur l'invitant à sélectionner un autre CSP dont il aimerait obtenir une trace. La boîte de dialogue doit énumérer tous les CSP disponibles dans le système d'exploitation avec la fonction `CryptEnumProviders` (voir `CSPLOG3` dans code source, annexe 9).

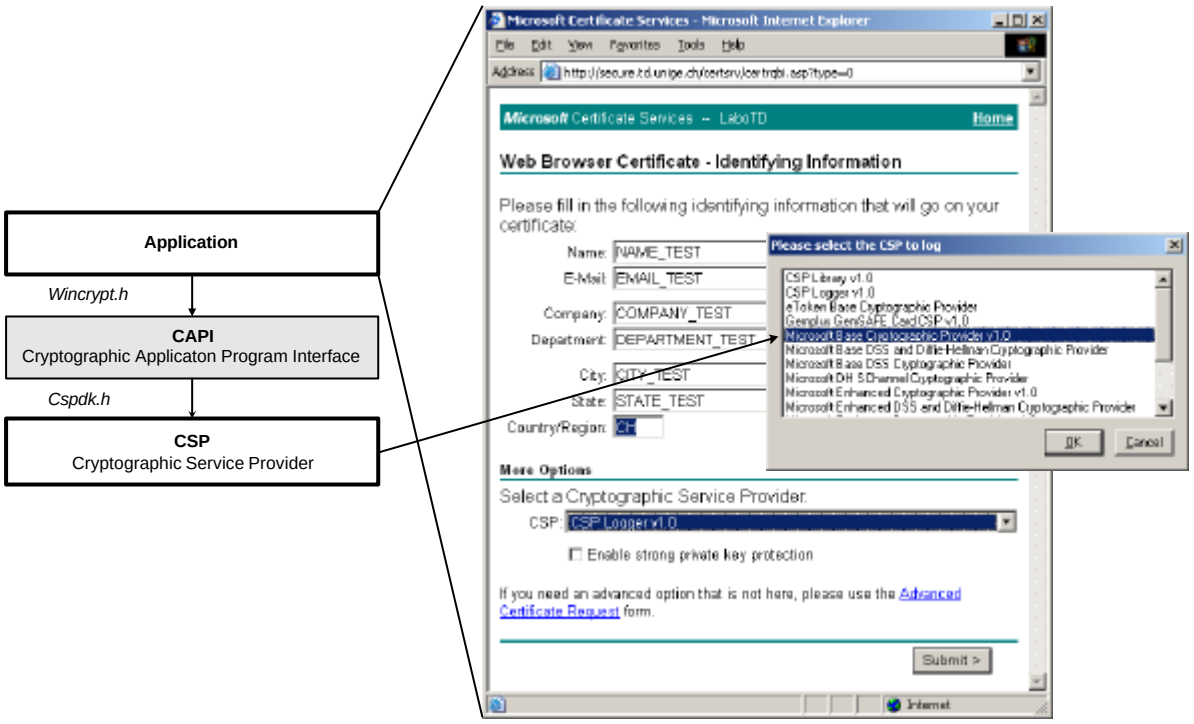


Lorsque l'utilisateur a sélectionné un CSP, la boîte de dialogue récupère le nom de la DLL associée dans la base de registre (voir `CSPLOG4`).

Ensuite, la DLL correspondante est chargée (voir `CSPLOG1`) et les pointeurs sur chaque point d'entrée sont récupérés (voir `CSPLOG2`). Vous trouverez le code source du projet CspLog dans l'annexe 9.

11. Trace obtenue lors de la création d'un certificat

L'autorité de certification du laboratoire de transmission de données a été utilisée pour effectuer la requête d'un certificat. Elle peut être accédée depuis une interface Web à l'adresse 'http://secure.td.unige.ch/certsrv'.



Pour créer un certificat, les tâches suivantes sur le CSP sont effectuées :

- Création d'un nouveau conteneur de clé
- Création de la clé d'échange
- Export de la clé publique d'échange
- Hachage du certificat
- Signature du hash avec la clé privée d'échange

Il est donc nécessaire de pouvoir générer une paire de clés asymétriques ainsi que de supporter un algorithme de hachage pour générer la signature du certificat. Vous trouverez plus d'informations sur le sujet au point V.12.

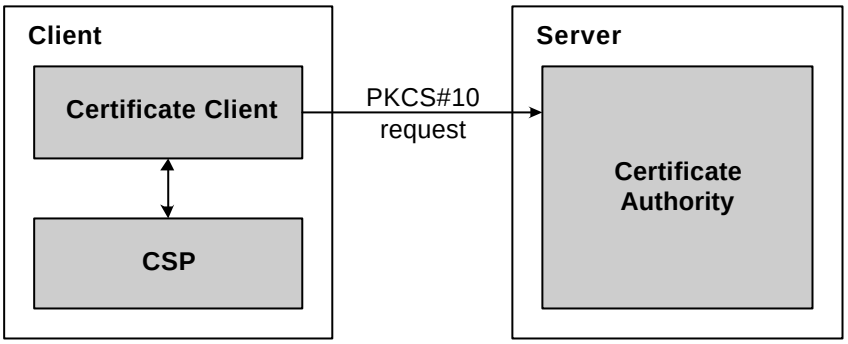
Une librairie cryptographique est la solution la plus simple et la plus rapide pour implémenter ces fonctions en logiciel. Les points V.15 et V.16 vous donneront plus de détails sur le sujet.

La trace obtenue lors de la création d'un certificat est disponible dans l'annexe 4.

12. Les entrailles de la création d'un certificat

12.1 Introduction

Lors de la requête d'un certificat, le client obtient les informations utilisateurs, effectue des opérations via un CSP et envoie une requête de certificat PKCS#10 au serveur de certificat.



Dans le cas d'Internet Explorer, le client est un contrôle ActiveX.

12.2 Requêtes de certificat du côté CSP

Les points d'entrées suivants sont appelés lors de la création d'un certificat :

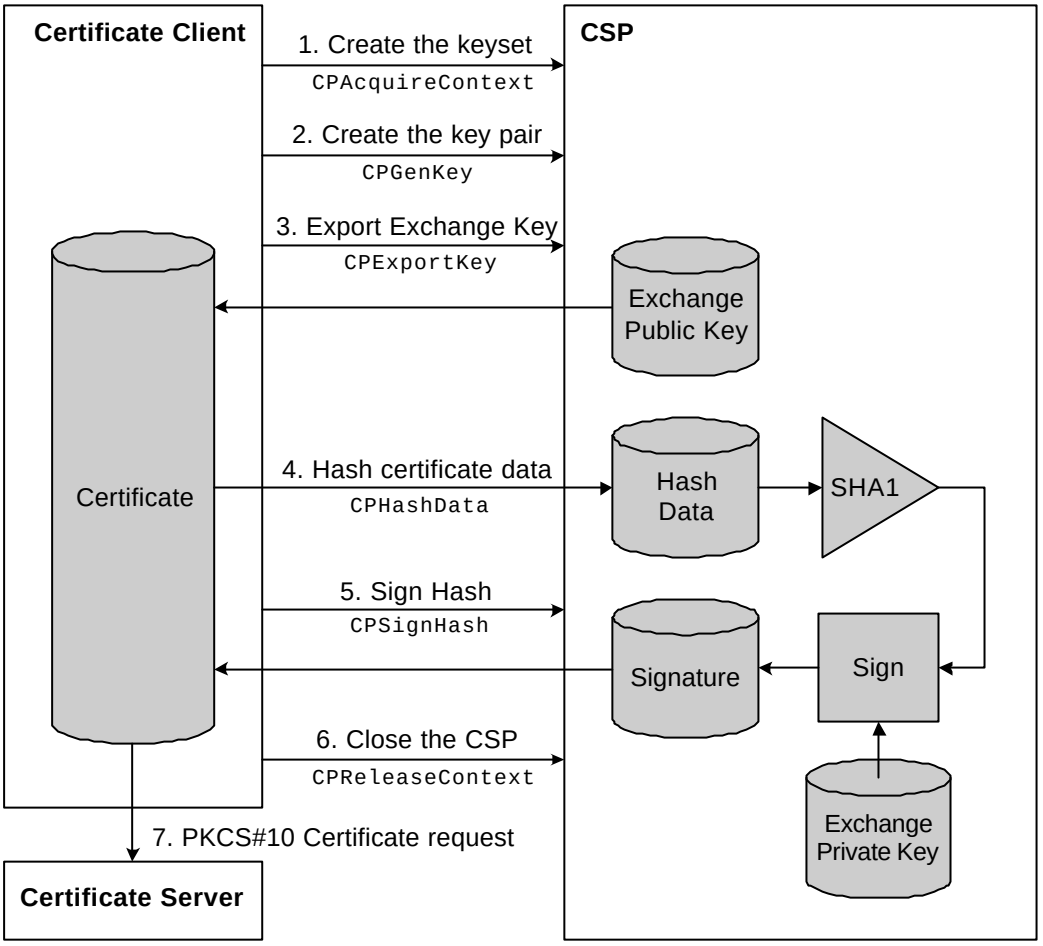
- `AcquireContext et ReleaseContext`
- `GetProvParam avec PP_NAME, PP_PROVTYPE, PP_ENUMALGS`
- `GetUserKey avec AT_KEYEXCHANGE`
- `ExportKey`
- `DestroyKey`
- `CreateHash et DestroyHash`
- `HashData`
- `SignHash`

Des données sont échangées sous différents formats dans les points d'entrée `ExportKey`, `HashData` et `SignHash`. Vous trouverez plus de détails sur ces formats au point V.13.

Les données suivantes ont été sauvées pour être analysées :

- **Exchange.key** contient la clé publique qui va être intégrée dans le certificat,
- **Hash.dat** contient les données à hacher,
- **Hash.sig** contient la signature générée à partir de hash.dat.

Le schéma qui suit permet de mieux comprendre les différentes données inhérentes à chaque étape de la création :



13. Format des données

13.1 Format PEM et DER

DER est un format standard utilisé pour stocker des clés RSA et des certificats. PEM permet la transmission de données codées DER en format texte en les codant en Base64. Vous trouverez plus d'informations à ce sujet dans le diplôme 2001 de Cotte.

13.2 Format PUBLICKEYBLOB de CryptoAPI

CryptoAPI définit des formats d'exportation spécifiques pour les clés publiques et privées. Seul le format pour les clés publiques va être expliqué dans la suite de ce point, le format pour les clés privées étant très similaire. Vous trouverez un lien du Platform SDK expliquant ces formats dans l'annexe 1.

Lors de la génération d'un certificat, une clé publique a été sauvegardée grâce à CspLogger. Voici le fichier 'export.key' ouvert avec l'éditeur de Visual C++ en mode hexadécimal :

```
000000  06 02 00 00 00 A4 00 00  52 53 41 31 00 04 00 00   ........RSA1....
000010  01 00 01 00 5D 29 24 B8  48 07 BE 3E 3D 98 03 0A   ....])$.H..>=...
000020  F0 7A 52 59 0D 73 10 BC  7B CD 82 59 1A 82 D5 EC   .zRY.s..{..Y....
000030  E4 44 82 85 E6 E2 94 B8  35 70 24 C4 29 7D 32 A3   .D......5p$.)}2.
000040  46 08 D7 E1 46 3E 7A 92  C3 EC 4C DB DE 0D E2 62   F...F>z...L....b
000050  6B 88 59 18 E9 2F 0A AB  57 EC 2A BD F3 E1 A5 A5   k.Y../..W.*.....
000060  5B 0A 4C 35 88 BC F0 19  A7 18 FD 04 CD A4 5C 0B   [.L5..........\.
000070  50 79 E0 DE 88 96 69 AA  36 F1 95 0B 16 C9 E2 E9   Py....i.6.......
000080  64 A2 64 60 51 51 14 69  81 0D 93 A9 00 0F F9 D9   d.d`QQ.i........
000090  74 18 5C D0                                       t.\.
```

Il est possible de décoder le format comme suit :

- **bType** = 0x06 (PUBLICKEYBLOB)
- **bVersion** = 0x02 (CUR_BLOB_VERSION)
- **wReserved** = 0x0000 (Reserved)
- **aiKeyAlg** = 0x00A40000 (CALG_RSA_KEYX)
- **dwMagic** = 0x52534121 (RSA1)
- **dwBitLen** = 0x00040000 (1024 bits)
- **dwPubExp** = 0x01000100 (Public exponent)
- **pbModulus** (1024 bits of public key)

Le module de la clé privée (*modulus*) est inversé par rapport à celui présent dans le format DER. Le format CryptoAPI met l'octet du poids le plus faible en premier tandis que le format DER fait le contraire.

13.3 Format du hash

Après le hachage des données, la valeur du hash doit être signée avec la clé privée appairée à celle du certificat.

Avant de générer la signature, des octets doivent être ajoutés à la fin du hash. Ces octets ne sont malheureusement pas documentés dans le Platform SDK à ma connaissance.

Pour retrouver la valeur des octets à ajouter au hash, j'ai eut recours à une paire de clé dite **nulle**. Cette paire de clé possède la particularité d'avoir tous ses exposants de valeur 1. Si l'on chiffre une donnée avec une telle clé, le résultat du chiffrement sera **inchangé**.

Un article de Microsoft traite le sujet et propose du code pour générer de telles clés :
http://support.microsoft.com/support/kb/articles/Q228/7/86.ASP

14. Hiérarchie d'objets d'un CSP

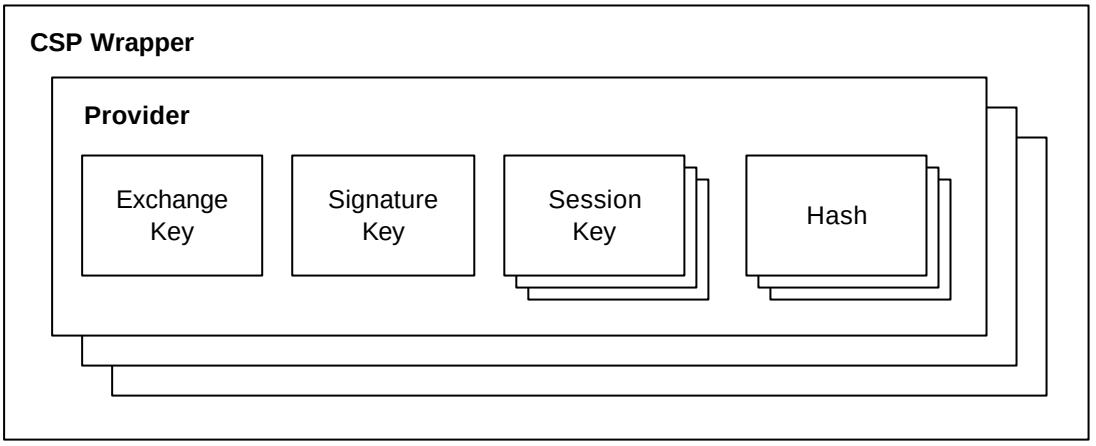
Avant le développement d'un CSP, il est préférable de définir une hiérarchie d'objets pour la suite du développement. Le développement va être effectué en langage C++, les objets définis dans ce chapitre vont donc correspondre à une classe C++. Vous trouverez le détail de la création d'une DLL avec Visual C++ dans l'annexe 2.

Le premier objet de la hiérarchie sera une classe qui possède une réplication complète des points d'entrée du CSP. La classe C++ sera nommée **CCspWrapper**.

Pour définir les autres objets composant ce CSP, il est indispensable de regrouper tous les points d'entrée par fonctionnalités. En analysant les points d'entrée, on remarque que les objets suivants ressortent clairement :

- Un fournisseur
- Les clés de session
- Les clés asymétriques d'échange de clés de sessions et de signature
- Les hashs

La figure suivante représente la hiérarchie créée avec le lien d'appartenance entre les divers objets:



Les classes suivantes ont été créées correspondant à ces objets:

- L'objet **CCspRsaKey** sera chargé de créer et de gérer les clés asymétriques d'échange et de signature.
- L'objet **CCspKey** effectuera toutes les opérations relatives aux clés de sessions comme la création, la suppression, l'import et l'export.
- L'objet **CCspHash** s'occupera de créer, de supprimer et de hacher des données.
- L'objet **CCspProvider** possédera une liste de clés et une liste de hashs. Il sera aussi chargé de générer des nombres aléatoires et de chiffrer des données.

Cette hiérarchie de classes a été développée et semble être bien adaptée. La gestion complète du CSP est implémentée. L'étape suivante consiste à ajouter les fonctions cryptographiques dans cette architecture.

15. Librairie cryptographique

15.1 Introduction

La première étape du travail consiste à implémenter les fonctions cryptographiques en logiciel grâce à une librairie cryptographique.

Pour cela, plusieurs librairies ont été évaluées pour finalement choisir OpenSSL. Vous trouverez plus de détails dans la suite du chapitre.

15.2 Crypto++

La recherche d'une librairie cryptographique puissante et du domaine public sur Internet nous amène presque naturellement vers Crypto++, une librairie entièrement développée en C++ par Wei Dai.

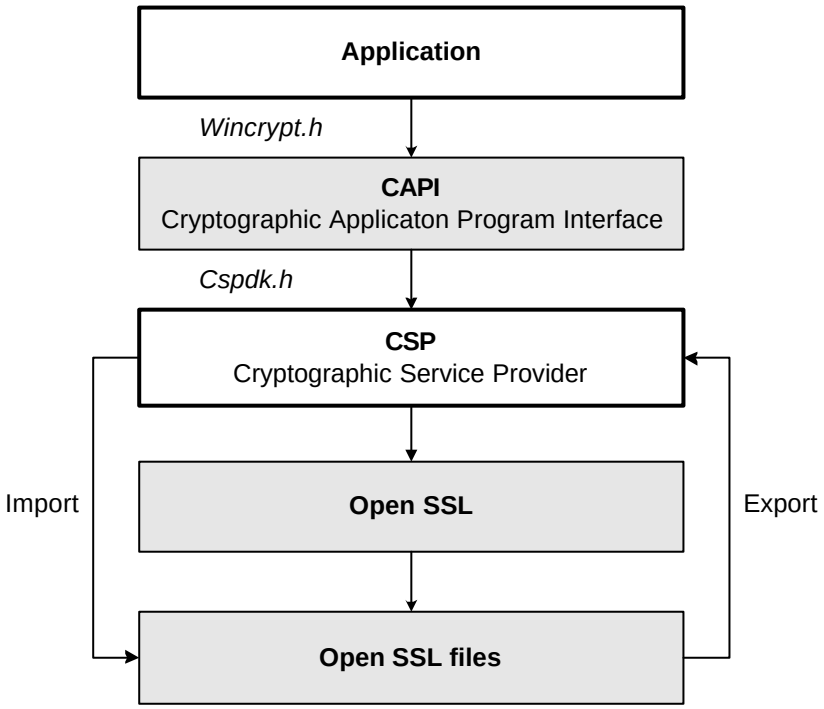
Malheureusement, j'ai remarqué que cette librairie comporte plusieurs bugs qui m'ont empêché d'aller plus loin dans le développement. Des changements de contextes apparaissaient lors du retour de fonctions utilisant la librairie. Ce genre de problèmes peut survenir lorsque les exceptions C++ sont mal gérées.

Une autre solution plus didactique a été trouvée en utilisant OpenSSL.

15.3 OpenSSL

OpenSSL est un outil permettant d'effectuer des opérations cryptographiques en mode 'ligne de commande'. Vous trouverez plus de détails sur cet outil dans le mémoire de diplôme 2001 de Cotte.

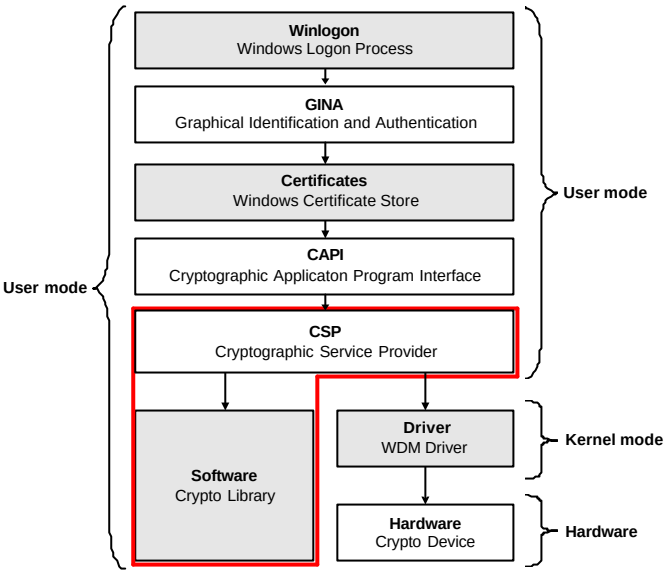
En plus d'être très performant, OpenSSL est un outil très didactique. Tous les fichiers générés sont disponibles dans un format standard. L'idée est d'utiliser OpenSSL en ligne de commande depuis le CSP pour générer les fichiers nécessaires et d'écrire des fonctions pour l'importation et l'exportation de ces fichiers.



L'utilisateur du CSP aura ainsi accès à tous les fichiers générés par OpenSSL durant par exemple la création d'un certificat avec un fichier log contenant toutes les commandes effectuées sur les divers fichiers. Cette méthode à l'avantage d'être très didactique ainsi que simple à mettre en œuvre. Vous trouverez les commandes utilisées par le CSP dans l'annexe 11.

16. Implémentation des points d'entrée d'un CSP en logiciel avec OpenSSL

16.1 Introduction



L'implémentation de l'interface entre le CSP et OpenSSL est effectuée dans la classe COpenSSLWrapper. Cette classe gère un fichier log contenant toutes les commandes effectuées sur OpenSSL tels que :

- Génération de paires de clés RSA
- Extraction de la clé publique d'une paire de clé
- Hachage de données
- Génération d'une signature numérique
- Vérification d'une signature numérique
- Génération de nombres aléatoires

La configuration du CSP telle que les algorithmes supportés et les longueurs des clés est effectuée dans le fichier nommé 'config.h' (voir `CSPLIB1` dans code source, annexe 10).

16.2 Création d'un keyset

L'ouverture d'un CSP lors de la création d'un certificat est effectuée avec le paramètre `CRYPT_NEWKEYSET` qui signifie qu'un nouveau conteneur de clés (voir point V.7). Un répertoire situé dans l'environnement de l'utilisateur va être utilisé pour stocker tous les fichiers utilisés par le CSP (voir `CSPLIB2`).

16.3 Exportation de la clé publique

Lors de l'exportation d'une clé publique dans le point d'entrée `CPExportKey`, il faut convertir le format utilisé par OpenSSL dans celui de CryptoAPI. OpenSSL utilise le format standard DER (voir point V.13.1) pour stocker les clés, il est donc nécessaire de le décoder pour le passer dans le format `PUBLICKEYBLOB` de CryptoAPI (voir point V.13.2). Vous trouverez le code d'implémentation dans l'annexe 10 à l'endroit `CSPLIB3`.

Comme indiqué dans le point V.13.2, les octets ne sont pas placés dans le même ordre entre les deux formats. La fonction `SwapBytes` sera utilisée pour intervertir l'ordre des octets lors du passage d'un format à l'autre (voir `CSPLIB4`).

16.4 Génération de la signature

Comme expliqué au point V.13.3, des octets doivent être ajoutés à la valeur du hash avant de la signer (voir CSPLIB5). La valeur de ces octets ne change pas en fonction de l'algorithme de hachage utilisé, ni de la longueur de la clé.

Comme pour le module de la clé publique, les octets de la signature doivent être inversés lors du passage de OpenSSL à CryptoAPI (voir CSPLIB6).

17. Points d'entrée d'un CSP

Les points d'entrée d'un CSP permettent d'effectuer les opérations cryptographiques les plus communément utilisées telles que :

- Génération de nombres aléatoires
- Chiffrement/Déchiffrement de données
- Gestion des clés cryptographiques
- Génération du hash d'un flux de données
- Signature et vérification d'un flux de données

CryptoAPI définit divers types de CSP. Il est ainsi possible de créer un CSP qui ne permet que de générer des nombres aléatoires sans avoir besoin de d'implémenter les points d'entrée non utilisés. Lorsqu'un point d'entrée n'est pas supporté pour un type de CSP donné, il doit retourner l'erreur E_NOTIMPL.

Vous trouverez la liste complète des divers types de CSP dans le fichier *wincrypt.h* en effectuant une recherche de PROV_RSA_FULL. Malheureusement, il n'existe pas à ma connaissance un document qui décrit les points d'entrée à implémenter en fonction du type de CSP choisit. Vous trouverez par contre une description des CSP de base livrés avec le système d'exploitation dans le Platform SDK. Le lien correspondant est présent dans l'annexe 1.

La liste suivante décrit tous les points d'entrée d'un CSP :

Points d'entrée	Description
Service Provider Functions	
CPAcquireContext	Permet d'obtenir un *handle* sur le CSP spécifié
CPReleaseContext	Libère un contexte obtenu avec CPAcquireContext
CPGetProvParam	Permet d'obtenir les paramètres relatifs à un CSP
CPSetProvParam	Permet de modifier les paramètres relatifs à un CSP
Random Generation Functions	
CPGenRandom	Génère une suite de nombres aléatoires
Data Encryption Functions	
CPEncrypt	Permet de chiffrer un tableau d'octets
CPDecrypt	Permet de déchiffrer un tableau d'octets
Key Generation and Exchange Functions	
CPGenKey	Génère une clé cryptographique
CPDeriveKey	Génère une clé de session dérivée à partir de données de base. Garantit aussi que chaque clé

Points d'entrée	Description
	générée à partir de la même base et avec le même algorithme sera identique
CPDestroyKey	Libère un handle de clé
CPExportKey	Permet d'exporter une clé cryptographique en dehors du CSP d'une manière sûre
CPGetKeyParam	Permet d'obtenir les paramètres relatifs à une clé cryptographique
CPSetKeyParam	Permet de modifier les paramètres relatifs à une clé
CPGetUserKey	Permet d'obtenir le handle d'une clé permanante du CSP, par exemple la clé d'échange ou la clé de signature
CPImportKey	Transfert une clé cryptographique d'un *keyblob* dans le CSP
Hashing and Digital Signature Functions	
CPCreateHash	Permet d'obtenir un handle qui sera utilisé pour effectuer un *hash* sur un flux de données
CPDestroyHash	Détruit le handle précédemment créé avec CPCreateHash
CPGetHashParam	Permet d'obtenir les paramètres relatifs à un objet hash. La valeur du hash est aussi obtenue avec cette fonction
CPSetHashParam	Permet de modifier les paramètres relatifs à un objet hash
CPHashData	Hash les données spécifiées dans l'objet spécifié par le handle
CPHashSessionKey	Hash une clé cryptographique dans l'objet spécifié par le handle
CPSignHash	Permet de signer un objet hash
CPVerifySignature	Permet de vérifier la signature d'un objet hash

18. Etat actuel du développement

Le projet CSPLIB implémentant les opérations cryptographiques grâce à OpenSSL remplit le cahier des charges. Pour rappel, les opérations suivantes sont supportées par ce CSP :

- Gestion des conteneurs de clés
- Génération de clés asymétriques de type RSA
- Exportation d'une clé publique
- Hachage de données
- Signature d'un hash avec la clé privée

19. Perspectives futures

L'étape suivante de ce développement serait de modifier le CSP actuel pour l'interfacer avec un périphérique cryptographique.

D'un point de vue didactique, il serait intéressant de continuer le développement pour implémenter toutes les opérations cryptographiques avec OpenSSL. Beaucoup de travail est encore nécessaire pour arriver à ce résultat, mais la structure est déjà en place.

20. Pré-requis pour continuer le développement

Pour continuer ce développement, il est préférable posséder :

- De bonnes connaissances du langage C++
- De bonnes connaissances de l'environnement Visual C++ et des MFC
- De bonnes connaissances de la cryptographie
- Une bonne compréhension du code source et de ce chapitre
- Une bonne dose de patience

Les programmes suivants doivent être installés sur la machine de développement :

- Microsoft Visual C++ 6.0
- MSDN Library
- Platform SDK (voir annexe 1 pour installation et liens)
- CSP Developer's Kit (voir point V.5)

La marche à suivre qui a été utilisée pour remplir le cahier des charges (génération d'un certificat) dans ce développement a été d'effectuer une trace du CSP de base de Microsoft avec l'application finale (dans ce cas, le client de certificat) grâce à CspLog. Une fois que la trace obtenue est comprise, il faut implémenter les points d'entrée utilisés puis tester le CSP avec l'application jusqu'à ce que chaque étape soit réalisée avec succès.

21. Conclusion

CryptoAPI est une technologie relativement ancienne. L'architecture d'un CSP est basée sur un modèle non orienté objet. Pour cela, un gros travail est nécessaire pour mettre en place les charpentes qui sont nécessaires à la construction du composant en lui-même. Il aurait été souhaitable qu'un tel squelette soit fournit par Microsoft dans sont CSP Developer's Kit. Aucun exemple n'est non plus disponible.

Le travail aurait pu être simplifié si CryptoAPI était basé sur un modèle orienté objet. Pour outrepasser cet obstacle, Microsoft à développé une interface COM (Component Object Model) pour CryptoAPI nommée CAPICOM. Cette interface simplifie le développement d'applications cryptographiques, mais malheureusement pas le développement d'un CSP.

VI. Conclusion

1. Synthèse du travail

1.1 Partie GINA

La DLL GINA fonctionne. Elle permet d'authentifier un utilisateur sur le domaine local d'un ordinateur avec son nom et son mot de passe. La DLL développée a ensuite été modifiée pour utiliser une carte de développement USB comme jeton d'accès. Cette DLL pourrait facilement être reprise pour être interfacée avec un système d'authentification évolué.

La difficulté majeure du développement d'une DLL GINA est l'environnement de test. Beaucoup de temps est perdu dans les nombreux redémarrages nécessaires au début du développement.

1.2 Partie CSP

Le CSP permet de générer un certificat. Les opérations cryptographiques telles que la génération des clés asymétriques de type RSA et la génération de signatures numériques ont été implémentées en logiciel grâce à la librairie OpenSSL. Tous les fichiers utilisés pour la génération d'un certificat sont accessibles à l'utilisateur ce qui rend ce CSP un outil très didactique.

Le développement d'un CSP est un travail lourd et difficile. Beaucoup de code est nécessaire à la mise en place du squelette qui permettra par la suite d'ajouter les fonctionnalités désirées. Aucun exemple de CSP n'est disponible dans le Platform SDK ni sur Internet. Un autre point qui a rendu le développement difficile est la pauvreté de la documentation du Platform SDK sur certains points.

2. Améliorations à apporter

La DLL GINA n'implémente pas certains détails qui servent uniquement à améliorer l'expérience de l'utilisateur. Dans une version plus évoluée de cette DLL, il serait bien entendu souhaitable d'ajouter ces fonctionnalités.

Le CSP pourrait être complété pour offrire une implémentation complète de toutes les opérations cryptographiques. Le squelette de base pourrait aussi être adapté pour une implémentation matérielle.

3. Mot de la fin

Dans une époque où la sécurité informatique et l'identification personnelle sont de mise, Winlogon et CryptoAPI offrent à Windows toute la puissance et la souplesse nécessaire pour évoluer avec les avancées technologiques.

Mario Pasquali, le 4 décembre 2001