



Travaux pratiques

JAVA CARD

TP n°1 – Authentification : Utilisation de la cryptographie et du multi-canal

TP n°2 – Authentification mutuelle inter-applets

TP n°3 – Utilisation du PIN Global de GP

TP n°4 – Utilisation des Secure Channels fournis par GP

TP n°5 – Personnalisation d'une application via une SD et changement d'état via les API GP

TP n°6 – Création d'un Security Domain, personnalisation de ses clés et installation d'une application dans ce SD.

Nous utiliserons dans cette série de TP l'environnement Eclipse avec le plugin JCOP déjà activé.

Je rappelle le fondement de Java, il n'y a pas d'arithmétique sur les types autres que numériques (et en Java Card sur les entiers). Pensez aux règles de base de génie logiciel, pas de constantes en dur dans le code, nommer correctement vos variables, etc.

Pour tous les TP, nous utiliserons des JCOP.

Pour charger dans la carte, on utilisera suivant les TP et les besoins soit Eclipse directement soit GPShell, soit un programme Java faisant office de terminal. Ce dernier utilise la librairie Opal qui est une implémentation de GP. Vous pouvez charger, télécharger et exécuter des commandes depuis un monde Java.

Procédure de création d'une application :

Donner un nom à votre projet, sélectionner les profils de votre carte (en général JC2.2.1 et GP2.1.1), à votre package et à votre applet. Donner également un AID à votre package et à l'applet (en le dérivant de celui du package).

Écrire le programme Java correspondant à la spécification. Pour ce faire, créer un projet Java Card (un projet par package). Le nom du projet et/ou des packages et applets n'ont pas de réelle importance c'est plus pour vous y retrouver.

Une fois l'application codée, vous la chargez sur la carte au travers du Run Configuration. Double cliquez sur Java Card Application et sélectionner le lecteur de carte comme destination et en avant « Run ».

À l'aide du shell de JCOP vous pouvez échanger avec la carte via des commandes prédéfinies ou directement vos APDU. Vérifiez le bon fonctionnement de votre application.

Si vous rencontrez des erreurs n'hésitez pas à me les signaler :

damien.sauveron@unilim.fr

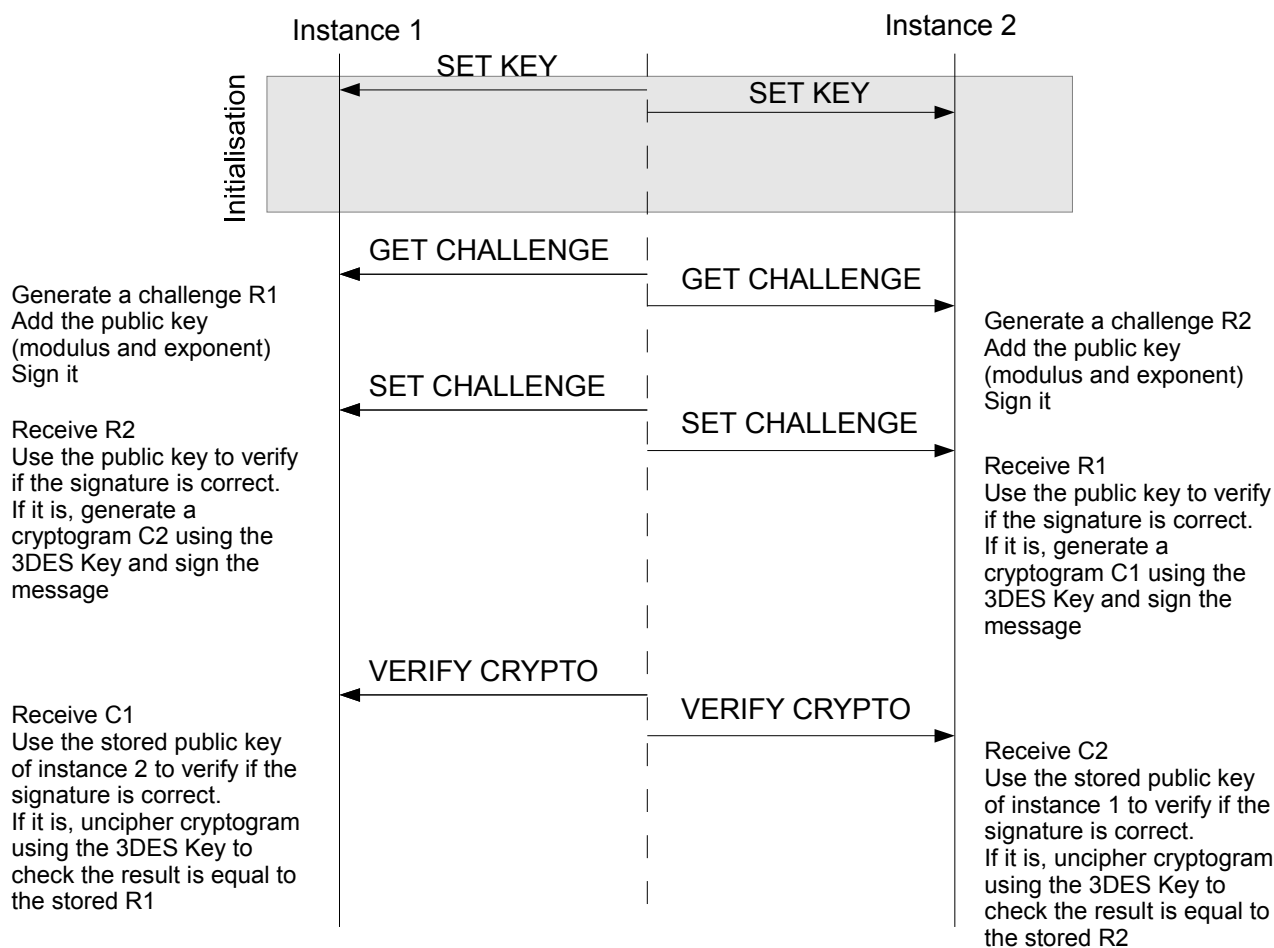
TP n°1 – Authentification mutuelle : utilisation de la cryptographie et du multi-canal

Objectif : Manipulation des primitives de cryptographie et manipulation du multi-canal
L'utilisation de plusieurs instances de la même application en multi-canal nécessite certaines modifications d'une applet classique

Dans ce TP nous allons utiliser les primitives cryptographiques pour réaliser une authentification mutuelle. Pour cela nous implémenterons un protocole *challenge/response* dans lequel au départ les deux parties possèdent une clé symétrique 3 DES de 192 bits qui sera utilisé pour chiffrer les challenges (de 8 octets) en cryptogramme et inversement en utilisant l'algorithme ALG_DES_CBC_NOPAD.

Chaque partie disposera d'un générateur de nombres aléatoires et d'une paire de clés RSA de 512 bits (auto-généré au format CRT) qui sera utilisée pour signer et vérifier les messages avec l'algorithme ALG_RSA_SHA_PKCS1.

Les deux parties seront simulés par 2 instances de la même application.



Utiliser des constantes pour les différentes instructions (respectivement dans l'ordre du schéma 0x10, 0x20, 0x30 et 0x40).

Allez y progressivement : création des différents objets, RNG, RNG+clé pub, RNG+clé pub+signature, ...

À l'issue de l'authentification mutuelle on pourrait poursuivre les échanges en chiffrant avec la clé 3DES et en signant. On pourrait dériver des clés de sessions, etc.

Idéalement si nos cartes supportent les canaux logiques, chaque instance utilisera un canal logique différent.

Dans ce contexte, il faudra que l'applet dont on créera 2 instances utilise Java Card 2.2.1 et implémente l'interface `MultiSelectable`.

On ajoutera une commande avec l'INS à 0x50 pour pouvoir récupérer le canal logique afin de faire du debug.

On sélectionnera les applications sur le bon canal avec `/select <AID> numero_du_channel` et les commandes suivantes verront leur CLA impacté par le numéro du channel. Toutefois avec la version du plugin JCOP que nous avons, si la carte est en T=0 et qu'elle doit transmettre des données dans le cas 4 des APDU, le GET RESPONSE sera fait avec un CLA à 0x00 (au lieu de contenir le channel), ce qui pose problème (il s'agit d'un bug – si vous faites les opérations à la main avec TestResMan, tout fonctionnera) !

Nous pouvons donc développer un client Java réalisant les opérations proprement en utilisant `openLogicalChannel()` sur un objet `Card`.

Astuce : Si on ne dispose pas de carte supportant les canaux logiques, on peut ne travailler que sur une seule instance ou encore sélectionner et désélectionner les instances pour travailler alternativement avec chacune d'elle :-)

Est ce que ce protocole est sécurisé ? En terme d'authentification mutuelle ?

TP n°2 – Authentification mutuelle inter-applets

Objectif : Mise en œuvre d'un processus d'authentification inter-applets.

Afin d'assurer qu'il n'y a pas eu d'usurpation d'AID sur une plate-forme, une applet cliente peut vouloir vérifier que l'applet serveur dispose bien d'un secret commun avec elle. Inversement, l'application serveur peut n'autoriser l'accès à ses services que si elle a confiance dans l'application cliente (c'est-à-dire si elle partage un secret commun avec elle) ?

Ces vérifications peuvent être réalisées au travers d'un mécanisme d'authentification mutuelle.

Afin de rester simple, il vous est demandé d'implémenter un mécanisme simple d'authentification mutuelle entre les applets basés sur du challenge/response, une clé symétrique (identique :-)) utilisée par chaque applet pour chiffrer le challenge.

Exemple de scénario d'authentification :

1. l'applet cliente génère et envoie un nombre aléatoire au serveur
2. l'applet serveur chiffre le nombre aléatoire reçu et renvoie ce cryptogramme ainsi qu'un nombre aléatoire qu'elle génère au client
3. l'applet client vérifie le cryptogramme et chiffre le nombre aléatoire reçu ; elle envoie ensuite ce cryptogramme à l'applet serveur
4. l'applet serveur vérifie le cryptogramme et donne son accord.

Un échec à une des étapes doit mettre fin au protocole d'authentification. L'échec peut provenir que les deux applets n'ont pas les mêmes clés. Cela peut aussi être un « attaquant » qui essaye de sauter une ou des étapes(s).

Comment procède-t-on pour mettre en œuvre le processus d'authentification entre l'applet cliente et l'applet serveur, sachant que le firewall est présent ?

On pourra utiliser dans nos messages un champ destiné à identifier l'étape liée au message. On pourra maintenir l'état courant sur chaque application.

On pourrait aussi imaginer offrir une interface Shareable à l'issue du processus d'authentification dans notre **méthode**.

Commentaire : On pourrait aussi imaginer que dans `getShareableInterfaceObject(...)` l'applet serveur essaye de récupérer une interface Shareable au niveau du client via le mécanisme partageable : mais c'est un peu le serpent qui se mord la queue !

Amélioration : on pourrait ajouter un compteur d'authentifications afin d'ajouter de la sécurité. On peut aussi ajouter des mécanismes comme un temps d'attente qui augmente à chaque tentative d'authentification échouée (réfléchissez à comment on peut réaliser un tel mécanisme en Java Card).

TP n°3 – Utilisation du PIN Global de GP

Objectif : Prise en main de l'API GP pour l'utilisation du PIN Global

Dans ce TP, nous allons développer une application utilisant l'API GlobalPlatform pour fixer, vérifier le PIN Global.

Pour ce TP cette application sera installée avec le privilège adapté (CVM Managment) dans l'ISD (Issuer Security Domain).

TP n°4 – Utilisation des Secure Channels fournis par GP

Objectif : Utilisation de l'API GP pour gérer l'authentification et la sécurité du canal (confidentialité et intégrité)

Dans ce TP, nous allons développer une application utilisant l'API GlobalPlatform pour exploiter le mécanisme de Secure Channel afin de réaliser une authentification mutuelle et pour pouvoir choisir le niveau de sécurité souhaité pour le canal.

L'applet sera installée dans l'ISD (Issuer Security Domain) et mettra en œuvre une authentification mutuelle avec les clés par défaut.

Afin de tester les niveaux de sécurité du canal, nous réaliserons un APDU faisant l'écho de 0x10 octets passés comme données dans la commande APDU.

TP n°5 – Personnalisation d'une application via une SD et changement d'état via les API GP

Objectif : Utilisation d'un Security Domain pour personnaliser une application. Mise en œuvre des différents états d'une application.

Dans ce TP, nous allons développer une application qui sera personnalisée par le Security Domain auquel elle sera associée. L'application devra donc implémenter une interface spécifique de l'API GP : Application.

Le package et l'application seront installés dans l'ISD et celui-ci sera utilisé pour personnaliser l'application.

À l'issue de la personnalisation, nous utiliserons l'API GP, pour passer l'application dans un état SELECTABLE.

TP n°6 – Création d'un Security Domain, personnalisation de ses clés et installation d'une application dans ce SD.

Objectif : Création d'un Security Domain, gestion de ses clés, installation d'application associée et utilisation de ses services.
--

Dans ce TP, nous allons utiliser JCOPShell pour :

- créer un Security Domain
- personnaliser ses clés
- pour installer notre application dans ce SD.

À l'issue de ces opérations, nous utiliserons les classes de GlobalPlatform Pro pour développer notre propre client afin d'automatiser les opérations.