Application du protocole SVC pour la communication dans un réseau de transport public

Marseille, 02/12/2016

v2.0

Préface

Dans le cadre d’une collaboration du Groupe de Recherche ERISCS, on conduit un projet pour la RTM concernant la conception d’un système de communication sécurisé entre les bus et le(s) serveur(s) de surveillance.

Ce document détaille la conception d’une version améliorée par nos soins d’un protocole de communication s’appelant SVC [1], qui s’adapte le mieux au besoin de notre partenaire industriel.

Je me permets de remercier Messieurs T. MUNTEAN, A. FEVRIER, l’équipe RTM et tous mes collègues de m’avoir aidé tout au long de ce travail.

Table de contenu

I. Problématique

La Régie des Transports Marseillais (RTM) est la société qui est en charge du service de transports (bus, métro, etc…) à Marseille – une de trois plus grandes villes de France. Dans les efforts d'évolution et d'amélioration de ses services, en particulier le réseau de transport en bus, la RTM a besoin d'y mettre en place un système de monitoring et surveillance temps-réel, qui est une des exigences de la spécification EBSF [2].

Pour supporter la communication entre les bus et les serveurs de la RTM, des réseaux sans-fil hétérogènes seront utilisés (wifi, 3G, 4G), sur lesquels doit être conçu un nouveau protocole de communication sécurisé. Des tels protocoles existent déjà sur le marché, mais rien n'a été conçu pour les besoins spécifiques rencontrés dans les réseaux de transport urbains.

Notre mission est donc d'analyser la nature de la communication, les avantages et désavantages des solutions existantes et de proposer une solution la plus adaptée aux spécifications des besoins.

II. Analyse

1. La nature de la communication entre les bus et le serveur

La communication dans le réseau de transport est un type de communication spécifique. Elle subit des contraintes et obligations imposées par la nature du service. Nous pouvons résumer ici les contraintes suivantes :

- **Basculement rapide de connexion** : en raison de la mobilité des bus, la connexion vers le serveur est souvent interrompue quand le bus sort d'une zone de couverture d'une antenne vers celle d'une autre, ou est empêchée par de grands obstacles physiques (bâtiments, tunnels, etc…). Le changement du type de réseaux peut aussi être nécessaire dans le cas où le support du réseau courant ne peut pas desservir toutes les stations qui sont autour d'un point d'accès, ou les débits s'avèrent insuffisants pour répondre aux contraintes dynamiques de communication des applications embarquées dans le bus (ex. vidéo-surveillance).

- **Gros volume de données** : les données récoltées pendant les trajets du bus, concernant le statut de tous les équipements, la position cartographique du véhicule et les points d'accès, et notamment les images de vidéo-surveillance, doivent être envoyées vers le serveur avec des contraintes de temps spécifiques. Non seulement un réseau haut débit mais aussi un bon protocole de transmission qui peut réduire, dynamiquement si nécessaire, la surcharge de bande passante est requise.

- **Diversité de types et de priorités de données :** les données remontées vers le serveur viennent de sources différentes, certaines sont capables de tolérer des erreurs (par exemple quelques trames de vidéo-surveillance), d'autres requièrent strictement d'être livrées sans erreurs. On utilise aussi des priorités d'envoi, avec lesquelles les données sont traitées et envoyées dans le cas de congestion.

- **Temps réel :** toutes les données de surveillance doivent être transmises au serveur dès qu'elles sont disponibles, pour que les contrôleurs aient des réactions à temps en cas d'incidents. Aujourd'hui ces données sont enregistrées pendant les trajets et ne sont exploitables qu'après lecture à l'arrivée du bus au dépôt, souvent en fin de journée.

- **Sécurité :** les informations collectées dans le bus, y compris les images des passagers sont liées à la vie privée et doivent donc être transmises et exploitées de façon sécurisée. Une méthode de chiffrement appropriée est indispensable dans l'ensemble de la solution proposée et de bout-en-bout pour tout échange avec les serveurs quelques soient les réseaux sans fil utilisés.

2. Les exigences d’un protocole de communication spécifique

Suite aux caractéristiques de la communication citées au-dessus, un protocole qui s’adapte à ces besoins doit posséder les propriétés suivantes :

- Sans connexion : une communication avec un basculement dynamique et efficace de connexion ne devrait pas utiliser un protocole de transport en mode connecté comme TCP, ce qui augmenterait la latence de connexion à cause de sa phase de négociation et des contrôles inutiles. Ainsi, un protocole de type UDP restera notre choix, mais un protocole « hybride » a été considéré comme remplaçant avec des contrôles simplifiés pour la garantie de livraison de paquets.
- Reprise de connexion rapide : dans le même but de baisser les temps de connexion/reconnexion, on utilisera une identité d’extrémité pour déterminer le bout de communication.
- Confidentialité persistante parfaite : pour bien protéger les identités des clients et les données des services, le protocole sécurisé doit supporter la **confidentialité persistante parfaite** (« perfect forward secrecy » en anglais). Comme ça, il garantit que la découverte par un adversaire de la clé privée d'un correspondant (secret à long terme) ne compromet pas la confidentialité des communications antérieures.
- Cryptographie elliptique : une courbe elliptique sera utilisée pour réduire la taille des clés en gardant le même niveau de sécurité, économiser la bande passante et améliorer la vitesse de chiffrement.
- Indépendance de l’environnement support : le protocole est destiné à utiliser sur plusieurs plateformes et appareils, il faudrait donc être indépendant des versions du système d’exploitation et des bibliothèques externes, et fournir sa propre implémentation des algorithmes et services.
- Sécurité prouvée de bout-en-bout : les réseaux utilisés sont en général hétérogènes pour un réseau complexe de transport métropolitain ; les propriétés de sécurité exigées par les applications doivent être respectées de bout-en-bout pour le protocole de communication.

3. Solutions existants et motivations pour un nouveau protocole

Les protocoles sécurisés « classiques » utilisés de nos jours sont basés essentiellement sur TCP : TLS, SSH, OpenVPN, etc…

Déjà, le TCP n'est pas un choix idéal pour notre solution. En effet, il ne satisfait pas la première caractéristique de la connexion car l'établissement et la reprise de la connexion TCP prend toujours du temps considérable. Le changement dynamique de support de connexion réseau es donc compromis. De plus, la garantie de livraison de paquets n'est pas nécessaire en permanence car souvent les données applicatives sont tolérantes aux pertes. Le compromis entre fiabilité et efficacité pour TCP augmente la latence de connexion, la rendant moins efficace à utiliser dans un contexte temps-réel.

On peut par contre trouver quelques versions UDP des protocoles ci-dessus (comme par exemple DTLS), ou ces protocoles offrent une option UDP eux-mêmes. Pour garder la même opérabilité qu'en TCP, des modifications supplémentaires sont introduites. Dans l'optique de minimiser la complexité, ce n'est pas non plus une bonne décision.

D'ailleurs, comme indiqué dans la définition initiale du protocole SVC par le groupe ERISCS[1], les protocoles mentionnés imposent encore des limites pour la protection d'identité de clients ou l'exposition des informations sensibles.

Dans cette optique est née l'idée de SVC, un nouveau protocole sécurisé qui surmonte ces limites en proposant une négociation simple et un découplage entre l'authentification et le chiffrement. SVC implémente les algorithmes de chiffrement les plus récents sans utiliser des sources externes, cela facilitera le processus d'administration et maintenance.

Pourtant, la version originale du SVC a besoin de quelques améliorations pour mieux s'adapter au problème posé par le contexte des applications des réseaux mobiles hétérogènes utilisés par la RTM dans les applications de transport métropolitain. Dans la section suivante, on propose une version modifiée du SVC et la conception de l'ensemble de la solution.

1 G. Risterucci, T. Muntean, L. Mugwaneza. *A new Secure Virtual Connector approach for communication within large distributed systems*. ERISCS Research Group.

4. Les désavantages de la version originale du SVC

On a trouvé beaucoup d'idées innovantes dans la conception du SVC, parmi lesquelles on a choisi à développer l'idée de *découplage de l'authentification et du chiffrement*, et l'idée de *protection d'identité du client*. Et pourtant, SVC présente dans la conception initiale des points faibles à améliorer. Dans la version originale proposée par ses auteurs, le protocole commence par une requête depuis le serveur vers le client (après un succès de connexion). Cette approche est justifiée comme une façon de réduire la durée de la négociation. Nous, on propose d'optimiser encore plus la phase d'initialisation de connexion car SVC consomme 4 exchanges.

Le problème le plus pénalisant c'est le fait d'avoir l'étape d'échange de clés initiée par le serveur ce qui rend le protocole sensible aux attaques par rejeu. Une façon pour l'éviter, c'est de rajouter du timestamps, mais qui ne sera pas une bonne idée (synchronisation de horloges est donc requise).

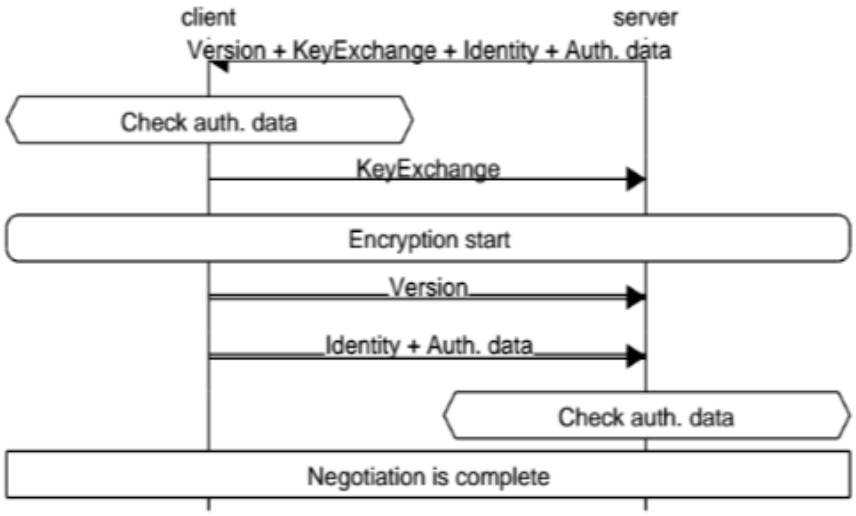


Figure 3.1 – Les échanges dans la version originale du SVC

III. Solution

1 Une version modifiée du SVC

Avec seulement 3 échanges, on propose une nouvelle approche basée sur le principe du « handshaking en trois temps » qui garde encore les idées basiques du SVC.

Dans le diagramme suivant, on distingue les données « entre crochets », de celles qui sont en dehors. Les dernières sont des données de la négociation DH-STS, qui assurent l'autorisation du service. Les premières sont des données d'authentification de l'application qui utilise SVC.

Dans la vérification de l'authenticité, l'approche « challenge – proof » est introduite pour éviter tout type d'attaque par rejeu. Le DH-STS est forcément imperméable [3], seulement vulnérable à quelques attaques du type « unknown key-share » [4].

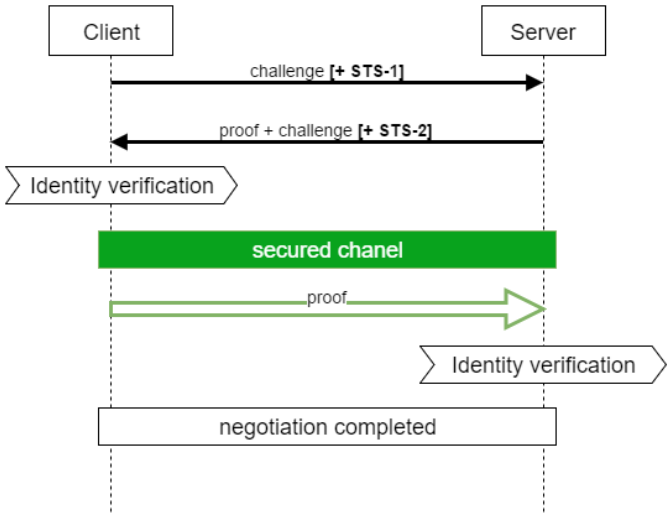


Figure 3.2 – Les échanges dans la version modifiée du SVC

Comparée avec la version originale, la nouvelle que nous proposons ici montre des avantages supplémentaires :

- Moins d'échanges (3 contre 4)
- Résistant aux attaques par rejeu
- Découplage de service

1. L'architecture « service-applications »

Pour mettre en place l'idée de *découplage de l'authentification et du chiffrement,* on va séparer la couche « application », qui est en charge de l'authentification et l'échange de données, avec une instance de la couche « daemon », qui s'occupe des services de chiffrement et de négociation. Comme ça :

- Les services gèrent toutes les « connexions » depuis et vers des hôtes et sont en charge de la reconnexion. La transmission de données au niveau d'application se fait de façon transparente.

- La mise à jour se fait en simplement remplaçant et redémarrant l'instance du daemon. Toutes les applications restent intouchées.

Le diagramme ci-dessous présente les composant dans la nouvelle architecture :

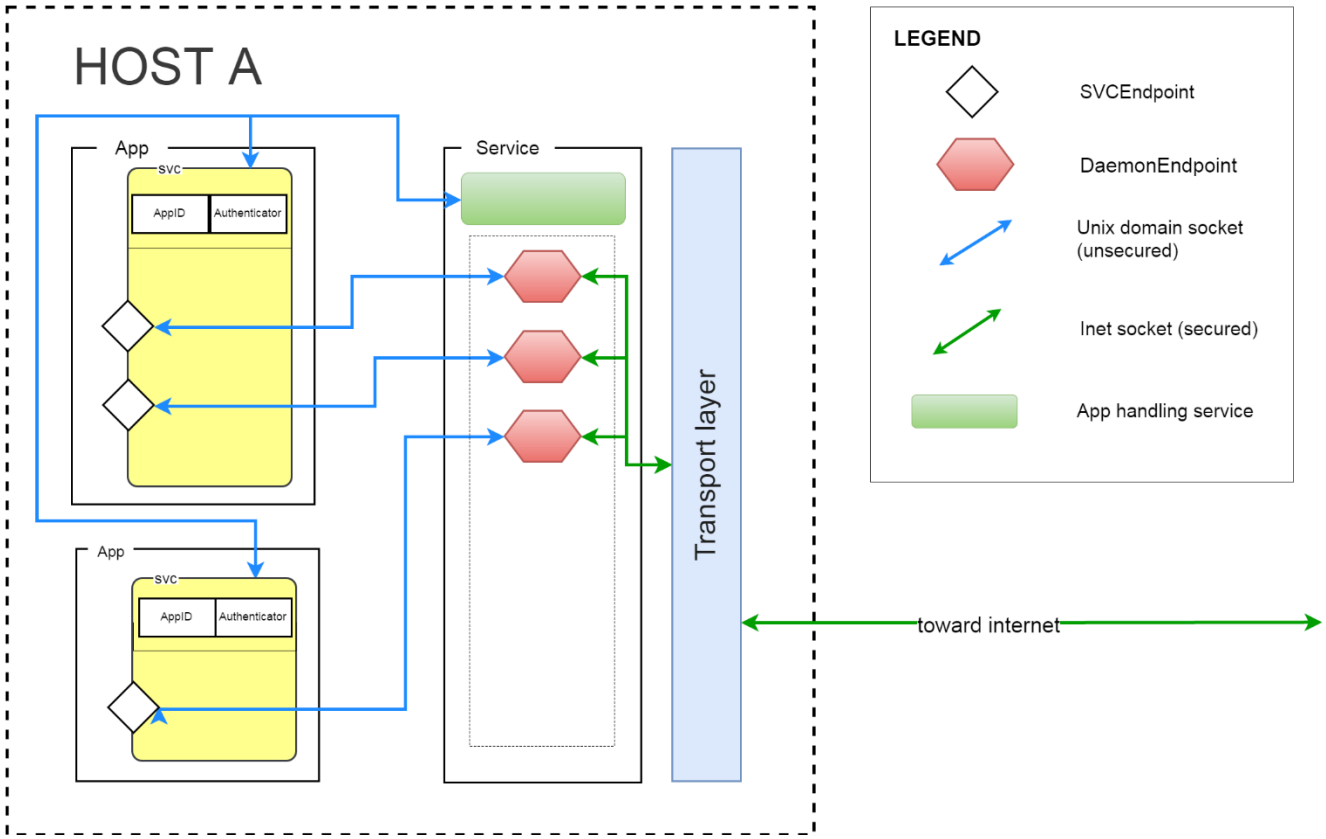


Figure 3.3 – L'architecture « service-applications » du SVC

Conception détaillée

IV. Interface de programmation du SVC

1 SVCEndpoint

Un SVCEndpoint est un bout de communication, à travers lequel les données sont envoyées et reçues de façon transparente. Pour cela, une interface « socket » facile et simplifiée est introduite.

```
class SVCEndpoint{

public:

        bool negotiate();

        int sendData(const uint8_t* data, uint32_t dalalen);

        int sendData(const uint8_t* data, uint32_t datalen, uint8_t option);

        int readData(uint8_t* data, uint32_t* len, int timeout);

        void shutdown();

        ...

}
```

Figure 4.1 – Interface de programmation du SVCEndpoint

La méthode ‘negotiate’ déclenche le processus de négociation. Elle effectue des échanges nécessaires avec les services de fond pour sécuriser le canal de communication. Un mécanisme d’IPC (inter-process communication) est donc utilisé (Unix domain socket).

Après un succès de négociation (negotiate retourne TRUE), un SVCEndpoint est prêt pour la transmission de données. Pour fermer le canal, la méthode ‘shutdown’ sera utilisée.

2 SVC et la génération d’un SVCEndpoint

Dans l’architecture du SVC, les programmes s’exécutent au mode « singleton », c’est-à-dire qu’il n’y a qu’une seule instance qui est

active pour un moment donné. Ces programmes se distinguent par des identités s'appelant appID, qui seront fournies au constructeur du SVC.

Porter le même nom du protocole, une instance de la classe SVC représente le programme qui l'utilise. Elle possède l'identité du programme (appID) et le mécanisme d'authentification (SVCAuthenticator).

A partir de l'instance SVC créée, pour obtenir un socket SVCEndpoint, effectuer une connexion vers un SVCHost, ou écouter une demande de connexion. Chaque application peut avoir plusieurs connexions depuis ou vers des hôtes.

```
class SVC{

public:

        SVC(std::string appID, SVCAuthenticator* authenticator);

        SVCEndpoint* establishConnection(SVCHost* remoteHost, uint8_t
        option);

        SVCEndpoint* listenConnection(int timeout);

        void shutdown();

};

//-----------------------------------------------------------------

int main(int argc, char** argv){

        SVCAuthenticator* auth = new
        SVCAuthenticatorSharedSecret(path_to_shared_secret);

        SVC* svc = new SVC("FILE_COPY_APP", auth);

        //This line will throws error, as FILE_COPY_APP has been used
        // SVC* svc2 = new SVC("FILE_COPY_APP", auth);

        SVCHost* remoteHost = new SVCHostIP("FILE_COPY_SERVER_APP",
        "192.168.1.10");

        SVCEndpoint* endpoint1 = svc->establishConnection(remoteHost,
        SVC_NOLOST | SVC_SEQUENCE);
        SVCEndpoint* endpoint2 = svc->listenConnection(3000);

        if ((endpoint1 != NULL) && (endpoint1->negotiate())){
           //send data
           endpoint1->shutdown();
        }
        svc->shutdown();
```

```
    }
```

Figure 4.2 – Interface de programmation du SVC et exemple

3 Les mécanismes d'authentification et leur interface

Un des avantages du protocole SVC est la liberté de choix des mécanismes d'authentification (secret partagé, infrastructure de clés publiques, etc…). Pourtant, l'implémentation de ces mécanismes doit suivre le modèle « challenge – réponse », dans lequel le client (ce qui lance la demande) et le serveur (ce qui écoute) partagent une connaissance commune (un mot de passe, une autorité de certification, un protocole, etc…), et héritent une interface générique comme ci-dessous.

```
class SVCAuthenticator{

        public:
                SVCAuthenticator(){}
                virtual ~SVCAuthenticator(){}

                virtual std::string generateChallenge(const std::string&
                challengeSecret)=0;
                virtual std::string resolveChallenge(const std::string& challenge)=0;
                virtual std::string generateProof(const std::string& challengeSecret)=0;
                virtual bool verifyProof(const std::string& challengeSecret, const
                std::string& proof)=0;
                virtual std::string getRemoteIdentity(const std::string&
                challengeSecret)=0;
          };
```

Figure 4.3 – Interface du modèle « challenge – réponse »

Quelques mécanismes compatibles avec ce modèles sont :

- SVCAuthenticatorSharedSecret : tous les parties partagent un secret
- SVCAuthenticatorPKI (à développer) : les parties partagent une autorité de certificat et les clés publiques
- SVCAuthenticatorGroupSignature (à développer) : les parties appartiennent à un groupe de signature

5. Les options

Les options caractérisent le comportement des sockets et des paquets circulent dans l'infrastructure du SVC. Il y en a :

- SVC_NOLOST : assurer la livraison des paquets de bout en bout

- SVC_SEQUENCE : assurer l'ordre des paquets lors de la réception
- Etc… (voir svc-header.h)

Elles sont disponibles pour SVC::establishConnection (définir le comportement par défaut du SVCEndpoint à la création) et SVCEndpoint::sendData (outrepasser le comportement par défaut pour l'envoie d'un seul paquet) et peuvent être combinées par opération XOR.

V. Les trames SVC

Tous les échanges entre les services et applications du SVC sont encapsulés dans des trames s'appelant SVCPacket.

Il y a deux types de trames : les trames de commandes servent à établir et gérer la connexion, les trames de données contiennent les données de la communication (chiffrées ou non chiffré selon le contexte).

Quel que soit son type, toutes les trames SVC commencent par :

- **InfoByte** – 1 octet : contient des informations basiques d'un paquet
  - 0x80 : SVC_COMMAND_FRAME : déterminer le type de trame : commande/données (1/0)
  - 0x40 : SVC_NOLOST
  - 0x20 : SVC_SEQUENCE
  - 0x10 : [pas d'usage]
  - 0x08 : SVC_ENCRYPTED : déterminer si la trame est chiffrée : chiffrée/non-chiffrée (1/0)
  - 0x04 : [pas d'usage]
  - 0x03 : SVC_URGENT_PRIORITY : ce paquet sera traité avec un maximum de priorité
  - 0x02 : SVC_HIGH_PRIORITY : ce paquet sera traité plus prioritairement
  - 0x01 : SVC_NORMAL_PRIORITY : ce paquet sera traité normalement
  - 0x00 : SVC_LOW_PRIORITY : ce paquet sera traité avec un minimum de priorité

- **EndpointID** – 8 octets : identité d'extrémité de connexion. Les bouts d'une connexion partagent une même identité. Une endpointID est unique parmi celles des connexion d'un même hôte. L'utilisation de l'extrémité de connexion permet la reprise rapide en cas de reconnexion. La structure d'une endpointID est la suivante :

- 4 premiers octets : appID du programme
- 2 octets suivant : compteur de connexion côté client
- 2 derniers octets : compteur de connexion côté serveur

Grâce à cette approche, on obtient toujours une identité unique pour chaque connexion. Les derniers octets de l'EndpointID seront communiqués lors de SVC_CMD_CONNECT_OUTER2.

- **Sequence** - 4 octets : la sequence permet de distinguer les trames SVC entre elles. Cette information est aussi utilisee comme valeur du vecteur de chiffrement AES.

Une visualisation d'une trame SVC est donnée ci-dessous :

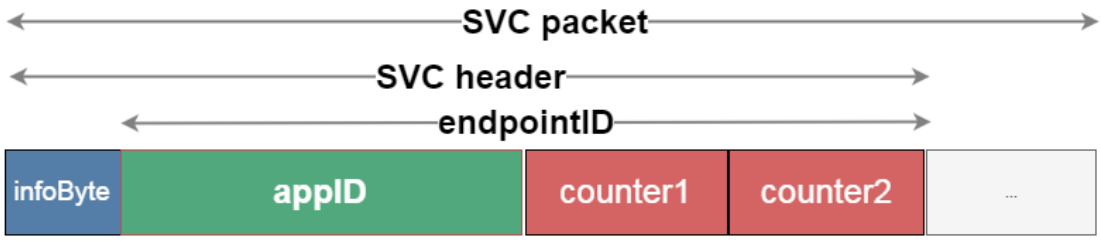


Figure 5.1 – La structure générale d'une trame SVC

Une trame de données du SVC contient, suivant l'entête, 4 octets de longueur de la charge utile de données, suivie par la charge elle-même et le HMAC (la longueur du HMAC dépendent de la version du SVC). Une trame de données doit être toujours chiffrée.

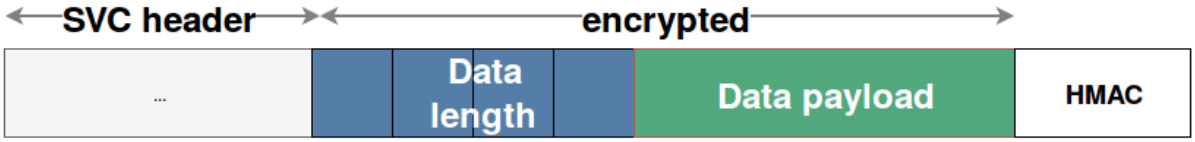


Figure 5.2 – Une trame de données du SVC

Une trame de commande du SVC commence par l'identité de commande (CID) suivre le header. Le reste est la partie des paramètres, dans laquelle chaque paramètre est succédé par 2 octets de longueur. Sauf les commandes d'initialisation de connexion qui ne sont pas chiffrées, toutes les autres sont chiffrées, et se terminent par un HMAC.

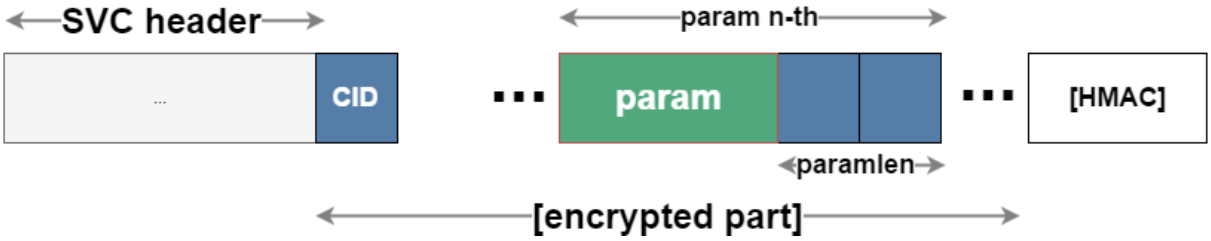


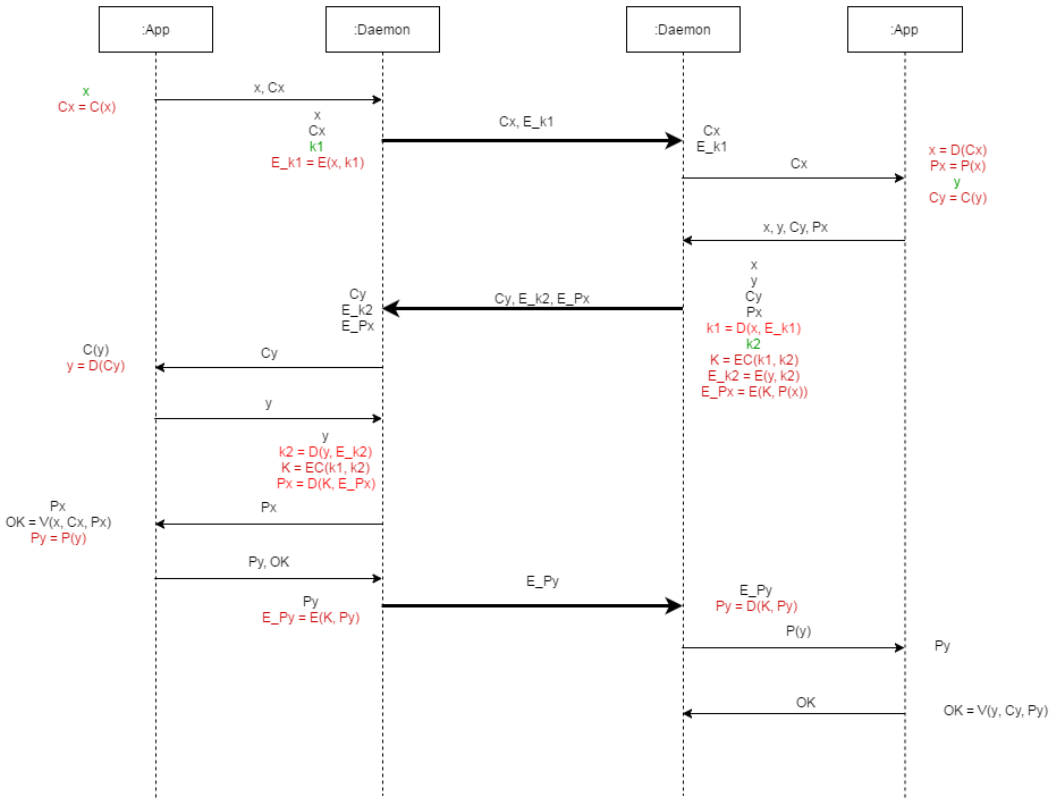
Figure 5.3 – Une trame de commande du SVC

VI. Le processus d’authentification

On se rendre plus au détail du modèle d’authentification « challenge – réponse » qu’utilise SVC.

Le succès du processus d’authentification se base sur le fait qu’une entité peut résoudre et prouver qu’il peut résoudre un challenge d’une autre entité (et inverse). Un challenge est une transformation d’un secret généré de façon aléatoirement.

A noter que tous ceux qui peuvent résoudre le challenge (et donc trouver le secret) sont considérés comme une entité valide de la communication. Le protocole est vulnérable aux attaques MITM si un adversaire peut résoudre le challenge. En raison de cela, une implémentation de l’interface SVCAuthenticator doit assurer la confidentialité du challenge généré.



C(x): la fonction de génération de challenge
D(x): la fonction de résolution de challenge
P(x): la fonction de génération de preuve
V(x, y, z): la fonction de vérification que la preuve z conforme à la challenge y, sous le secret x
H(x): fonction d'hachage unidirectionnelle

E(x, k): chiffrement symétrique de k avec clé x
D(x, k): déchiffrement symétrique de k avec clé x
EC(x, y): la génération de clé basé sur x et y

Figure 6.1 – Le détail du modèle « challenge – réponse » du SVC

Au niveau de service, les instances de DaemonEndpoint sont en charge des logiques de chiffrement et négociation de clé (Diffie – Hellman). L'accès à ces instances et leurs données doit être strictement protégé.

Ces données sont bien évidemment encapsulées dans des trames SVC de type commande.

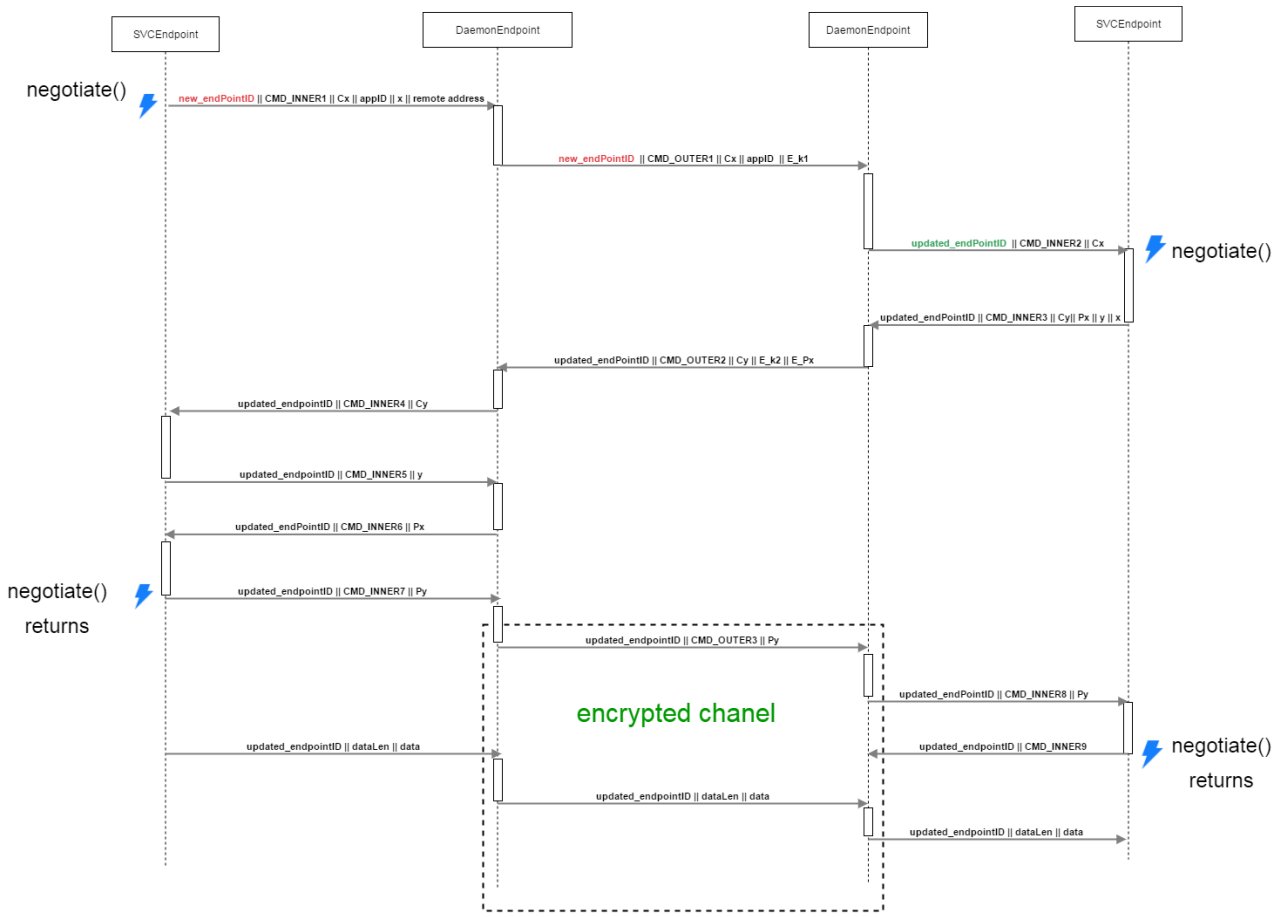


Figure 6.2 – Le processus d’initialisation de connexion du SVC

VII. Références

[1] G. Risterucci, T. Muntean, L. Mugwaneza. “A new Secure Virtual Connector approach for communication within large distributed systems”, ERISCS Research Group.

[2] Intelligent garage and predictive maintenance,

[3] Station-to-Station protocol,

[4] Blake-Wilson, S.; Menezes, A. (1999), "Unknown Key-Share Attacks on the Station-to-Station (STS) Protocol", *Public Key Cryptography*, Lecture Notes in Computer Science, **1560**, Springer, pp. 154–170