École Centrale de Lyon

Travail de fin d'études

Intégration de la RFC 1086

Mise au format XML de fichiers bancaires

Présenté le 19 mai 2009

Tuteur ECL :
Mohsen Ardabilian

Étudiant :
Philippe Mané

Tuteur entreprise :
Philippe Carreras

Promotion 2008

septembre 2008 – mars 2009

Resum

El treball de fi d'estudis (TFE) presentat es va realitzar a la societat AFSOL, ubicada a Perpinyà. Aquesta ofereix aplicacions de monètica, o sigui de gestió de les transaccions financeres electròniques, als bancs i alguns organismes privatius, principalment francesos.

Es tractava primer d'implementar, dins de la biblioteca de xarxa de les aplicacions d'AFSOL, la RFC 1086, que descriu un protocol de conversió entre xarxes X.25 i TCP/IP. De fet, el protocol X.25, tot i que encara llargament utilitzat a l'estat francès per a aplicacions financeres, és obsolescent, d'aquí el seu reemplaçament progressiu. Si bé aquesta fase es va dur a terme, va topar amb dificultats lligades a "dades de crida", transportades pel paquet de sol·licitud d'obertura de sessió X.25, que impedeixen l'ús del protocol bancari CB2A abans de la seva versió 1.2.

D'altra banda s'havia de considerar el pas al format XML dels arxius de transaccions financeres, que fins ara posseïen una estructura fixa i seqüencial. Així es prepara l'arribada del protocol EPAS, encara no publicat, que es deriva de la norma ISO 20022 i té per objectiu la interoperabilitat dels protocols bancaris a nivell europeu. S'ha pogut establir una bona base que permet la conversió dels antics arxius, i que queda compatible amb aquestes parts de les aplicacions d'AFSOL que encara no tindrien en compte el format XML.

Nogensmenys, com que va quedar clar durant la fase d'estudi de la RFC 1086 que calia protegir les dades que des d'ara circularien per xarxes TCP/IP públiques, va aparèixer al llarg del TFE una tercera part, que va consistir en l'estudi de la capa criptogràfica SSL, les primeres proves de comunicació xifrada amb un terminal de punt de venda, a més d'un panorama del material i del programari criptogràfics que haurien de formar part de la infraestructura futura .

Paraules claus: monètica, targeta de crèdit, terminal de punt de venda, transacció, X.25, TCP/IP, RFC 1086, TP0, dades de crida, SSL, certificat X.509, infraestructura de clau pública, autenticació, XML, esquema XML del W3C, XPath, ISO 20022, UNIFI, EPAS, CB2A, CBPR.

Résumé

Le travail de fin d'études (TFE) présenté s'est déroulé au sein de la société AFSOL, située à Perpignan. Celle-ci offre des applications de monétique, permettant le gestion des transactions financières électroniques, aux banques et à certaines organismes privatifs, principalement français.

Il était d'abord question d'implémenter, dans la bibliothèque réseau des applications d'AFSOL, la RFC 1086, qui décrit un protocole de conversion entre réseaux X.25 et TCP/IP. En effet, le protocole X.25, bien qu'encore largement utilisé en France pour les applications financières, est obsolescent, d'où son remplacement progressif. Bien que cette phase ait pu être menée à bien, elle s'est heurtée à des difficultés liées aux « données d'appel », transportées par le paquet de demande d'ouverture de session X.25, qui empêchent l'utilisation du protocole bancaire CB2A avant sa version 1.2.

D'autre part il fallait considérer le passage au format XML des fichiers des transactions financières, qui jusqu'à présent possédaient une structure fixe et séquentielle. De la sorte, on prépare l'arrivée du protocole EPAS, encore non publié, qui dérive de la norme ISO 20022 et a pour objectif l'interopérabilité des protocoles bancaires au niveau européen. Une bonne base a pu être établie ; elle permet la conversion des anciens fichiers, et reste compatible avec les parties des applications d'AFSOL qui ne prendraient pas encore en compte le format XML.

Néanmoins, il s'est avéré pendant la phase d'étude de la RFC 1086 qu'il convenait de protéger les données qui circuleraient dorénavant par des réseaux TCP/IP publics ; une troisième partie est alors apparue, qui a consisté à l'étude de la couche cryptographique SSL, les premiers essais de communication chiffrée avec un terminal de paiement électronique, ainsi qu'un panorama du matériel et des logiciels cryptographiques qui devraient faire partie de la future infrastructure.

Mots clefs : monétique, carte de paiement, terminal de paiement électronique (TPE), transaction, X.25, TCP/IP, RFC 1086, TP0, données d'appel, SSL, certificat X.509, infrastructure à clefs publiques, authentification, XML, schéma XML du W3C, XPath, ISO 20022, UNIFI, EPAS, CB2A, CBPR.

Abstract

The end-of-studies internship (*travail de fin d'études*, TFE) we present took place within the company AFSOL, located in Perpignan, which offers electronic banking solutions to banks or some private institutions, mainly French ones.

Our first task was to implement RFC 1086 within AFSOL's network libraries. This RFC describes a conversion protocol between X.25 and TCP/IP networks. Indeed, the X.25 protocol, however still widely used in France for financial applications, is obsolescent, hence its replacement. Although this phase was completed, we faced difficulties related to "call user data", transported by X.25 session establishment packets, which prevent the use of CB2A banking protocol before its version 1.2.

Subsequently, financial transactions files had to be written in XML, instead of the original fixed and sequential format. This opens the way to the adoption of the ISO 20022 compliant EPAS protocol, not yet published, whose aim is interoperability at the European level. We provided a good basis, able to convert old files and compatible with parts of the application that would not be "XML aware" yet.

Nevertheless, it turned out, while RFC 1086 implementation was still under study, that data which from now on would circulate through public TCP/IP networks had to be protected. This became a third part of our work, which included information retrieval about the SSL cryptographic layer, basic enciphered communication tests with a point of sale terminal, as well as a survey about what cryptographic software and hardware should be used in a forthcoming infrastructure.

Keywords: electronic banking, credit card, point of sale terminal (POS terminal), transaction, X.25, TCP/IP, RFC 1086, TP0, call user data, SSL, X.509 certificate, public key infrastructure, authentication, XML, W3C XML schema, XPath, ISO 20022, UNIFI, EPAS, CB2A, CBPR.

Remerciements

So long, and thanks for all the fish !

The Hitchhiker's Guide to the Galaxy
Douglas Adams

Je remercie Messieurs Alain Maravitti et Philippe Carreras de m'avoir accueilli au sein de leur société pendant ces cinq mois, ainsi que le reste des membres de l'équipe, pour leur prévenance et leur observance des rites gastronomiques hivernaux.

Toute ma reconnaissance à Monsieur Mohsen Ardabilian, responsable de l'option « Informatique et communication » et tuteur ECL, qui a fait en sorte que ce TFE puisse avoir lieu dans de bonnes conditions malgré des circonstances particulières.

Je sais également gré aux membres de ma famille et amis qui ont aidé à la relecture de ce rapport, quand bien même les les notions abordées pouvaient être relativement éloignées de leur domaine de prédilection.

Table des matières

Introduction **1**

1 Cadre du TFE **2**
1.1 Présentation de l'entreprise 2
1.1.1 Historique et domaine d'activité 2
1.1.2 Place d'AFSOL dans le marché et environnement économique 3
1.2 Cadre technique, monétique et STAP 3
1.2.1 À propos de la monétique 3
1.2.2 Présentation technique de STAP 4
1.2.3 Modèle commercial 5
1.2.4 Quelques enjeux se profilant sur le marché de la monétique 5

2 RFC 1086 **7**
2.1 Utilisation du protocole X.25 dans les applications monétiques . 7
2.1.1 Présentation générale du protocole X.25 7
2.1.2 Solutions de conversion X.25/TCP 9
2.2 Présentation de la RFC 1086 11
2.2.1 RFC 1086 et protocoles ISO 11
2.2.2 Protocole proposé par la RFC 1086 13
2.3 Implémentation de la RFC 1086 dans la bibliothèque d'isolation réseau d'AFSOL 14
2.3.1 Mode RFC1086 14
2.3.2 Mode EMULRFC1086 17
2.4 Bilan 18

3 SSL **19**
3.1 Généralités sur SSL 19
3.1.1 Notions cryptographiques de base 19
3.1.2 Certificats X.509 24
3.1.3 Protocole SSL 28
3.1.4 Infrastructure à clefs publiques 31
3.2 Intégration de SSL aux applications monétiques 32
3.2.1 Simple authentification 32
3.2.2 Double authentification 35
3.2.3 Quelques idées à propos d'une infrastructure à clefs publiques (ICP) 36
3.3 Bilan 37

4 **Écriture des bruts au format XML** **38**
4.1 Vue synoptique de la chaîne de traitement 38
4.1.1 Initiation de la transaction et réception des remises (télécollecte) 38
4.1.2 Traitement des remises 39
4.1.3 Exploitation des remises 39
4.2 Protocoles bancaires : de CBPR à UNIFI 40
4.2.1 Protocoles utilisés actuellement 40
4.2.2 ISO 20022 et l'arrivée progressive d'EPAS 40
4.3 Migration vers UNIFI 42
4.3.1 Écriture des bruts avant le passage à XML 42
4.3.2 Passage à XML et compatibilité descendante 43
4.3.3 Outils logiciels 43
4.3.4 Génération des bruts XML 46
4.3.5 Lien avec la structure de référence 47
4.3.6 Intégration à la chaîne de compilation 49
4.3.7 Modifications apportées au code préexistant 52
4.3.8 Développements futurs 53
4.4 Bilan 54

Conclusion **56**

Bibliographie **57**

Annexes **61**

A **À propos de ce document** **62**

B **Résumé des fonctions pour l'écriture des remises au format XML à l'usage du développeur** **63**
B.1 Fonctions afférentes aux remises XML 63
B.1.1 Structures de données définies dans brutX.h 63
B.1.2 Fonctions définies dans brutX.c 64
B.2 Fonctions de chargement du dictionnaire 68
B.2.1 Structures de données définies dans dicoX.h 68
B.2.2 Fonctions définies dans dicoX.c 70
B.3 Fonctions d'ouverture/fermeture et de lecture/écriture par blocs des bruts 71
B.3.1 Fonctions d'ouverture/fermeture dans fctB.c 71
B.3.2 Modification des fonctions de lecture et d'écriture par blocs dans readB.c et writeB.c 72
B.4 Fonctions ajoutées dans l'API 72
B.4.1 Fonctions ajoutées dans xmlfct.h 72
B.4.2 Fonctions ajoutées dans dthe_fct.h 75

C **Génération de certificats X.509 avec OpenSSL** **77**
C.1 Création d'une autorité de certification 77
C.1.1 Environnement de l'autorité de certification 77
C.1.2 Configuration de l'autorité de certification 77
C.2 Signature de certificat 79
C.3 Génération d'une demande de signature de certificat 79

C.4 Divers 80
C.4.1 Retirer une passphrase 80
C.4.2 Créer un conteneur PKCS#12 80

D Utilisation de stunnel **81**
D.1 Généralités 81
D.2 Paramètres en ligne de commande 81
D.3 Paramétrage avec un fichier de configuration 82

Glossary **84**

Index **88**

Table des figures

1.1 Schéma synoptique présentant le fonctionnement de STAP 4

2.1 Éléments constitutifs d'un réseau X.25 8
2.2 Passerelliste 11
2.3 Format de paquet défini par la RFC 1006 12
2.4 Procédure d'enregistrement d'un serveur sur un pont TP0 13
2.5 Transfert de connexion par un pont TP0 14
2.6 Surveillance de la connexion avec le pont TP0 15
2.7 Émulation d'un pont TP0 pour un TPE IP 17

3.1 Utilisation de la cryptographie symétrique 20
3.2 Quelques algorithmes symétriques ou à clef secrète 21
3.3 Utilisation de la cryptographie asymétrique 22
3.4 Quelques algorithmes asymétriques 22
3.5 Signature d'un message grâce à la cryptographie asymétrique . . 23
3.6 Description ASN.1 de la partie principale d'un certificat X.509 . 25
3.7 Chaîne de certificats 27
3.8 Structurations en couches du protocole SSL 29
3.9 Phase de négociation SSL 30
3.10 Infrastructure à clefs publiques 32

4.1 Vue synoptique de la chaîne de traitement 39
4.2 Structure initiale des bruts 42
4.3 Un exemple de structure C correspondant à un bloc transaction . 43
4.4 Rôle initial des structures particulière de référence 44
4.5 Copie d'un nœud du gabarit dans l'arbre XML de la remise . . . 47
4.6 Contenu d'une annotation `xs:appinfo` 49
4.7 Génération du corpus et des gabarits à la compilation 50
4.8 Vision simplifiée du makefile générant le corpus et le gabarit . . . 51
4.9 Schéma UML décrivant la structure de données corpus 52

Introduction

C'était à Mégara, faubourg de Carthage, dans les jardins d'Hamilcar.

Salammbô, Gustave Flaubert

Le travail de fin d'études (TFE) présenté dans ce rapport s'est déroulé, de septembre 2008 à mars 2009, au sein de la société AFSOL, à Perpignan. AFSOL propose aux établissements bancaires, ainsi qu'à des organismes privatifs, des solutions de monétique, permettant le traitement informatisé des transactions financières, notamment celles initiées par carte à puce.

Nous avons été amené à intervenir sur le produit phare de la société, STAP, ou « Serveur de Télécollecte Acquéreur Paiement », dont le rôle consiste à acquérir les données financières provenant de terminaux de paiement électroniques — opération connue sous le nom de « télécollecte » — et à les rendre disponibles aux acteurs financiers.

Initialement il était prévu que ce TFE serait composé de deux parties. La première visait l'étude et l'implémentation de la RFC 1086 dans la bibliothèque d'isolation réseau des applications d'AFSOL. Il s'agissait de mettre en place un protocole réalisant une conversion entre réseaux X.25 et TCP/IP. X.25, pourtant encore largement utilisé pour des applications financières, est une technologie moribonde, en cours de remplacement progressif.

La seconde avait pour but de passer au format XML les fichiers de remises financières. Le format initial, séquentiel et de taille fixe, relativement pauvre d'un point de vue sémantique, n'était guère utilisable en dehors des applications d'AFSOL et compliquait les évolutions. Le passage, à terme, à EPAS, un nouveau protocole bancaire basé sur ISO 20022 qui utilisera XML pour la transmission des messages, a motivé ce remaniement.

Il est cependant apparu, au cours de la mise en œuvre de la RFC 1086, qu'il était nécessaire, afin de protéger les transactions transitant par des réseaux TCP/IP, de mettre en place une couche cryptographique nommée SSL. Cela a donné lieu à une recherche documentaire ainsi qu'à un premier essai des technologies. Ce thème étant un projet à part entière, notre rôle a surtout consisté à poser les bases de son développement futur.

Le chapitre 1 effectue une brève présentation de la société et de STAP. On y situe le contexte des actions qui ont été menées. Au chapitre 2 est exposée la mise en œuvre de la RFC 1086, ainsi que les limites d'utilisation qui sont apparues. Les travaux sur SSL qui en ont découlé sont décrits au chapitre 3. Enfin, le chapitre 4 traite de la migration des fichiers de remises au format XML.

Chapitre 1

Cadre du TFE

Ce chapitre présente le cadre dans lequel s'est déroulé le TFE, il y est question de l'entreprise qui l'a accueilli, et de son domaine d'activité. Nous introduirons quelques concepts liés à la monétique*, ainsi que les composants de l'application sur laquelle nous avons travaillé.

1.1 Présentation de l'entreprise

Ce TFE a été effectué au sein de l'entreprise AFSOL, à Perpignan. AFSOL est une SAS [1] basée à Tecnosud, zone d'activités économiques ayant pour ambition d'accueillir des entreprises spécialisées dans les nouvelles technologies ainsi que des centres de recherche et de formation.

1.1.1 Historique et domaine d'activité

La société est créée en Mars 2005 par MM. Alain Maravitti et Philippe Carreras, suite à la disparition de l'antenne de MoneyLine basée à Perpignan.

Dates clefs pour la société :

- Novembre 1987 : première version de l'application de télécollecte STAP
- Mars 2005 : création de la société AFSOL
- Juin 2005 : achat de la branche d'activité STAP à Moneyline
- Décembre 2005 : entrée du GICM [2] et de Lyra Network [3] dans le capital.

La société AFSOL est spécialisée dans la monétique, c'est-à-dire le traitement informatisée de transactions financières. Elle conçoit principalement des solutions relatives à la télécollecte bancaire (collecte des données relatives aux transactions financières initiées par carte de crédit) et au traitement d'autres moyens de paiement comme le chèque.

Ces applications peuvent être utilisées tant par des banques, que par des fournisseurs de solutions privatives (cartes pétrolières...).

1. Société par actions simplifiée. La responsabilité des associés est limitée au montant de leurs apports.
2. Groupement informatique du Crédit Mutuel
3. « Opérateur monétique d'acheminement de transactions financières* » [**1**], fournissant une solution de conversion entre réseaux X.25 et TCP/IP

1.1.2 Place d'AFSOL dans le marché et environnement économique

La clientèle d'AFSOL est principalement française, mais il est intéressant de noter que 30% du chiffre d'affaire est réalisé dans les départements et régions d'outre-mer.

Jusqu'à présent, les protocoles bancaires étaient au mieux propres à un État ; c'est le cas en France. Dans le pire des cas, il existe un protocole par institution bancaire, comme en Espagne et en Italie. Dans ce contexte, il est relativement difficile de prendre des parts de marché à l'étranger. Nous verrons que la constitution du SEPA [4] et l'adoption de la norme ISO 20022 pourraient changer la donne.

1.2 Cadre technique, monétique et STAP

1.2.1 À propos de la monétique

Le terme *monétique*, que nous venons de définir, n'a pas de définition officielle [5]. Néanmoins, il est bon de définir certains éléments du vocabulaire couramment utilisé dans ce domaine.

Support de la transaction
Il s'agit généralement d'une carte de crédit, mais AFSOL effectue également la télécollecte chèques. Ont été successivement utilisées les cartes à piste magnétique, à logique câblée, et à puce. Cette dernière, que beaucoup d'entre nous utilisent quotidiennement, est un véritable ordinateur spécialisé — exécutant un nombre réduit d'applications — et inviolable. Un premier tour d'horizon est brossé dans [**36**] ; on pourra lire le reste du magazine pour entrer dans les détails.

Émetteur
organisme financier (par exemple, une banque) qui met à disposition du porteur un support. En France, l'émission de cartes bancaires est encadrée par le GIE CB (Groupement d'Intérêt Économique Cartes Bancaires).

Porteur
personne pour qui on met à disposition le moyen de paiement. Ce n'est pas forcément le titulaire du compte. Ainsi, un employé peut être amené à utiliser une carte de crédit attachée au compte de sa société. Il est le seul à en connaître le code confidentiel.

Acquéreur
Banque domiciliaire du commerçant. Celui-ci — l'accepteur — passe un accord avec celle-là en vue de l'acquisition des transactions initiées par carte bancaire, puis de l'introduction de ces données dans les systèmes d'échange des émetteurs. STAP est un exemple de système d'acquisition.

Accepteur
entreprise qui accepte le moyen de paiement pour le règlement d'une transaction.

---

4. Single Euro Payment Area

5. En particulier, il n'est pas attesté dans les documents décrivant les protocoles bancaires. Ainsi, [**2**], sur lequel est basé CB2A, se contente d'utiliser le terme « Messages initiés par carte de transaction financière ».

1.2.2 Présentation technique de STAP

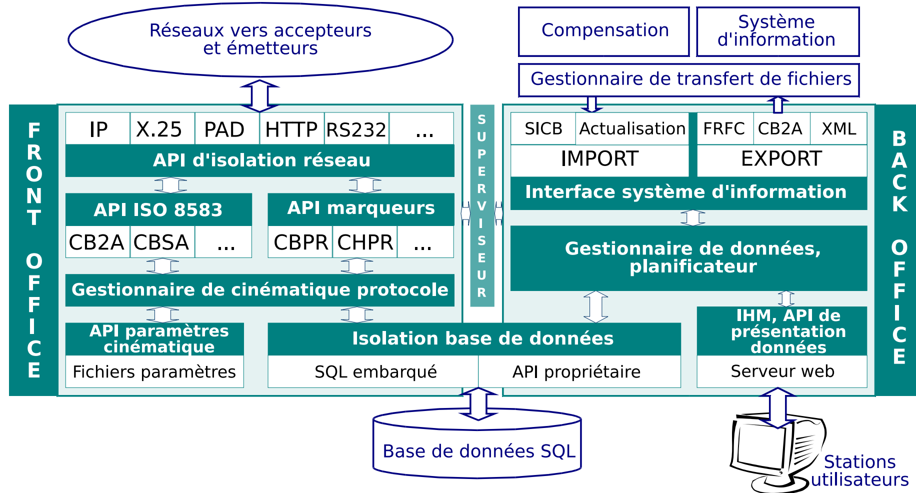


FIGURE 1.1 – Schéma synoptique présentant le fonctionnement de STAP

STAP [6] est une solution permettant la gestion des moyens électroniques de paiement. C'est une plate-forme monétique complète, composée de deux grands ensembles :

- Le « front office », qui gère les communications avec les systèmes d'acceptation et les échanges avec les émetteurs
- Le « back office », pour l'administration des accepteurs et l'exploitation des opérations.

C'est principalement sur le « front office » que j'ai eu à intervenir pendant mon TFE.

Les remises financières* peuvent être reçues à travers de nombreux protocoles réseaux tels que TCP/IP, X.25, GPRS, etc. Pendant très longtemps, c'est presque exclusivement X.25 qui a été utilisé. Ce dernier, bien qu'encore largement utilisé en production pour les applications bancaires, est aujourd'hui considéré comme obsolète, ayant été détrôné par TCP/IP, qui a connu le succès que l'on sait. Il devrait être abandonné à terme. La première partie du TFE consistait à étudier et implémenter un protocole de conversion entre X.25 et TCP/IP décrit par la RFC 1086 [**10**], au sein de la couche d'isolation réseau de STAP. Ce thème est abordé au chapitre 2. Le chapitre 3 discutera de la sécurisation des flux transitant désormais par TCP/IP grâce à SSL*/TLS*.

À travers ces diverses couches réseau peuvent transiter des messages relevant de divers protocoles bancaires [7] [8], comme des remises financières transmises depuis un TPE*. Ces remises sont stockées dans des fichiers appelés *bruts*, sur lesquels j'ai travaillé dans un deuxième temps. Le chapitre 4 présente le dispo-

6. Serveur de Télécollecte Acquéreur Paiement

7. d'où le terme « isolation réseau » précédemment utilisé : on peut transporter du CB2A tant sur IP que sur X.25, par exemple

8. Les protocoles bancaires couramment utilisés sont rapidement introduits à la section 4.2.

sitif mis en place pour la migration de ces fichiers au format XML. Une fois reçues, les informations pertinentes d'une remise sont placées dans une base de données SQL* (étape de traitement).

Le « back office » permet à l'utilisateur (opérateur au sein du centre informatique d'un établissement bancaire ou privatif) de gérer les flux financiers et le parc de TPE, à travers une interface web. La comptabilisation de l'ensemble des transactions réalisées sur une période aboutit à la *compensation**, c'est-à-dire le calcul global des montants à créditer ou débiter aux autres établissements bancaires.

Le cœur de STAP, dont le développement remonte à 1987, est écrit en langage C ; c'est celui que j'ai utilisé chaque fois que j'ai été amené à y intégrer de nouvelles fonctionnalités. L'interface utilisateur, initialement programmée elle aussi en langage C, tend aujourd'hui à être principalement basée sur PHP, et les technologies web usuelles telles que JavaScript (dont son avatar Ajax) et CSS.

STAP tournait initialement sous des Unix propriétaires, comme AIX, Xenix et SCO. Depuis 2001, il a été porté sous GNU/Linux, la migration est presque totalement réalisée chez les clients.

1.2.3 Modèle commercial

Le modèle commercial d'AFSOL peut être qualifié d'hybride. C'est tout d'abord un éditeur logiciel, fournissant des solutions pour les banques et certains organismes financiers privatifs, dont le produit phare est STAP, que nous venons de présenter. Ce dernier est hébergé chez un prestataire de services, ou bien installé chez le client : il est important de noter qu'AFSOL ne joue pas le rôle d'hébergeur.

Néanmoins, AFSOL est aussi un fournisseur de services. Pour la majorité de ses clients, la société assure le support technique et la surveillance des serveurs. Elle est en communication permanente avec une partie de sa clientèle, et se situe à mi-chemin entre un fournisseur et un service interne. C'est un atout dans un contexte de crise, où les organismes financiers cherchent à se séparer des services totalement externalisés.

1.2.4 Quelques enjeux se profilant sur le marché de la monétique

Le marché de la monétique connaît actuellement certains bouleversements, qui ont de près ou de loin un rapport avec ce TFE. Les standards PCI DSS [**4**], élaborés par le consortium Payment Card Industry (PCI), comprenant notamment MasterCard Worldwide et Visa, poussent les acteurs à une sécurisation avancée des systèmes par lesquels transitent des informations sensibles telles que des numéros de cartes bancaires. Cela implique certaines modifications concernant l'architecture et l'administration des systèmes informatiques, mais aussi le chiffrement de telles données, et demande une protection particulières des clefs cryptographiques. Ces thèmes seront abordés respectivement aux sections 4.3.8 et 3.2.1.

Par ailleurs, le protocole EPAS, basé sur ISO 20022, rendra possible dès sa publication l'interopérabilité au niveau européen, ce qui n'est absolument pas le

cas actuellement. Cette perspective a motivé la migration au format XML des fichiers de transactions financières.

Chapitre 2

RFC 1086

La première partie du TFE portait sur l'implémentation d'un nouveau protocole réseau au sein de STAP. La RFC 1086 décrit un protocole de conversion entre X.25 et TCP/IP, que la machine serveur soit du côté TCP/IP ou X.25. Après avoir dressé un rapide portrait de X.25 et exposé la nécessité d'une migration vers TCP/IP, nous présenterons la RFC 1086 et les deux modes qui ont été implémentés dans STAP, selon que la machine serveur est accessible à travers un pont RFC 1086, ou émule un pont RFC 1086 pour un client IP.

2.1 Utilisation du protocole X.25 dans les applications monétiques

2.1.1 Présentation générale du protocole X.25

À l'heure actuelle, les réseaux X.25 sont encore largement utilisés pour le transport des transactions financières*. La présentation qui en est faite ci-dessous s'inspire principalement du guide X.25 fourni par Cisco [**8**].

Éléments constitutifs d'un réseau X.25

X.25 [**8**, **11**, **13**] est un protocole réseau fonctionnant par commutation de paquets en mode point à point, normalisé conjointement par l'UIT [1] et ISO* [2]. Il définit l'interface entre ETTD [3] (ou DTE [4]) et ETCD [5] (ou DCE [6]), sans prendre en compte la nature des systèmes connectés au réseau. Les ETTD peuvent être des terminaux « muets », analogues au Minitel français, ou bien des dispositifs plus sophistiqués comme des ordinateurs équipés d'une carte X.25. Les ETCD sont des dispositifs de communication comme des modems, qui interfacent des ETTD avec le réseau d'un opérateur de télécommunications. Ce réseau est com-

1. Union Internationale des télécommunications
2. International Organization for Standardization ou Organisation Internationale de normalisation
3. Équipement Terminal de Traitement de Données
4. Data Terminal Equipment
5. Équipement Terminal de Circuit de Données
6. Data Communication Equipment

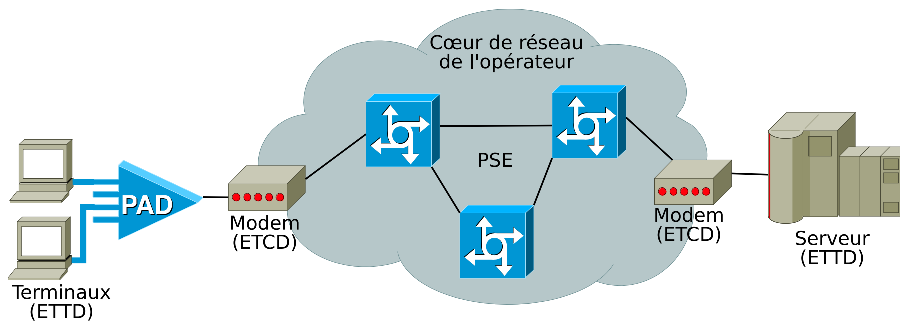


FIGURE 2.1 – Éléments constitutifs d'un réseau X.25

posé de commutateurs de paquets, ou PSE[7], qui se chargent de « router » les paquets en se basant sur le numéro X.121 du destinataire. La figure 2.1 met en relation ces différents éléments.

Une adresse X.121 est une suite de quinze chiffres au maximum, analogue à un numéro de téléphone[8]. Une étape de « routage »[9] correspond à un groupe de chiffres de cette adresse, qui disparaît avant la prochaine étape, à la manière de la fusée Ariane, qui perd des étages à chaque étape de son vol. Arrivé à l'entrée du réseau destinataire, l'éventuel reliquat pourra être utilisé pour multiplexer entre diverses applications ou machines.

Au commencement, les terminaux, qui, comme nous venons de le dire, étaient « muets », s'avéraient trop simples pour gérer le dialogue X.25 ; on insérait donc entre les ETTD et les ETCD des *PAD*[10], dont le rôle était de mettre les paquets en mémoire tampon, d'assembler et désassembler les paquets, ainsi que d'ajouter un en-tête ou de le supprimer avant de transmettre un paquet à l'ETTD ou l'ETCD.

Initiation des connexions

Une « session X.25 » est établie lorsqu'un ETTD en contacte un autre pour demander l'ouverture d'une « session de communication ». S'il accepte, le transfert d'information peut commencer. La connexion s'achève dès que l'un des ETTD le demande. Tout cela semble classique, si ce n'est que la norme X.25 permet d'acheminer des *données d'appel*[11] avant même que la connexion ne soit établie.

Il s'agit d'un bloc de données de seize octets, qui peuvent être adjoints au paquet de demande d'appel. Ce comportement n'admet aucune correspondance dans les spécifications de TCP/IP, les paquets de demande de connexion ne pouvant contenir aucune information concernant les couches supérieures à la couche de transport.

7. Packet-Switching Exchange
8. Ce n'est pas un hasard, les réseaux X.25 ayant été déployés à l'initiative des opérateurs de télécommunication, contrairement à IP.
9. Nous utilisons ce terme par analogie avec IP, mais il faut garder à l'esprit qu'il n'appartient pas officiellement à la terminologie de X.25.
10. Packet Assembler Disassembler
11. en anglais *Call User Data*, ou CUD

Dans la pratique, ces données ont pu être utilisées de deux manières :

- soit comme un mécanisme pour « router » les paquets vers telle ou telle machine, souvent à un niveau matériel. C'est, il me semble [12], l'usage le plus courant, et qui paraît même évident pour certains constructeurs, mais ce n'est *pas* l'utilisation prépondérante dans le secteur bancaire ;
- soit comme données *applicatives*, qui en l'occurrence sont tout à fait significatives pour un certain nombre de protocoles bancaires. Le mode RFC1086 décrit à la section 2.3.1 ne permet pas l'utilisation de ces derniers, et nous verrons qu'aucune solution de remplacement satisfaisante n'a pu être trouvée.

Circuits virtuels

Cette notion peut être mise en correspondance avec celle de connexion au sens TCP/IP. Un circuit virtuel est une connexion logique permettant une connexion fiable entre deux dispositifs réseau et peut traverser un certain nombre de nœuds intermédiaires (DCTE et PSE) ; ils sont démultiplexés à l'extrémité du réseau.

Structuration en couches

Bien que X.25 soit antérieur à OSI, on retrouve une structuration en *couches*, même s'il faut noter que le terme X.25 désigne l'ensemble de ces couches, et non pas seulement la couche réseau comme c'est le cas pour IP par exemple :

- la couche physique correspond à diverses interfaces série, entre lesquelles EIA/TIA-232, EIA-TIA-449, EIA-530 et G.703 ;
- la couche de liaison est généralement assurée par LAPB [13], ce protocole permet de garantir l'absence d'erreur dans les trames ainsi que leur bon ordonnancement ;
- la couche réseau, PLP [14], gère les échanges de paquets entre ETTD, à travers un *circuit virtuel*. Elle se charge de fragmenter et réassembler les paquets, et de gérer les erreurs.

Fiabilité du protocole X.25

Le protocole X.25 a été conçu pour fonctionner sur des liens peu sûrs. À l'instar de TCP, il comporte donc les primitives nécessaires pour s'assurer de l'intégrité des paquets transmis. Ce fait est pris en compte dans la RFC 1006 [**12**], dont il sera rapidement question à la section 2.2 lorsque nous entrerons dans le détail de la RFC 1086.

2.1.2 Solutions de conversion X.25/TCP

X.25, une technologie obsolescente

Les réseaux X.25 sont obsolescents, et de fait se sont déjà largement inclinés face à TCP/IP. Il faut certainement y voir plus que l'effet de facteurs

12. du moins c'est ce que j'en ai déduit à la lecture des manuels de certains dispositifs réseau, et après échange avec un ingénieur support de la société Funkwerk
13. Link Access Procedure, Balanced
14. Packet Layer Protocol

contingents. X.25 a été mis en place à la fin des années soixante-dix [15] et a connu un très grand succès dans certains États de l'Europe occidentale comme la France et l'Allemagne dans les années quatre-vingts, toutefois certaines de ses caractéristiques ne sont plus d'actualité. Il ne s'agit pas seulement de limitations technologiques : on sait que la norme limite les débits à deux mégabits par seconde, mais sur ce point, une évolution serait envisageable. Le nœud du problème tient à la conception même de ce réseau.

D'abord, il n'y a plus grand intérêt à faire communiquer des terminaux passifs, étant données les puissances de calcul disponibles de nos jours ; cela élimine l'intérêt des PAD. Au-delà, il est inutile de s'encombrer d'un cœur de réseau complexe qui se charge de vérifier l'intégrité des paquets et leur bon acheminement : TCP a démontré que ces opérations peuvent avantageusement être effectuées aux extrémités du réseau ; c'est là qu'est désormais concentrée l'intelligence, la bonne approche étant d'utiliser des protocoles simples mais efficaces pour interconnecter des machines et des réseaux, c'est ce qui a fait le succès d'Internet. De plus, les liens étant bien meilleurs, les mécanismes de correction de X.25 sont devenus superflus, et nuisent à son efficacité.

La tarification joue également un rôle : suivant les principes couramment appliqués par les opérateurs de télécommunications, l'utilisation de X.25 implique l'acquittement d'un abonnement *et* le paiement du temps de communication. Les opérateurs ne souhaitant pas maintenir indéfiniment ce réseau, les prix augmentent régulièrement et de manière conséquente, sans compter le coût du matériel, du fait de sa rareté.

Enfin, la division en couches de X.25 n'est pas aussi souple que celle de TCP/IP ; il reste difficile de les utiliser séparément.

La migration vers TCP/IP s'effectue peu à peu. Dans l'intervalle, il est nécessaire de pouvoir interconnecter les anciens réseaux aux nouveaux. Pour ce faire, un certain nombre de solutions existent ou sont envisageables.

Les « passerellistes »

Tout d'abord certaines sociétés proposent à leurs clients de se charger d'acheminer les transactions entre les deux types de réseaux. C'est le cas de Lyra Network [**1**], qui propose un certain nombre de solutions de migration vers IP. Un établissement possédant un parc de TPE* suffisamment important peut les paramétrer de manière à ce qu'ils appellent en X.25 Lyra, qui se chargera d'assurer la translation, en redirigeant les flux en TCP/IP vers le système informatique de cet établissement. Ce peut être une bonne solution pour un parc de TPE suffisamment grand, et pour un client ayant des moyens financiers suffisants. La figure 2.2 présente un aperçu de ce mode de gestion des flux X.25.

Utilisation de routeurs « mixtes » X.25 et TCP/IP

Pour des applications de taille relativement modeste, par exemple pour certaines applications privatives, ou lorsque le parc de TPE appelant en X.25 est relativement réduit, on peut préférer une autre solution, qui consiste à intercaler du matériel réseau dédié à la gestion des flux X.25.

On évite ainsi l'installation de matériel X.25 sur les serveurs. Non seulement cela simplifie leur mise en place et l'administration, mais surtout on s'affranchit

15. Le réseau français, Transpac, date de 1978.

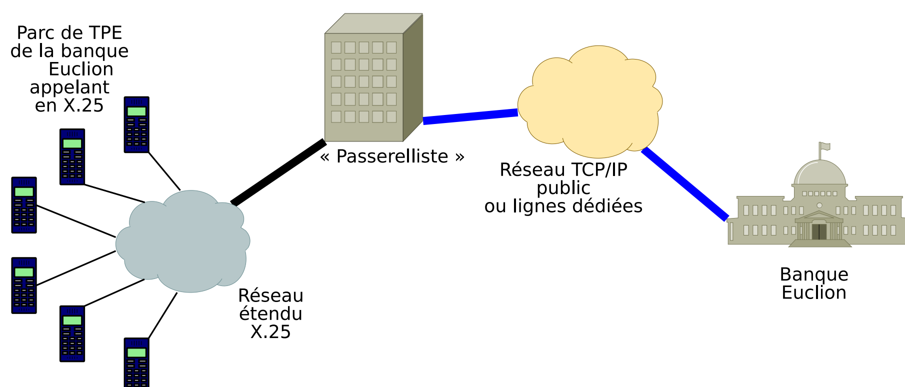


FIGURE 2.2 – Passerelliste effectuant une conversion entre réseaux X.25 et TCP/IP

d'un matériel en voie de raréfaction, coûteux, et qui dans la pratique gêne la virtualisation, sachant que de plus en plus de clients installent STAP dans une machine virtuelle de type VMWare ou Xen. De plus, à partir de ce routeur, on peut facilement, côté IP, rediriger vers les serveurs de son choix.

AFSOL souhaitait mettre en place un protocole normalisé et ouvert, avec lequel ses clients pourraient facilement s'interfacer, idéalement quel que soit le matériel choisi. La première partie du TFE proposait ainsi d'implémenter la RFC 1086, dont il est question dans la prochaine section.

2.2 Présentation de la RFC 1086

« RFC* » est l'abréviation de « Requests For Comment ». C'est ainsi que sont nommés les documents proposés officiellement par l'IETF* [16]. Si toutes les RFC ne sont pas des standards, il s'agit de documents ouverts, approuvés en principe par l'ensemble de la communauté gravitant autour d'Internet. Ce fonctionnement diffère de celui d'institutions comme l'ANSI [17] ou ISO* dont le premier fait partie. Les spécifications de l'IETF sont librement accessibles, sans avoir à s'acquitter de droits comme c'est le cas pour les publications d'ISO.

2.2.1 RFC 1086 et protocoles ISO

La RFC 1086 [**10**], écrite en 1988 et intitulée *ISO-TP0 bridge between TCP and X.25*, décrit le fonctionnement d'un pont [18] permettant à des machines X.25 et TCP/IP d'échanger des paquets.

Dans l'esprit de ses auteurs, cette RFC complétait la RFC 1006 [**12**], *ISO Transport Service on top of the TCP*, écrite l'année précédente. À cette époque, en effet, le modèle OSI [19], qui faisait l'objet de la publication de normes par ISO*, était en concurrence avec TCP/IP. OSI était plus théorique que TCP/IP

16. Internet Engineering Task Force
17. American National Standards Institute
18. que par la suite nous nommerons indifféremment « pont TP0 » ou « pont RFC 1086 ».
19. Open Systems Interconnection

et cherchait à décrire de manière stricte une structuration des protocoles en sept couches. L'implémentation de chacune d'entre elles prenait du temps, et il arrivait que les couches hautes arrivent à maturité *avant* les couches basses. En page quatre, la RFC 1006 part de ce constat pour proposer une solution permettant d'utiliser les couches hautes des protocoles ISO sur TCP.

Les auteurs proposent d'encapsuler TP0* dans TCP. TP [20] est le protocole de transport ISO, c'est donc l'homologue de TCP [21] ; il en existe cinq « classes », allant de TP0 à TP4. TCP étant déjà un protocole sûr, on décide de prendre la version de TP la plus faible, TP0. La RFC 1086 reprend cette idée d'encapsulation et propose que le pont décrit serve à l'échange de paquets TP0, à ceci près que si d'un côté du pont ceux-ci sont effectivement encapsulés dans TCP, de l'autre ils le sont dans X.25.

Huit ans plus tard, en 1996, paraît la dernière version des protocoles ISO ; ce sera la dernière. Avec l'explosion d'Internet, le succès de TCP/IP est flagrant, et il n'est désormais plus question de migration vers les couches ISO. De fait, l'utilisation que nous souhaitions faire de la RFC 1086 ne va pas tout à fait dans le sens que lui prêtaient ses auteurs, puisqu'il s'agit de migrer vers TCP/IP, et que seule nous intéresse la notion de translation entre les réseaux X.25 et TCP/IP.

Il reste quand même un reliquat de TP0 : il s'agit de l'en-tête, décrit en page quatorze de la RFC 1006 et qui comprend quatre octets. Le premier, toujours égal à trois, indique la version du protocole décrit dans cette RFC [22], le second est toujours à zéro, il s'agit d'une zone « réservée » [23]. Enfin, les deux derniers octets indiquent la taille du paquet en ordre réseau [24], *y compris* les quatre octets que nous venons de décrire. Ce format de paquet est illustré par la figure 2.3.

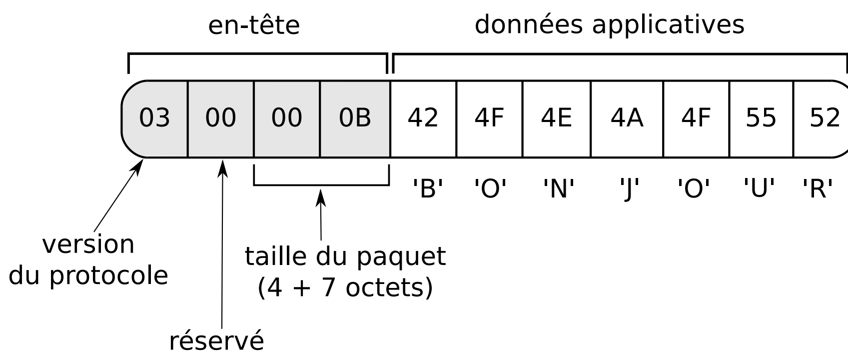


FIGURE 2.3 – Format de paquet défini par la RFC 1006. Chaque octet est noté en hexadécimal. Le message transporté est « BONJOUR » (codé en ASCII*), long de sept octets, ce qui donne une longueur de onze octets en comptant l'en-tête.

20. Transport Protocol
21. Transmission Control Protocol
22. et qui en fait n'a jamais évolué
23. et qui n'a jamais trouvé d'usage
24. l'octet de poids fort placé avant l'octet de poids faible

2.2.2 Protocole proposé par la RFC 1086

Nous venons de voir que la RFC 1086 permet l'échange de paquets TP0 entre les réseaux X.25 et TCP/IP, sachant que de TP0 il ne reste plus qu'un en-tête de quatre octets. Deux cas sont abordés, selon que l'hôte TCP/IP est client ou serveur, mais dans ces deux éventualités, c'est toujours l'hôte TCP/IP qui est responsable de l'initialisation du dialogue.

En effet, avant tout échange de paquets entre les deux réseaux, l'hôte TCP doit procéder à un *enregistrement* sur le pont TP0. Cela consiste à envoyer sur le port 146 du pont TP0 un certain nombre d'informations nécessaires pour le transfert des paquets :

- un octet indiquant si la machine TCP/IP est client ou serveur ;
- la sous-adresse X.25, correspondant à l'adresse à laquelle un serveur TCP veut être appelé, ou celle qu'un client TCP/IP veut appeler ;
- quelques autres informations concernant X.25 qu'il n'est pas utile de décrire [25]
- l'adresse IP et le port TCP de la machine TCP/IP. Ce peut être une autre machine que celle effectuant l'enregistrement.

Les figures 2.4 et 2.5 illustrent ce dialogue. Si l'enregistrement échoue, le pont coupe la connexion établie sur le port 146. Dans le cas contraire, on sait que l'enregistrement a réussi. Dès lors, si l'hôte TCP s'est enregistré en tant que serveur (c'est le cas sur la figure 2.5), toutes les demandes de connexions sur la sous-adresse fournie lors de l'enregistrement (44 sur la figure) seront reportées côté TCP par le pont, les paquets ensuite échangés seront transmis par l'intermédiaire du pont TP0, qui rajoutera l'en-tête dont il a été question à la section 2.2.1 avant de les confier à l'hôte TCP, et le supprimera dans l'autre sens.

Cette conversion a lieu tant que l'hôte TCP maintient sur le pont TP0 la connexion sur le port 146 qui, rappelons-le, est celui qui a été utilisé lors de la phase d'enregistrement. La déconnexion entraîne la coupure des connexions en cours entre les hôtes X.25 et TCP. Pour communiquer à nouveau, il faut répéter à partir de la phase d'enregistrement.

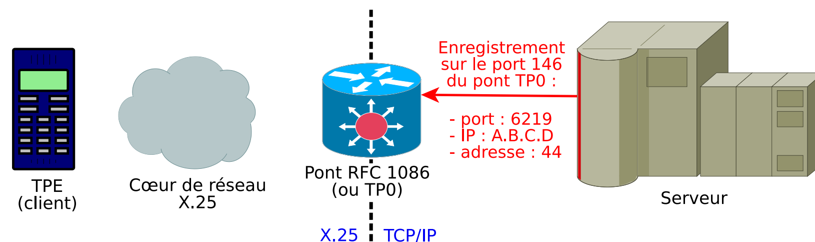


FIGURE 2.4 – Procédure d'enregistrement du serveur TCP/IP sur un pont TP0. Le serveur déclare son port d'écoute TCP (6219), son IP ainsi que la sous-adresse X.121 correspondant aux appels qui lui sont destinés (44).

25. en pratique, ils sont tous positionnés à zéro, sauf le champ « Protocol Id », qui est à 01000000 (en hexadécimal), qui fait passer pour un PAD le pont TP0 placé entre les hôtes TCP et X.25

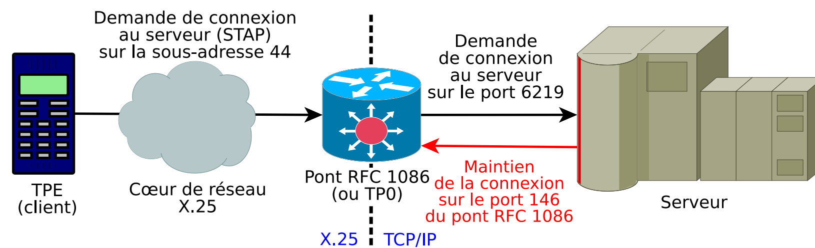


FIGURE 2.5 – Transfert de connexion depuis une sous-adresse X.121 vers un port TCP par un pont TP0

2.3 Implémentation de la RFC 1086 dans la bibliothèque d'isolation réseau d'AFSOL

Deux modes correspondant à cette RFC ont été implémentés dans STAP. Le premier, RFC1086, correspond au cas illustré par la figure 2.5. L'hôte TCP/IP, sur lequel est installé STAP, s'enregistre en tant que serveur sur le pont TP0, qui se charge par la suite de lui transférer les paquets envoyés par les TPE X.25 (les clients), tant que la connexion sur le port 146 est maintenue.

Dans le second, EMULRFC1086, illustré par la figure 2.7, STAP (toujours côté IP) se fait passer pour un pont TP0, et accepte de la part d'un TPE IP*, qui croit communiquer avec une machine X.25 à travers un pont TP0, une demande d'enregistrement en tant que client, puis les informations que ce dernier lui fait parvenir. Il se trouve en effet que les TPE IP, qui commencent à peine à être commercialisés, utilisent souvent le dialogue RFC 1086, semble-t-il pour rester compatibles avec les serveurs de télécollecte qui continueraient à n'utiliser que X.25, mais aussi parce que la procédure d'enregistrement permet de communiquer certaines informations qui peuvent être significatives.

2.3.1 Mode RFC1086

On peut distinguer deux parties dans l'implémentation de ce mode : la phase d'enregistrement, se prolongeant par le maintien de la connexion, et la gestion du format des paquets TP0.

Phase d'enregistrement et surveillance de la connexion sur le port 146

L'utilisation particulière du port 146 que fait la RFC 1086 impose le maintien de la connexion après la phase d'enregistrement. Pour ce faire, nous avons utilisé une boucle infinie, et pour cette raison il était plus aisé d'écrire un exécutable [26] effectuant ces opérations que des fonctions. Cette boucle infinie se charge de vérifier périodiquement l'état de la connexion ; si elle n'est plus en activité, on attend une trentaine de secondes afin d'éviter de monopoliser trop de ressources, et on essaie de s'enregistrer à nouveau. La figure 2.6 illustre ce comportement.

26. chargé au démarrage de STAP grâce aux fonctions C `fork` et `exec`

La difficulté technique consistait à vérifier l'état d'une connexion TCP entre deux hôtes. En effet, une fois établie, les hôtes n'effectuent aucun échange de signaux TCP destinés au maintien de la connexion ; si aucune donnée applicative n'est envoyée par une des parties, aucun paquet ne circulera, et pourtant la connexion sera toujours considérée comme active des deux côtés, jusqu'à ce qu'on atteigne un « timeout* » qui est en général de l'ordre de deux heures.

Il reste toutefois possible d'activer l'option « TCP Keepalive » [**7**] sur le socket* utilisé grâce à la fonction C `setsockopt`. Cette option force l'envoi régulier d'acquittements ou sondes (paquets TCP « ACK ») et nécessite de préciser trois paramètres :

- `tcp_keepalive_time` : l'intervalle de temps (en secondes) entre le dernier paquet de données envoyé et la première sonde ;
- `tcp_keepalive_intvl` : l'intervalle de temps (en secondes) à respecter avant l'envoi d'une nouvelle sonde ;
- `tcp_keepalive_probes` : le nombre de sondes à envoyer avant de considérer que la connexion n'est plus active.

Lorsque la dernière sonde aura été renvoyée sans succès, la fonction C `read` renvoie alors une erreur, on essaie alors de répéter la phase d'enregistrement.

Le programme ne peut être dévié de cette boucle infinie que par réception d'un signal, positionné sur une fonction `FermerTP0Bridge` qui cherche à fermer proprement la connexion.

En plus de ce mécanisme, l'exécutable est lui-même surveillé par une fonction préexistante de STAP, qui se réveille à intervalles réguliers de quelques minutes. Celle-ci est chargée de vérifier les numéros de processus (PID, « Process Identifier ») et de recharger le processus s'il n'est plus en mémoire (il peut avoir été tué par le système ou par une fausse manœuvre de l'utilisateur).

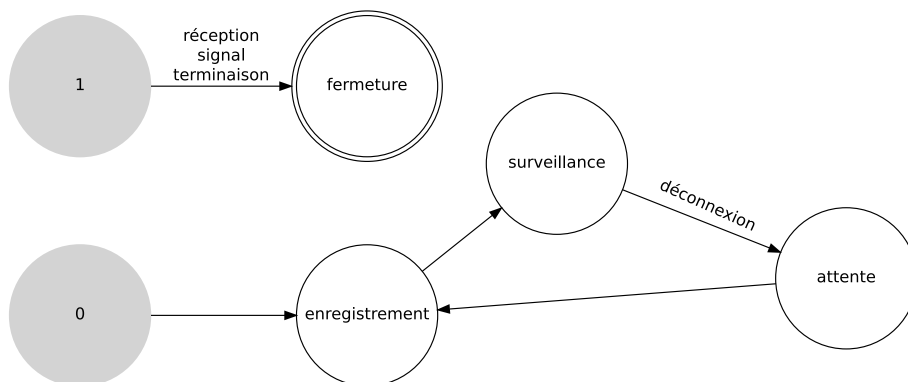


Figure 2.6 – Description sous forme d'automate fini de l'exécutable chargé de surveiller la connexion avec le pont TP0

Gestion des paquets TP0

Comme il a été dit à la section 2.2.1, lorsque le pont TP0 reçoit des données de l'hôte X.25, il rajoute un en-tête de quatre octets. De même, l'hôte TCP doit rajouter cet en-tête aux données qu'il envoie au pont, ce dernier le supprimera

avant de les transférer à l'hôte X.25. Le code écrit dans la bibliothèque d'isolation réseau tient compte de cette particularité.

Ce format étant propre aux données échangées en RFC 1086, il est impossible de le faire coexister avec d'autres modes sur le même port. Il se trouve qu'au moment où j'ai implémenté ce protocole, STAP n'était capable d'écouter que sur un seul port, or l'intérêt de la RFC est de pouvoir télécollecter quelques TPE X.25, à côté de flux IP plus importants. Une modification de la couche réseau, permettant d'écouter sur plusieurs ports en utilisant divers protocoles, était donc nécessaire. Elle était en cours de réalisation au moment où j'ai quitté AFSOL.

Données d'appel

En travaillant sur la RFC 1086, nous avons rencontré un problème qui n'avait pas été soupçonné pendant la phase d'étude, ne connaissant pas tous les arcanes de X.25. Il concerne les données d'appel, dont il a été question section 2.1.1. Rappelons qu'il s'agit de seize bits de données transmis dans le paquet de demande de connexion, qui en l'occurrence sont des données applicatives, cruciales pour certains protocoles bancaires.

D'où vient le problème ? Lorsque la phase d'enregistrement s'est correctement déroulée, le pont TP0 acceptera de la part des hôtes X.25 les demandes d'ouverture de session sur la sous-adresse déclarée par la machine TCP/IP. À ce moment-là, il ouvrira à son tour une connexion sur le serveur TCP, en utilisant l'adresse et le port indiqués dans le paquet d'enregistrement. TCP/IP ne permet pas de rajouter des données dans un paquet de demande de connexion. Les données d'appel sont perdues.

Pour accepter ce comportement, il faudrait que le pont TP0 envoie un paquet de données contenant les données d'appel juste après s'être connecté sur l'hôte TCP.

Solutions de remplacement

Les tests ont été effectués sur un routeur R4300 mis à disposition par la société Funkwerk (anciennement Bintec). Le développement a pu être validé assez rapidement, mais les problèmes liés aux données d'appel subsistaient. Nous sommes resté assez longuement en liaison avec un ingénieur support, mais il est apparu que Funkwerk n'avait pas prévu ce cas d'utilisation et que rien ne pouvait être fait pour récupérer les données d'appel. Il est vrai que la RFC ne les mentionne pas, on ne peut donc pas objectivement blâmer Funkwerk de ne pas avoir implémenté cette fonctionnalité.

Il n'a pas été possible de trouver d'autre dispositifs matériels intégrant la RFC 1086. Les fabricants de matériel X.25 ne courent pas les rues, et ceux que nous avons contactés ne l'implémentaient pas.

Nous avons alors cherché à savoir si un autre protocole pouvait convenir. Funkwerk ne proposait que d'autres formats de paquets, ce qui ne nous intéressait pas, et XoT [**9**]. Ce protocole consiste à encapsuler des paquets X.25 dans TCP. Cela ne nous intéressait pas non plus. Nous cherchions à nous *défaire* de X.25, or ce protocole le ramène sous une forme encapsulée, de surcroît beaucoup trop complexe à gérer !

Jonathan Goodchild, ingénieur chez Farsite, fabricant britannique de matériel réseau, nous a affirmé être en mesure de nous proposer un protocole prenant en compte les données d'appel. Malheureusement, il s'agit d'un protocole propriétaire, propre à Farsite, ce qui ne convenait pas à AFSOL, qui, comme nous l'avons précisé section 2.1.2, cherchait à mettre en place un protocole ouvert.

À l'heure actuelle, le protocole intégré à STAP ne peut pas être utilisé pour les versions de CB2A strictement inférieures à 1.2, qui correspond à une grosse partie du parc. Cependant, il fonctionne avec CBPR, et peut donc par exemple être utilisé pour des applications privatives, et avec CB2A 1.2, qui est la version en cours de mise en production en ce début d'année 2009.

2.3.2 Mode EMULRFC1086

Implémentation

Le mode EMULRFC1086, illustré par la figure 2.7, permet de faire passer un serveur STAP pour un pont TP0 auprès d'un TPE IP utilisant la RFC 1086. Cela consiste à accepter le paquet d'enregistrement lorsque le TPE initie la connexion, puis à accepter des paquets contenant l'en-tête décrit à la section 2.2.1, et à adjoindre ce même en-tête aux paquets retournés. Dans ce cas, l'enregistrement ne se fait pas sur le port 146, mais sur le port sur lequel STAP accepte habituellement les données de télécollecte.

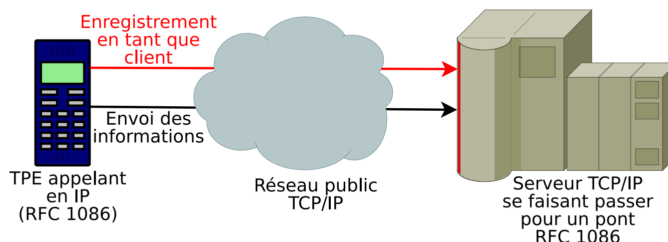


Figure 2.7 – TPE IP communiquant selon la RFC 1086 avec un serveur TCP/IP émulant le comportement d'un pont RFC 1086

Les données de la RFC 1086 de base contiennent notamment le numéro X.121 appelé, purement virtuel ici puisque tout est en IP, mais que STAP utilise pour savoir quel est le protocole bancaire utilisé, grâce à une table de « routage » interne, ainsi que l'adresse appelante. Ces données caractérisent une remise*.

Le GIE Cartes Bancaires a émis une version modifiée de la RFC 1086, dénommée RFC 1086 « Concert ». Il est important de noter que cet usage est peu orthodoxe et qu'il vaut mieux utiliser des protocoles normalisés chaque fois que cela est possible ! De plus, l'accès à ces spécifications, fort mal écrites au demeurant [27], est payant, ce qui est en totale contradiction avec le principe des RFC. Les modifications consistent simplement en l'envoi de la part du TPE d'une trame d'enregistrement spécifique avant la trame d'enregistrement « officielle ». Ces informations visent à identifier le TPE et permettent de savoir si le règlement de l'abonnement correspondant est à jour. Ces données ne sont

27. Une spécification digne de ce nom ne s'achève pas par l'expression « et cætera ».

d'aucune utilité à AFSOL, néanmoins les deux modes (RFC 1086 pure et sa variante « Concert ») sont fonctionnels.

Protection des flux IP

Dans ce mode, l'information transite entièrement sur un réseau TCP/IP. Ce réseau est *public*, ne serait-ce que parce qu'elle emprunte le réseau du fournisseur d'accès à Internet de l'accepteur avant d'être routé vers le serveur de télécollecte. Cela pose d'importants problèmes de sécurité.

Tant que les transactions passaient par des réseaux X.25, aucune mesure de protection particulière n'était préconisée. Une attaque de type « homme du milieu », ou « man in the middle » en anglais, c'est-à-dire une captation des flux par un intermédiaire est donc possible, du moins d'un point de vue théorique. On se retranche derrière le fait que le piratage sur les réseaux X.25 est peu répandu ; on peut aussi soupçonner que les établissements bancaires pensent que le secret qu'ils imposent sur les protocoles bancaires les protègent, ce qui est évidemment faux.

Quoi qu'il en soit, il en va tout autrement sur les réseaux TCP/IP publics. Le piratage y est courant, et l'interception de données beaucoup plus aisée. Le GIE Cartes Bancaires recommande donc la protection des remises transitant par ces réseaux grâce à une couche cryptographique nommée SSL*. Ce sera l'objet du chapitre suivant.

2.4 Bilan

La RFC 1086 a pu faire l'objet d'une implémentation complète et fonctionnelle dans STAP. Le mode RFC1086 pose cependant problème, aucun mécanisme n'étant envisageable pour récupérer les données d'appel. Le mode EMUL-RFC1086 gagnerait à être protégé par une couche cryptographique, afin d'éviter un accès frauduleux aux données transportées.

Chapitre 3

SSL

Le mode EMULRFC1086 introduit au chapitre précédent suppose la circulation de données sur un réseau TCP/IP public, et doit donc être protégé de l'utilisation frauduleuse des données transportées, au moyen de SSL* selon les recommandations du GIE Cartes Bancaires. Son apparition au cours de la phase d'étude ainsi qu'au début de l'implémentation a introduit une troisième partie dans notre TFE. De fait, il s'est vite avéré que cette technologie ne pouvait être mise en œuvre sans un minimum de connaissances de base, ce « minimum » renvoyant à un ensemble relativement complexe de concepts. Il nous a donc été demandé dans un premier temps de mener une recherche documentaire et de transmettre nos résultats à nos collègues à l'occasion d'une présentation. Il est d'emblée ressorti que l'implémentation de SSL dans les applications d'AFSOL était un projet à part entière, et qu'il n'était pas question d'aboutir sur un produit fini.

Dans ce chapitre, nous nous attacherons à décrire le protocole SSL, allant beaucoup plus loin dans la présentation des concepts fondamentaux que dans les autres parties. Nous considérons en effet qu'il s'agit d'une partie importante de notre travail au sein d'AFSOL, sur laquelle reposeront les développements futurs. Ensuite sont présentées les premières réalisations, ainsi que certaines considérations concernant les choix de matériel et de logiciel.

3.1 Généralités sur SSL

SSL, *Secure Sockets Layer*, permet de sécuriser les échanges de données sur Internet (sur un réseau TCP/IP) entre un client et un serveur, et répond aux quatre objectifs principaux d'un système cryptographique : assurer la *confidentialité* des données, en utilisant des algorithmes de chiffrement qui les rendent inintelligibles ; veiller à l'*intégrité* des données ; *authentifier* les hôtes participant au dialogue SSL ; garantir la *non-répudiation* des actions d'une ou des parties, c'est-à-dire qu'il leur soit impossible de les nier.

3.1.1 Notions cryptographiques de base

Cette section présente les notions de bases de la cryptographie nécessaires pour aborder SSL. Pour de plus amples informations, on pourra se reporter au

premier chapitre de [**19**], et au reste du livre si l'on souhaite aborder la cryptographie de manière plus extensive. En ce qui concerne SSL, nous recommandons vivement [**14**], malheureusement épuisé suite à la fermeture d'O'Reilly France, mais qui est présent dans de nombreuses bibliothèques universitaires et que l'on doit encore pouvoir acquérir sous forme électronique. [**15**] nous a également été utile, et il peut intéresser ceux qui souhaitent de surcroît se lancer dans la programmation avec la bibliothèque OpenSSL. Nous citons également [**20**], que nous n'avons pas eu entre les mains mais qui semble avoir eu un certain succès.

Chiffrement et déchiffrement des messages

On souhaite transformer un message « en clair » M en un message inintelligible, « chiffré » C. Dans ce but, on se donne

- une fonction de chiffrement E, telle que $C = E(M)$
- une fonction de déchiffrement D, telle que $M = D(C)$
- l'identité suivante doit donc être vérifiée : $D \circ E = id$

La cryptographie moderne fait usage de *clefs cryptographiques*. Une clef K est une information secrète, au moins en partie, appartenant à un ensemble $\mathcal{K}$ (l'*espace des clefs*), de cardinalité importante. Le couple (a, K) formé par un *algorithme de chiffrement* et une clef correspond à une fonction de chiffrement ou de déchiffrement.

On note K_c et K_d les clefs de chiffrement et de déchiffrement.

Chiffrement symétrique Dans ce mode ce chiffrement, la même clef est utilisée pour le chiffrement et le déchiffrement ($K_c = K_d$). La cryptographie symétrique est utilisée depuis l'Antiquité, l'exemple le plus connu étant le chiffre de César. Elle possède un gros inconvénient, qui rend impossible son utilisation à grande échelle. En effet, avant de communiquer de manière secrète, les deux correspondants doivent convenir de la clef. Difficile dans ces conditions pour un acheteur européen de communiquer de manière sûre son numéro de carte à un commerçant californien...

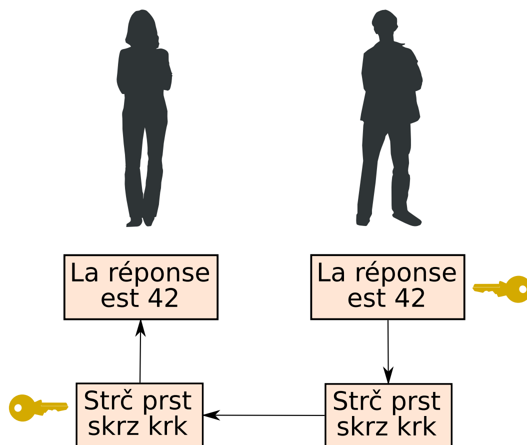


FIGURE 3.1 – Échange d'un message confidentiel utilisant la cryptographie symétrique

Néanmoins, les calculs sont rapides (beaucoup plus que les calculs de cryptographie asymétrique, voir ci-dessous), et la confidentialité est garantie du moment que les clefs sont d'une longueur suffisante (de manière à ce que la cardinalité de $\mathcal{K}$ soit suffisamment importante). Elle permet une forme basique d'authentification : si le message reçu a un sens une fois déchiffré, on identifie son émetteur comme le porteur de la clef. Nous verrons que ce mode de chiffrement est celui utilisé au sein d'une *session* SSL.

Ce mode de chiffrement est illustré par la figure 3.1. Supposons que Bob (à droite sur la figure), après sept millions et demi d'années de temps de calcul, ait trouvé un résultat important dans le cadre de ses recherches, et qu'il souhaite le communiquer à Alice (à gauche). Au préalable, Alice et Bob ont convenu d'une même clef cryptographique. Bob l'utilise pour chiffrer son résultat, qui apparaît alors sous une forme inintelligible et inexploitable pour qui ne possède pas la clef. Alice reçoit le message chiffré, et le déchiffre avec la même clef pour prendre connaissance de la note envoyée par Bob.

La figure 3.2 recense quelques algorithmes symétriques d'utilisation courante.

algorithme	**longueur de clef**	**commentaire**
DES[a]	56 bits	« craqué » en 1999 en 22h15min
AES[b]	128, 192 et 256 bits	remplaçant de DES
3DES[c]	112 et 168 bits	construit sur DES
IDEA[d]	128 bits	breveté par Mediacrypt
Blowfish	32 à 448 bits	peut remplacer DES, pas de brevet
RC5[e]	jusqu'à 2048 bits	breveté par RSA Security

[a] Data Encryption Standard
[b] Advanced Encryption Standard
[c] Triple DES
[d] International Data Encryption Algorithm
[e] Ron's Code ou Rivest's Cipher

FIGURE 3.2 – Quelques algorithmes symétriques ou à clef secrète (par blocs)

Chiffrement asymétrique Un algorithme de chiffrement est dit asymétrique lorsque la clef de chiffrement et la clef de déchiffrement sont distinctes ($K_c \neq K_d$). K_c est nommée *clef publique*, K_d *clef privée*. K_c peut être diffusée librement, tandis que K_d doit absolument être gardée secrète par son possesseur. Supposons qu'Alice ait généré une telle paire de clefs (un *biclef*). Tout le monde connaît K_c, et peut donc envoyer un message chiffré à Alice, qui sera la seule à pouvoir le déchiffrer avec K_d.

De la même manière, si Alice veut envoyer un message à Bob, elle utilisera la clef publique de ce dernier. Bob déchiffrera le message avec sa propre clef privée.

La figure 3.3 reprend la situation précédente, mais évidemment la situation est légèrement plus complexe. Bob souhaite transmettre le même message, en faisant appel à la cryptographie asymétrique. Alice et Bob n'ont pas eu besoin de se rencontrer. Alice a généré par ses propres moyens un biclef : la clef privée à gauche, la clef publique à droite ; toutes deux portent la lettre « A » pour bien indiquer qu'elles appartiennent à Alice. Bob peut récupérer la clef publique d'Alice sur un réseau public. À ce stade, il ne peut pas être sûr du propriétaire, mais ce problème est résolu par la mise en place de tiers de confiance, dont il

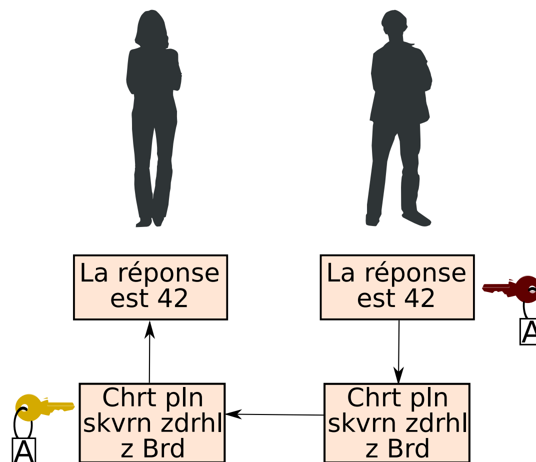


FIGURE 3.3 – Échange d'un message confidentiel utilisant la cryptographie asymétrique

est question section 3.1.2. Bob chiffre le message avec la clef publique d'Alice, cette dernière le déchiffre avec sa clef privée, et est donc la seule en mesure d'en prendre connaissance.

Un biclef peut aussi être utilisé dans l'autre sens. Alice peut chiffrer un message avec sa clef privée ; tout le monde peut alors le lire, mais le fait d'y parvenir prouve qu'Alice est bien l'auteur du message, puisqu'elle est la seule à détenir la clef privée. Nous y reviendrons lorsque nous parlerons d'authentification et de signature. Sur la figure 3.5 Alice chiffre un message avec sa clef privée, ce qu'elle est seule à pouvoir faire. Bob le déchiffre avec la clef publique d'Alice, ce que tout le monde peut faire ; néanmoins, s'il compare le résultat obtenu avec une copie qu'Alice lui aurait envoyée en clair, et qu'il en déduit que les deux messages sont identiques, il peut être certain que le message a été envoyé par Alice.

La cryptographie asymétrique, également désigné par le terme *cryptographie à clef publique*, apporte une solution satisfaisante au problème de l'échange des clefs. Néanmoins, elle est environ mille fois plus coûteuse en calculs que la cryptographie symétrique. Nous verrons que SSL allie les deux méthodes.

On utilise ce mode pour chiffrer des messages courts, pour échanger des clefs symétriques, et pour s'authentifier ou signer des messages ou des documents.

La figure 3.4 présente quelques algorithmes asymétriques parmi les plus cou-

algorithme	**longueur de clef**	**commentaire**
RSA[a]	supérieure à 1024 bits	le brevet a expiré en 2000
DSA[b]	généralement 1024 bits	uniquement pour la signature
Diffie-Hellman	supérieure à 768 bits	mécanisme d'échange de clefs

[a] Rivest, Shamir, Adleman [b] Digital Signature algorithm

FIGURE 3.4 – Quelques algorithmes asymétriques

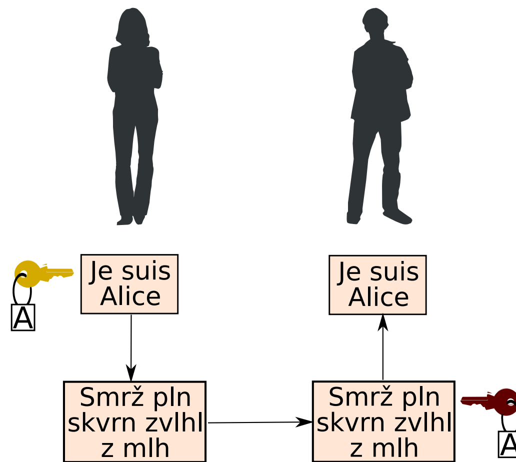


FIGURE 3.5 – Utilisation de la cryptographie asymétrique pour signer un message

ramment utilisés. Ceux-ci reposent sur l'explosion combinatoire engendrée par certains problèmes d'arithmétique : décomposition en facteurs premiers pour RSA, logarithme discret pour Diffie-Hellman. On constatera que les clefs sont beaucoup plus longues que pour les algorithmes symétriques. Les longueurs citées correspondent à un niveau de sécurité jugée acceptable ; il ne faut pas perdre de vue que, la puissance de calcul augmentant, elles doivent régulièrement être revues à la hausse.

Fonction de hachage, empreinte numérique et signature

Une fonction de hachage est une fonction f qui convertit une suite de bits de taille quelconque en une suite de bits de taille fixe. Elle est dite *à sens unique* lorsqu'il est très difficile de trouver $x_1 \neq x_2$ vérifiant $f(x_1) = f(x_2)$ (ce que l'on appelle une *collision*). Le résultat de f est alors nommé *empreinte numérique*.

En pratique, si l'on change un seul bit de l'information en entrée, le résultat est drastiquement différent. Une empreinte numérique de 160 bits est un minimum, elle correspond à l'algorithme SHA-1. À l'heure actuelle, on conseille plutôt de recourir à SHA-256, qui génère des empreintes numériques de 256 bits. MD5 ne doit plus être utilisé, il n'est plus considéré comme sûr.

La cryptographie asymétrique et les empreintes numériques permettent de générer des signatures numériques. Le principe consiste à envoyer à son correspondant un message en clair M, et le même message chiffré avec sa clef privée $S = E(M)$, appelé *signature*. Le destinataire du message calcule $M' = D(S)$ avec la clef publique. Si $M = M'$, la signature est vérifiée.

En pratique, on accompagne le message en clair d'un *code d'identification de message* ou *MAC* (de l'anglais « Message Authentication Code »), c'est-à-dire l'empreinte numérique chiffrée avec la clef privée. Le code d'identification étant chiffré avec la clef privée, on est certain de l'identité de son expéditeur. La vérification s'effectue en déchiffrant le MAC avec la clef publique pour obtenir l'empreinte numérique M' calculée par l'expéditeur d'une part, et en calculant

soi-même l'empreinte numérique, M ; on s'assure alors que M et M' sont égaux.

Utilisation d'algorithmes cryptographiques

Il est bon d'observer quelques règles lorsqu'il s'agit de bâtir ou de mettre en œuvre un système cryptographique. Tout d'abord, il ne faut jamais utiliser de méthodes « maison », mais des algorithmes publics, éprouvés par les cryptanalistes. Il est par exemple exclu de bricoler soi-même un algorithme de chiffrement, ou de procéder par obfuscation*, le niveau de sécurité obtenu serait à peu près nul.

Un soin tout particulier doit être accordé à la génération des clefs. La source d'entropie* utilisée doit être de qualité suffisante, préférer les interfaces telles que `/dev/urandom` sous les systèmes de type Unix. On peut faire confiance aux programmes tels que OpenSSL. Un bon compromis doit être établi en ce qui concerne la longueur des clefs entre le niveau de sécurité correspondant d'une part, et le temps de calcul et la charge induite par les calculs cryptographiques d'autre part.

Il convient enfin de conserver jalousement les clefs ou privées, et de maintenir les systèmes à jour, de manière à empêcher toute attaque qui s'appuierait sur des failles connues.

En France, l'utilisation de la cryptographie est libre, comme le rappelle [**16**] : « En vertu de l'article 30-l de la loi 2004-575 du 21 juin 2004, l'utilisation de moyens de cryptologie est libre ». Il n'en va pas forcément de même pour l'importation et l'exportation d'algorithmes de cryptographie : « En revanche, la fourniture, l'importation et l'exportation des ces moyens sont réglementés en France. Ces opérations sont soumises soit au régime de la déclaration, soit au régime de l'autorisation. »

3.1.2 Certificats X.509

Un certificat* est le support de la clef publique d'une entité (que nous appellerons Alice). Il sert également de carte d'identité, et contient certaines informations comme le nom de l'organisation et la date de fin de validité. Ainsi, lorsque l'on désire écrire un message confidentiel à Alice, il suffit de récupérer son certificat et d'utiliser la clef publique qui y est mentionnée. Comment cependant être certain de l'authenticité du certificat ? Quelle garantie avons-nous que cette clef n'appartient pas à une tierce personne qui chercherait à usurper l'identité d'Alice ?

La norme X.509 [**18**] répond à ce problème en se basant sur un nombre réduit de tiers de confiance, nommés *autorités de certification* (AC). Une AC assure qu'un certificat est valide en le signant (voir section précédente). Afin d'obtenir un tel certificat signé, il faut suivre une procédure particulière.

Tout d'abord, Alice doit générer un biclef, dont elle gardera de côté la clef privée ; en toutes circonstances, Alice reste la seule à connaître sa clef privée [1]. Alice génère un certificat en respectant la norme X.509, dans lequel elle place

1. cette règle peut être mise en défaut lorsque les certificats sont mis en œuvre de manière centralisée ; dans ce cas, il existe un service particulier (par exemple au sein d'une entreprise), qui se charge de générer des biclefs et des certificats pour ses utilisateurs, et parfois de garder la clef privée afin de générer un certificat identique si le précédent a été détruit accidentellement, ou perdu sans être compromis.

```
Certificate ::= SEQUENCE {
       tbsCertificate       TBSCertificate,
       signatureAlgorithm   AlgorithmIdentifier,
       signatureValue       BIT STRING }

  TBSCertificate ::= SEQUENCE {
       version         [0]  EXPLICIT Version DEFAULT v1,
       serialNumber         CertificateSerialNumber,
       signature            AlgorithmIdentifier,
       issuer               Name,
       validity             Validity,
       subject              Name,
       subjectPublicKeyInfo SubjectPublicKeyInfo,
       issuerUniqueID  [1]  IMPLICIT UniqueIdentifier OPTIONAL,
                            -- If present, version MUST be v2 or v3
       subjectUniqueID [2]  IMPLICIT UniqueIdentifier OPTIONAL,
                            -- If present, version MUST be v2 or v3
       extensions      [3]  EXPLICIT Extensions OPTIONAL
                            -- If present, version MUST be v3
       }
```

Figure 3.6 – Description ASN.1 de la partie principale d'un certificat X.509 extraite de la RFC 3280

la clef publique qu'elle vient de générer, et d'autres informations comme son nom, la durée de validité souhaitée, et certaines informations sur les algorithmes cryptographiques, sur lesquels nous reviendrons plus tard. Ce certificat non signé est appelé *demande de signature de certificat*. Pour qu'il soit fiable, il faut qu'il soit transmis à une AC, qui se chargera de vérifier l'identité d'Alice [2]. Si elle juge la demande de signature légitime, l'AC signe le certificat avec sa propre clef privée. Tout le système repose d'une part sur la confiance que l'on porte aux AC ; ce point est important, car certaines pourraient ne pas faire preuve de suffisamment de sérieux lors de la vérification de l'identité de leurs clients. D'autre part, le mécanisme tout entier suppose la non compromission de la clef privée de l'AC ! Dans le cas contraire, n'importe qui peut signer un certificat avec cette clef et usurper le capital de confiance de l'AC ; de surcroît, un pirate qui dans le passé aurait enregistré des communications SSL pourrait les *rejouer* et les déchiffrer.

La figure 3.6 reprend la description ASN.1 [3] de la partie principale d'un certificat X.509 telle que présentée dans la RFC* 3280 [**17**]. Les quatre premières lignes indiquent qu'un certificat est composé d'un champ renseignant l'algorithme utilisé par l'AC pour signer (`signatureAlgorithm`), d'un autre

2. Il existe plusieurs niveaux de sécurité, allant de la vérification d'un nom de domaine ou d'une adresse de courrier électronique, à la confrontation physique.

3. ASN.1 (Notation de Syntaxe Abstraite), est un standard émis par l'Union Internationale des Télécommunications [**39**] ; c'est une notation formelle permettant de décrire le format de messages.

contenant la signature de cette AC (`signatureValue`), et d'un TBS (« To Be Signed ») qui contient toutes les autres informations. Lors de sa demande de signature de certificat, Alice ne remplira que le TBS. Son AC en calculera l'empreinte numérique, qu'elle chiffrera avec propre sa clef privée, et placera le résultat dans le champ `signatureValue`. Elle renseignera l'algorithme utilisé dans `signatureAlgorithm`.

Champ d'utilisation d'un certificat

Les extensions de SSLv3 sont des champs supplémentaires permettent de destiner le certificat à un usage particulier. Rappelons que ces champs ne peuvent être modifiés, puisqu'un certificat signé ne peut être contrefait sans connaître la clef privée de l'autorité de certification.

À titre d'exemple on peut citer les extensions suivantes :

- avec *keyUsage*, on peut restreindre l'utilisation de la clef privée de l'utilisateur, de manière à ce qu'elle ne puisse servir que pour signer un document ou un courrier électronique (valeurs digitalSignature et nonRepudiation) ;
- avec *nsCertType*, on peut émettre un certificat qui ne pourra servir que pour un serveur(valeur server) ou pour un client de courrier électronique (clientAuth et emailProtection).

Nous allons voir qu'elles peuvent également être utilisées pour déléguer tout ou partie des pouvoirs de l'autorité de certification.

Chaîne de certificats

Un certificat signé par une AC « mère » peut à son tour signer un autre certificat. On parle alors de « chaîne de certificats ». Pour contrôler cette chaîne, on utilise l'extension `basicConstraints`, qui a pour valeur `CA:false` lorsqu'on interdit au certificat généré de signer un certificat, ou CA:true sinon. L'extension `keyCertSign` permet de limiter la longueur de la chaîne.

On voit de la sorte qu'un AC mère peut avoir plusieurs AC filles. Une AC peut décider de créer une AC mère qui ne servira que pour générer des AC spécialisés, l'une signant des certificats clients pour le courrier électronique, l'autre des certificats pour les serveurs, etc.

Ceci est précisé par la figure 3.7. L'autorité de certification mère délègue ses pouvoirs à trois autorités de certification filles spécialisées : l'une émet des certificats pour les clients de courrier électronique, l'autre pour les navigateurs web, la dernière pour des serveurs. Chacune de ces AC filles utilise sa clef privée pour signer des certificats, qui ne pourront être utilisés que dans le domaine correspondant. L'AC mère se borne alors à signer les certificats des AC filles. Pour valider le certificat d'un utilisateur, par exemple un certificat pour un navigateur web, à supposer que l'on fasse confiance à l'AC mère, il faut d'abord s'assurer que le certificat a été correctement signé par l'AC déléguée au courrier électronique, puis que le certificat de cette AC a bien été authentifié par l'AC mère.

Validation d'un certificat

Quelles sont les opérations à effectuer pour s'assurer de la validité d'un certificat ? Dans le cas d'un certificat serveur :

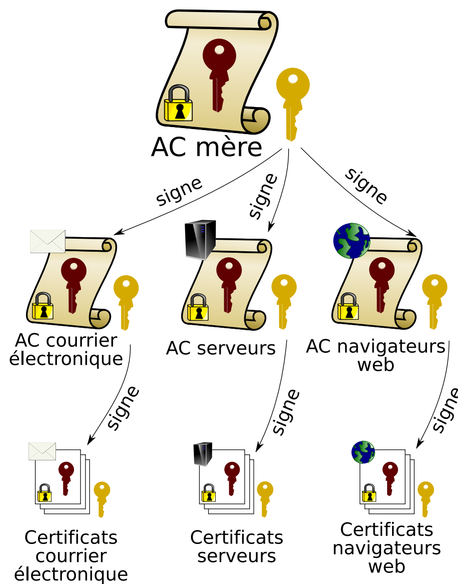


FIGURE 3.7 – Chaîne de certificats

1. on vérifie la période de validité ;
2. on s'assure que l'AC qui a signé le certificat est une AC de confiance, dont on possède le certificat racine ; on extrait la clef publique de l'AC pour l'étape suivante ;
3. on examine la validité de la signature du certificat serveur ;
4. selon les applications, on vérifie le nom de domaine. C'est le cas pour le protocole HTTP sécurisé, HTTPS.

Lorsqu'un serveur vérifie un certificat client, au lieu de vérifier le nom de domaine, il s'assure que le client possède la clef privée correspondante (voir l'étape CertificateVerify, section 3.1.3), et peut comparer le certificat à une copie installée localement, ou bien de manière complémentaire consulter les CRL ou un serveur OCSP pour savoir si le certificat est compromis (voir section 3.1.4). Enfin, il est possible de rechercher le *nom distingué*✶ renseigné par le certificat dans un annuaire de type LDAP, en même temps que les permissions accordées à l'utilisateur.

Il peut être utile de rappeler comment procéder à la vérification d'une signature, en appliquant au cas d'un certificat X.509 ce qui a été expliqué section 3.1.1. L'opération consiste :

1. à calculer avec la bonne fonction de hachage à sens unique l'empreinte numérique M du TBS. La fonction de hachage à utiliser est renseigné dans le certificat ;
2. à récupérer la clef publique de l'autorité de certification dans le certificat racine pré-installé,
3. ce qui permet de déchiffrer le contenu du champ signature S : $M' = D(S)$;

4. à s'assurer que $M = M'$, ce qui prouve que le certificat a bien été signé par cette autorité de certification, et qu'il appartient à la bonne personne (authentification), mais aussi qu'il n'a pas été contrefait (intégrité).

On valide un certificat fils de manière récursive avec la clef publique du parent, jusqu'à l'AC racine dont le certificat est « auto-signé », c'est-à-dire qu'il est signé avec la clef privée correspondant au certificat.

Formats de certificats

En premier lieu, les certificats X.509 peuvent en général être encodés de deux manières différentes. *DER* (« Distinguished Encoding Rules ») est un format binaire suivant les règles ASN.1 décrites par la norme X.509. *PEM* (« Privacy Enhanced Mail ») résulte de l'encodage d'un certificat DER en base 64, ce qui est plus adapté pour le courrier électronique comme suggéré par l'acronyme.

Un certain nombre de standards *PKCS* concernent également les formats de certificats. PKCS est l'acronyme de « Public Key Cryptographic Standards », ou standards de cryptographie à clef publique ; ces standards n'en sont pas vraiment dans la mesure où ils ont été conçus par la société RSA Security. Néanmoins, certains d'entre eux sont repris par des RFC émises par le groupe PKIX de l'IETF*.

PKCS#10 est le format des demandes de certificat. PKCS#12 définit un « conteneur » utilisé pour les certificats clients. Un certificat PKCS#12 rassemble en un seul fichier un biclef, le certificat correspondant et toute la chaîne de certificats, correspondant à la chaîne des autorités de certification.

3.1.3 Protocole SSL

SSL a été développé par la société Netscape, qui a équipé ses navigateurs de la version 2.0. La version actuelle est SSLv3. Le brevet déposé aux États-Unis [4] est racheté par l'IETF en 2001, qui décrit TLSv1* (Transport Layer Security) dans la RFC 2246. SSLv3 et TLSv1 sont subtilement incompatibles, mais suivent exactement le même principe.

Anatomie du protocole SSL

SSL est une couche cryptographique fonctionnant au-dessus d'un protocole de transport *connecté*, TCP. Il comporte deux phases, de natures très différentes.

La première utilise des moyens de cryptographie asymétrique. Client et serveur s'authentifient, et négocient une *clef de session*. Cette phase est dénommé en anglais par le terme *SSL handshake*, littéralement « poignée de mains » ; nous l'appellerons *négociation SSL*. Elle est prise en charge par le module *SSL Handshake Protocol*, en haut à gauche sur la figure 3.8. Nous nous concentrerons sur cette phase.

Par la suite, le module *SSL Record Layer* (placée au-dessus de TCP sur la figure) prend le relais. Son rôle est de chiffrer de manière symétrique le trafic avec la clef de session obtenue précédemment. La directive *Change Cipher Protocol* fait le lien entre ces deux phases (deuxième bloc en haut à gauche sur la figure).

4. Rappelons que l'Union européenne interdit les brevets logiciels.

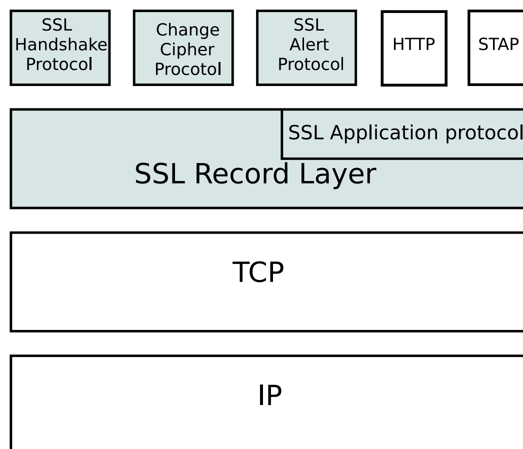


Figure 3.8 – Structurations en couches du protocole SSL

SSL fait appel à un très large éventail des possibilités offertes par la cryptographie. Qu'on en juge par un bref inventaire des familles d'algorithmes utilisées :

- un algorithme asymétrique pour l'authentification : RSA ou DSA ;
- un algorithme asymétrique pour l'échange de clefs secrètes : RSA ou Diffie-Hellman ;
- une fonction de hachage à sens unique pour calculer les codes d'identification de messages (MAC) ;
- la négociation achevée, un algorithme à clef secrète pour le chiffrement du trafic.

Négociation SSL

Il nous a paru utile de détailler les étapes de menant à l'obtention d'une clef de session, afin de bien saisir les tenants et aboutissants du protocole. Celles-ci sont synthétisées par la figure 3.9.

1. **ClientHello** : message par lequel le client demande la connexion au serveur et initie le dialogue. Le client renseigne la version de SSL qu'il supporte, un identificateur de session nommé *session_identifier*[5], et un nombre aléatoire de trente deux octets, *client_random*.

   Il présente également la liste des protocoles de compression qu'il connaît, ainsi que la liste des algorithmes cryptographiques qu'il accepte d'utiliser.

2. **ServerHello** : le serveur prend acte de la demande et renvoie la version du protocole SSL qu'il supporte, l'identificateur de session précédemment fourni, un nombre aléatoire sur trente deux octets, *server_random*, ainsi que la liste des algorithmes cryptographiques qu'il a choisis, parmi ceux proposés. Dans le cas où il n'y a pas correspondance, le serveur renvoie l'erreur *handshake_failure*, qui met fin au dialogue.

5. Cet identifiant permet la reprise de sessions SSL, dont nous ne traiterons pas ici, mais qui est abordé dans [**14**].

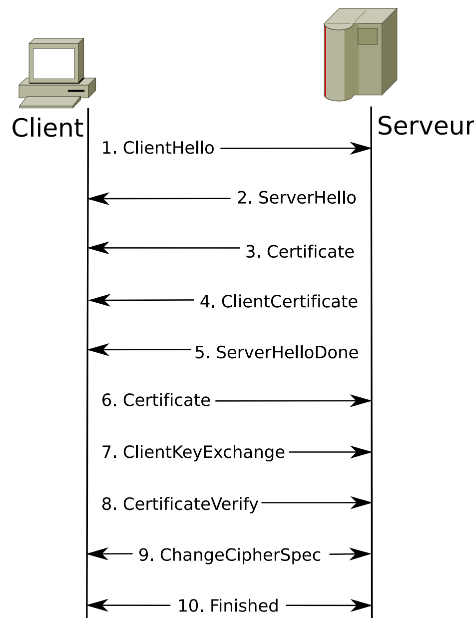


FIGURE 3.9 – Phase de négociation SSL

3. **Certificate** : le serveur envoie son certificat X.509. Le client le vérifie de suite, et coupe la connexion en cas d'erreur. Éventuellement, le message *ServerKeyExchange* peut être envoyé pour initier l'échange de la clef de session par la méthode de Diffie-Hellman ; autrement on utilise RSA pour cette opération.
4. **ClientCertificate** : cette étape est optionnelle ; elle est utilisée pour la double authentification. Le serveur fait savoir au client qu'il souhaite recevoir son certificat.
5. **ServerHelloDone** : le serveur n'a plus rien à communiquer au client ; il se met en attente d'une réponse.
6. **Certificate** : Si le serveur l'a demandé, le client envoie son certificat. S'il n'en a pas, il envoie le message d'alerte no_certificate. C'est alors au serveur de décider s'il poursuit la communication.
7. **ClientKeyExchange** : Le client envoie une pré-clef secrète PreMasterKey [6]. Il s'agit d'une suite aléatoire de quarante huit octets, chiffrée avec la clef publique du serveur.

   À partir de *PreMasterKey*, *client_random* et *server_random*, et à l'aide de fonctions de hachage, les deux parties calculent la clef de session *MasterKey* tant attendue.
8. **CertificateVerify** : En double authentification, le client prouve qu'il est en possession de la clef privée correspondant à son certificat en envoyant des données déjà échangées *chiffrées avec sa clef privée.*

6. si Diffie-Hellman est utilisé, cette donnée a déjà été échangée

9. **ChangeCipherSpec** : Le dialogue s'est déroulé avec succès, on informe le *record layer* qu'il peut prendre le relais.
10. **Finished** : la négociation est achevée. La session SSL commence. Dorénavant, les communications sont chiffrées de manière *symétrique* avec la clef de session *MasterKey*.

3.1.4 Infrastructure à clefs publiques

Une PKI (Public Key Infrastructure) ou ICP (infrastructure à clefs publiques) met à disposition les clefs publiques de ses utilisateurs (dont les supports sont des certificats X.509). Elle gère l'intégralité du cycle de vie d'un certificat, de sa signature à sa mort naturelle (dates de validité) ou sa révocation s'il est compromis. Un logiciel de PKI permet d'accomplir ces tâches et de garder une trace des certificats clients. Le groupe PKIX de l'IETF est à l'origine d'un effort de normalisation des spécifications et protocoles de PKI. Une ICP est composée de plusieurs entités ayant chacune un rôle particulier. Nous présentons ci-dessous un bref aperçu des plus courantes.

L'*autorité de certification* (AC ou CA) est une entité représentant le capital de la confiance de la PKI. C'est elle qui signe les demandes de certificat avec sa clef privée, dont la confidentialité est primordiale.

L'*autorité d'enregistrement* (AE ou RA) est chargée de recevoir les demandes de signature de certificat et de s'assurer de leur validité et de l'identité du demandeur. Elle peut être incluse à l'autorité de certification, selon la complexité de l'ICP.

L'*entité d'enrôlement* (EE) peut être accessible directement par l'utilisateur, ou uniquement par un opérateur de l'AE. Elle permet d'effectuer une demande de certificat, par exemple au travers d'une interface web. L'utilisateur peut fournir ses données, par exemple en les saisissant dans un formulaire, ou en transmettant une demande de signature de certificat au format PKCS#10.

L'*autorité de validation* gère les certificats compromis, c'est à dire ceux pour lesquels le secret de la clef privée n'est plus garanti. C'est clairement le point faible des ICP à l'heure actuelle. Un certain nombre de mécanismes sont normalisés, mais aucun n'est pleinement satisfaisant.

Les CRL (Certificate Revocation List, que l'on peut traduire par liste de certificats révoqués) sont « normalisées » par la RFC 3280. Mais celle-ci reste évasive sur un certain nombre de points, notamment le moyen de mise à disposition. Ainsi, en pratique, elles peuvent être accessibles à travers divers protocoles, comme LDAP ou HTTP.

Le mécanisme présente en soi un certain nombre de défauts. Le rythme d'actualisation peut varier fortement d'une ICP à une autre, et dans tous les cas subsiste un intervalle de temps pendant lequel le certificat compromis peut être utilisé de manière frauduleuse. Qui plus est, ces listes de révocation peuvent être volumineuses, et certains dispositifs embarqués incapables de les stocker. Enfin, et ce dernier défaut supplante tous les autres, très peu de clients les utilisent, ce qui aggrave le préjudice d'une compromission.

OCSP (Online Certificate Status Protocol, ou protocole de vérification en ligne de certificats), décrit par la RFC 2560, a été mis au point pour pallier les déficiences des CRL, sans toutefois pleinement y parvenir. Il ne s'agit plus de télécharger des listes, mais de demander en temps réel à un serveur si un

certificat est valide. Parmi ses principaux défauts, on peut citer le fait qu'il puisse déclarer valide un certificat qui n'a jamais été signé par une AC de confiance, mais aussi ses faiblesses face à certaines attaques, de par la mise en cache de certificats signés, et l'émission de messages d'erreur non signés, dont un pirate pourrait profiter pour tromper un utilisateur.

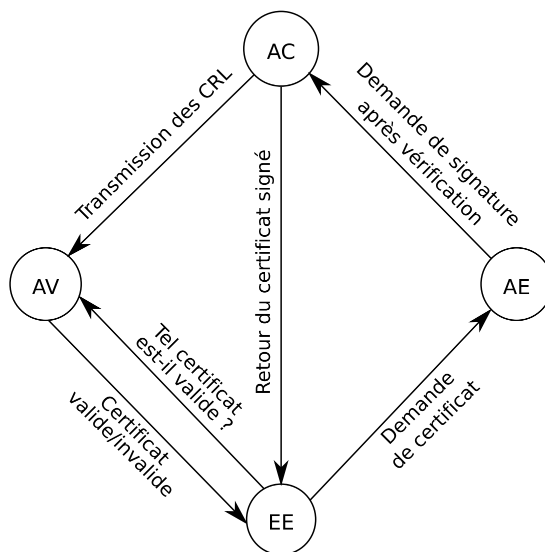


FIGURE 3.10 – Infrastructure à clefs publiques

La figure 3.10 présente les mécanismes de demande de signature de certificat sur (à droite) et de validation (à gauche). À travers l'entité d'enrôlement (EE) — qui peut être accessible au moyen d'une interface web, par exemple —, un utilisateur dépose une demande de signature de certificat ; celle-ci est tout d'abord examinée par l'autorité d'enregistrement (AE), qui procède à une vérification de l'identité du demandeur. Le cas échéant, elle demande à l'autorité de certification (AC) de signer le certificat. Il ne reste plus à l'utilisateur qu'à récupérer le certificat signé auprès de l'entité d'enrôlement.

L'autorité de validation (AV) reçoit régulièrement de la part de l'AC la liste des certificats compromis sous forme de CRL. Dès lors, quiconque désirant s'assurer de la validité d'un certificat peut, à travers l'entité d'enrôlement, interroger l'autorité de validation, qui peut selon le cas donner une réponse en temps réel si elle intègre un serveur OCSP, ou fournir à son tour les CRL.

3.2 Intégration de SSL aux applications monétiques

3.2.1 Simple authentification

On appelle *simple authentification* le mode dans lequel le serveur ne demande pas au client de présenter un certificat (voir étapes trois et six de la négociation, section 3.1.3), en conséquence de quoi le serveur est le seul à être authentifié.

Déploiement de la chaîne de certificats sur les clients

Il implique l'installation sur le TPE IP* de la chaîne de certificats, dont il a été question à la section 3.1.2. Cette étape est nécessaire, rappelons-le, pour que le TPE* soit en mesure de vérifier que l'authenticité de la clef publique, contenue dans le certificat du serveur, est garantie par une autorité de certification de confiance, ce que permet le mécanisme de signature des certificats.

Le déploiement de cette chaîne de certificats, tout comme le déploiement des certificats sur les TPE en général, n'est pas normalisé, et pour l'instant le silence du GIE Cartes Bancaires sur le sujet est assourdissant. Pour la charger sur l'EFT Smart de Sagem — le modèle sur lequel nous avons travaillé — il faut la transmettre à cette société, puis lancer une mise à jour ; il semblerait que l'on ne puisse avoir un contrôle direct sur cette opération. Il est fort probable que d'autres constructeurs aient recours à des procédures différentes, comme l'intervention physique d'un opérateur chez l'accepteur, par exemple.

Implémentation de SSL dans STAP

Le cœur du problème reste tout de même l'implémentation de SSL côté serveur. Est-il possible de rendre cette fonctionnalité indépendante de l'application existante ? Quelles en seront les conséquences sur les performances du serveur ? Quelles exigences de sécurité faut-il prendre en compte pour le déploiement de solutions SSL en milieu bancaire ? Nous verrons que ces questions sont liées.

Pour bien comprendre les enjeux concernant les performances, il faut se souvenir que la cryptographie asymétrique (abordée section 3.1.1) est très gourmande en calculs, un facteur mille la séparant de la cryptographie symétrique en termes de temps de calcul. Cela signifie que la négociation SSL (section 3.1.3) est une étape lourde pour le processeur, rendant négligeable le coût du chiffrement symétrique au sein d'une session SSL. Le problème se pose réellement dans le cas d'un serveur devant répondre à un nombre élevé de demandes de connexions par unité de temps ; nous verrons qu'en fait ce n'est pas forcément le cas de STAP en mode hors-ligne.

Utilisation d'un logiciel tiers gérant le dialogue SSL

Certains logiciels permettent d'ajouter une couche SSL à des applications dans lesquelles il n'est pas implémenté. stunnel est un logiciel cryptographique sous licence GPL*, reposant sur la bibliothèque OpenSSL. En mode serveur, il écoute les demandes de connexion SSL sur une adresse IP et un port TCP donnés, effectue les opérations cryptographiques nécessaires (ouverture d'une session SSL puis chiffrement des données), et transmet les paquets en clair à l'application cible, à travers cette fois-ci un socket client ouvert sur un autre couple adresse/port. Il se comporte donc comme un serveur mandataire*, et peut de surcroît être installé sur une machine séparée, qui se chargerait des calculs cryptographiques, le serveur hébergeant STAP se concentrant sur la gestion des opérations de monétique*.

Évidemment, les données transmises en clair doivent transiter par un réseau local considéré comme sûr, faute de quoi l'implémentation de SSL resterait lettre morte. Il est par ailleurs crucial de veiller à la confidentialité de la clef privée : rappelons-nous que c'est sur elle que repose le mécanisme d'authentification offert par les méthodes de cryptographie asymétrique. Si les exigences de sécurité

ne sont pas trop drastiques, on peut se contenter de la stocker sur un système de fichiers* particulier, éventuellement protégée par une « passphrase* ». Si tel n'est pas le cas, on recourra à un module matériel de sécurité, dont il est question ci-après. Une telle installation peut convenir pour certaines applications privatives, qui auraient à gérer moins de systèmes d'acceptation qu'un établissement bancaire, et qui ne seraient pas tenus au respect d'autant de normes de sécurité.

Nous avons eu l'occasion de tester la communication en SSL entre un TPE IP et STAP en utilisant stunnel. Après avoir fait installer le certificat racine de l'autorité de certification d'AFSOL sur le TPE, nous l'avons configuré de manière à ce qu'il appelle en SSL sur un port donné du serveur de télécollecte, sur lequel écoutait stunnel. Ce dernier transmettait les paquets en clair à STAP sur un autre port, mais sur la même machine, pourvu que les fichiers contenant clef privée et certificat soient correctement renseignés et accessibles. Au démarrage, stunnel prétend être en mesure de gérer simultanément plusieurs centaines de sessions, mais il était difficile de procéder à un test de mise à l'échelle pour voir ce qu'il en aurait été, avec un seul serveur hébergeant STAP et stunnel, ou deux dans le cas où on déporte stunnel sur un autre serveur.

Recours à l'accélération matérielle et aux modules matériels de sécurité

Il existe un certain nombre de dispositifs matériels capables de prendre en charge certains aspects de SSL ; ceux-ci répondent à deux besoins principaux, à savoir l'accélération des calculs cryptographiques, permettant d'augmenter le nombre de demande de connexions par unité de temps, et la confidentialité de la clef privée du serveur.

Nous avons étudié de tels dispositifs, en pratique, la gamme proposée par la société nCipher. Ces produits ont en premier lieu pour mission la protection des clefs privées. Il s'agit de *modules matériels de sécurité*, ou HSM, de l'anglais Hardware Security Module. Dans le meilleur des cas, une clef privée naît, vit et meurt à l'intérieur d'un HSM et n'en sort jamais. Un HSM est physiquement inviolable, c'est-à-dire que toute tentative d'ouverture du caisson, ou quelque autre agression que ce soit, mène immédiatement à la destruction de l'information sensible. Certaines opérations nécessitent les droits de plusieurs opérateurs à la fois, diminuant les risques de malveillance. PKCS#11 définit une API* pour les HSM, couramment utilisée par les constructeurs pour accéder aux fonctionnalités impliquant la clef privée.

Par la suite nCipher a mis au point des cartes accélérant le dialogue SSL. Dans tous les cas, selon ce qui nous a été expliqué lors de notre prise de contact par téléphone, elles ne prennent en charge que la partie asymétrique, c'est-à-dire la négociation. Les premières cartes jouaient *grosso modo* le rôle de coprocesseurs cryptographiques ; elles ne semblent plus très vendues et en voie d'obsolescence. Les cartes de seconde génération sont tout à la fois des HSM et des accélérateurs cryptographiques. Certaines sont même présentées sous forme de boîtiers accessibles par le réseau local. Il est apparu que les normes PCI DSS dont il a été question au premier chapitre imposent l'utilisation de matériel dont le coût est de l'ordre de 19000 € pour sécuriser un serveur... L'importance des montants en jeu impose une étude précise du marché avant de se décider pour un fournisseur.

Notre dialogue a permis de mettre l'accent sur certaines difficultés d'intégration de ce matériel avec STAP. Initialement, nous souhaitions pouvoir implémenter SSL séparément. Or, les produits nCipher ne prenant en charge, mise à part la partie HSM, que la partie asymétrique, il reviendrait à STAP de recevoir la demande de connexion (le message « ClientHello » présenté à la section 3.1.3), de transmettre la demande à la carte accélératrice, puis de prendre le relais, c'est-à-dire de gérer le chiffrement avec la clef de session qui vient d'être négociée.

Les changements requis seraient importants. Mais il existe une piste qui mérite d'être explorée. OpenSSL possède une extension appelée *Engine*, très mal documentée, qui lui permettrait de s'interfacer avec ce type de cartes. Or stunnel utilise OpenSSL, et d'après quelques questions posées sur le site de ce logiciel, il serait capable d'utiliser les primitives correspondantes. Il y a donc un espoir de charger stunnel de coordonner l'ensemble du dialogue SSL, mettant en jeu une carte accélératrice et la bibliothèque OpenSSL.

Il est apparu que, pour le moment, les besoins en calculs cryptographiques sont modestes. En effet, évalué rapidement, le nombre de demandes de connexions par seconde provenant de TPE appelant en X.25 reste inférieur à la dizaine, loin des centaines de négociations SSL par seconde revendiquées par le constructeur, et ce sur des serveurs très peu chargés. En IP, les débits étant bien plus importants, la connexion dure moins longtemps, ce qui devrait donner lieu à un nombre de demandes de connexion par unité de temps encore inférieur. Mais cela est vrai en mode « hors-ligne », quand le TPE appelle quotidiennement pour transmettre une remise* comportant les transactions* du jour. À l'avenir, la tarification des lignes TCP/IP ne se basant plus sur le temps de communication comme X.25, il est très probable que les TPE utilisent un mode « en ligne », dans lequel chaque transaction donne lieu à un appel ; cela augmenterait le nombre de connnexions au moins d'un facteur cent, quand bien même les appels seraient plus étalés dans la journée — en mode hors-ligne, les appels ont surtout lieu de nuit —, il est fort probable que l'accélération SSL ait alors un véritable intérêt.

Que ce soit pour respecter le niveau de confidentialité de la clef privée demandé par PCI DSS dans les mois qui viennent, ou pour préparer l'arrivée de la réception des transactions en ligne, il importe dès à présent de se pencher sur le choix de tels dispositifs. Ceux-ci ne seraient pas forcément revendus par AFSOL, mais proposés au client dans le cadre de l'intégration de SSL à STAP et de la mise en place des normes PCI DSS.

3.2.2 Double authentification

En *double authentification*, le serveur exige la présentation d'un certificat de la part du client. Ce mode est « recommandé » par le GIE Cartes Bancaires au détour d'un document, sans plus de précisions concernant son implémentation. De fait, les solutions proposées par les constructeurs de TPE sont encore plus expérimentales qu'en simple authentification.

Déploiement des certificats clients

Le principal écueil touche à la génération et au traitement de la clef privée du TPE. Fondamentalement, elle n'est pas censée être connue d'un autre que

l'accepteur. Globalement, deux visions ont cours, selon que l'ICP gérant les certificats clients fonctionne de manière centralisée ou décentralisée.

En mode décentralisé, l'accepteur génère lui-même un biclef. Comme on s'adresse à un public très large, pour lequel cette opération est *très* technique, il convient de mettre au point des systèmes suffisamment conviviaux et robustes, capables de prendre en charge cette étape. Par la suite, une demande de certificat au format PKCS#10 contenant la clef publique doit être envoyée à l'autorité de certification. Lyra Network, dont il a déjà été question aux chapitres 1 et 2, semble, d'après certaines spécifications que j'ai pu consulter, utiliser ce mode : à la première connexion, le TPE, ou un boîtier adjoint au TPE, envoie une demande de signature de certificat au format PKCS#10, accompagnée d'un certain nombre d'informations analogues à celles demandées dans la première trame de la RFC 1086 « Concert » (voir la section 2.3.2) et d'un numéro de client, destinés à identifier l'accepteur. Si ce dernier est connu de Lyra Network, et que son abonnement est à jour, le TPE recevra un certificat client signé, dans un conteneur PKCS#12. Rappelons (voir la section 3.1.2) que celui-ci contient la clef privée du client, son certificat, et la chaîne de certificats racine. Le protocole utilisé est complètement propriétaire, et n'est pas transférable en-dehors du réseau de Lyra Network.

En mode centralisé, qui semble être celui qu'a choisi Sagem, l'ICP se charge aussi de générer le biclef. Elle transmet alors le certificat PKCS#12 à son client, protégé par une « passphrase », sur un support *physique* tel qu'une clef USB, une carte mémoire ou encore une carte SIM. Sagem semble être encore en phase de recherche sur ce thème. Pour activer cette fonctionnalité, il nous a été communiqué le moyen d'accéder à des menus cachés du TPE ; mais il nous a été demandé de ne pas aller au-delà de la phase de test, l'implémentation n'étant pas définitive, ni entièrement validée.

Une fois encore, nous avons utilisé stunnel avec le Sagem EFT Smart. Celui-ci permet en effet de travailler en double authentification, c'est-à-dire d'exiger un certificat de la part du client pour l'authentifier. Le premier essai n'a pas abouti, et c'est à ce moment-là que nous nous sommes rendu compte, après analyse du problème, que le conteneur PKCS#12 devait également contenir la chaîne de certificats racine. Il faut également configurer correctement stunnel pour que la non présentation du certificat déclenche une erreur ; il existe en effet un mode intermédiaire dans lequel elle est optionnelle. Le fonctionnement basique de la double authentification a donc pu être validé sur le TPE de test.

Nous espérions initialement que les dispositifs cryptographiques de type cartes accélératrices permettraient de se charger du mécanisme de double authentification. Il s'est avéré qu'en ce qui concerne la gamme nCipher, ce n'est pas le cas. Une fois de plus, si l'on veut éviter le travail de développement, il faut se tourner vers des applications comme stunnel.

3.2.3 Quelques idées à propos d'une infrastructure à clefs publiques (ICP)

En double authentification, il y a autant de certificats clients actifs que d'accepteurs. Dès lors, il apparaît nécessaire de mettre en place une infrastructure à clefs publiques pour en permettre la gestion. Il s'agit de mettre en place une architecture relativement complexe qui, comme nous l'avons souligné lors de la phase de recherche documentaire sur SSL, est un projet à part entière.

Il faut dans un premier temps choisir, et éventuellement adapter, un logiciel. Parmi ceux que nous avons rapidement étudiés, ont été retenus OpenTrust PKI [7], EJBCA et OpenCA. OpenTrust PKI est supposé exister sous une forme commerciale, comportant notamment un certain nombre de modules d'interfaçage avec du matériel cryptographique, mais aussi en tant que logiciel sous licence GPL, or cette dernière est introuvable et n'a donc pas pu être testée. EJBCA s'exécute au sein de JBOSS, qui est assez lourd, et éloigné des technologies usuelles de STAP ; par ailleurs il nous a semblé peu modulaire. OpenCA utilise Perl et une interface web, ce qui est plus proche des technologies utilisées actuellement par AFSOL, et est beaucoup plus modulaire. Quand bien même la documentation pourrait être plus élaborée [8], nous pensons que c'est le meilleur choix.

Au-delà du logiciel, il importe de considérer l'infrastructure matérielle. Avec OpenCA, chaque entité de l'ICP (voir la section 3.1.4) peut être installée sur un serveur différent. Encore faut-il arrêter le nombre de serveurs, et les dimensionner. En théorie, il faut au moins séparer l'autorité de certification, qui a accès à la clef, au mieux par l'intermédiaire d'un HSM, de l'autorité d'enregistrement, plus directement en contact avec le public. Il faut étudier l'interaction avec le matériel cryptographique, ainsi que la mise à disposition des CRL, auprès de stunnel si c'est ce dernier que l'on utilise.

Enfin, le facteur humain nous semble prépondérant. Il sera très certainement nécessaire de former les opérateurs au sein d'établissements bancaires ou d'organismes privatifs, or comme le montre le long développement que nous venons d'écrire, la mise en place de SSL est un sujet relativement technique, faisant appel à de nombreux concepts. C'est un problème d'autant plus aigu qu'en cas de problème, il pourrait s'avérer difficile pour AFSOL d'intervenir sur certaines parties sensibles, sans avoir à partager la responsabilité du client en tant qu'ICP.

3.3 Bilan

Comme nous l'avions annoncé, cette partie ne pouvait déboucher sur une application directe ; le sujet en lui-même est vaste, et le temps nous était compté. Néanmoins nous estimons avoir ouvert la voie sur un certain nombre de points : la connaissance des concepts en jeu tout d'abord, que nous avons tâché de transmettre au cours d'une présentation faite devant nos collègues ; la mise en application à un niveau basique des mécanismes de simple et double authentification entre STAP et un TPE IP ; pour finir, une première vue d'ensemble du matériel cryptographique et des logiciels disponibles dans ce domaine.

7. anciennement IDEALX-PKI.

8. mais après tout c'est un défaut malheureusement répandu dans un certain nombre de projets libres*.

Chapitre 4

Écriture des bruts au format XML

Dans la seconde partie du TFE, il s'agissait, afin de préparer l'arrivée du protocole européen EPAS, d'étudier la mise au format XML des fichiers contenant les remises financières* reçues par le serveur de télécollecte. Ces fichiers étaient jusqu'à présent au format ASCII*, sous une forme fixe et séquentielle. Les nouvelles fonctionnalités devaient être capables de traiter les anciens fichiers ; nous verrons que la solution proposée permet une migration progressive de l'ensemble de l'application.

Nous offrirons dans un premier temps un aperçu de la chaîne de traitement des transactions financières* et des protocoles bancaires les plus utilisés. Ensuite nous expliquerons les enjeux de la migration, ainsi que la solution proposée.

4.1 Vue synoptique de la chaîne de traitement

Le cycle de vie d'une transaction, de sa conclusion entre le porteur et l'accepteur, jusqu'à l'exploitation de remises et la compensation* entre banques, est relativement complexe et comprend plusieurs étapes que nous détaillons ci-dessous. La figure 4.1 en présente une vision synthétique.

4.1.1 Initiation de la transaction et réception des remises (télécollecte)

Initialement, la transaction effectuée est stockée sur le TPE* de l'acquéreur [1]. Le TPE appelle le serveur de télécollecte (généralement la nuit) et lui transmet une « remise », c'est-à-dire un ensemble de transactions. Cette étape est prise en charge par la couche réseau du « front office », qui a été présentée section 1.2. À leur réception, les remises sont stockées dans des fichiers individuels, appelés « bruts ».

C'est sur ce processus que j'ai eu principalement à intervenir. Initialement, ces fichiers étaient sous une forme ASCII* fixe et séquentielle : les champs,

1. En mode « on line », les transactions arrivent au serveur en temps réel, ce qui engendre une charge plus importante. Avec le passage progressif en IP et l'abandon de la tarification à la communication, ce mode devrait se répandre de plus en plus.

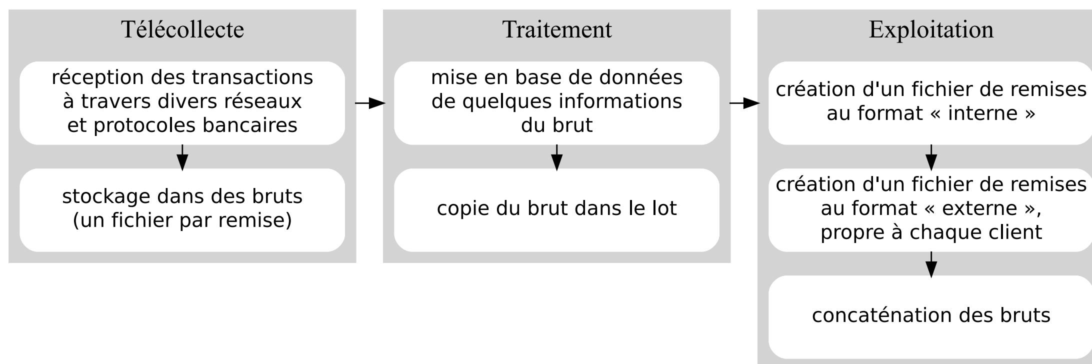


FIGURE 4.1 – Vue synoptique de la chaîne de traitement

de taille constante, sont décrits par une structure C. Ce mode de stockage est efficace, tant en termes de rapidité à l'exécution, que d'occupation mémoire ou disque. Il présente toutefois quelques inconvénients :

- Le dictionnaire est confondu avec le code source de l'application, et un tel format séquentiel est relativement pauvre d'un point de vue sémantique ;
- une modification du dictionnaire, à l'occasion par exemple d'un changement de protocole ou de version d'un protocole peut s'avérer problématique ;
- les « bruts » obtenus ne sont pas humainement lisibles, et difficilement échangeables.

Pour toutes ces raisons, il a été décidé de profiter du passage à ISO 20022 [**3**] pour adopter le format XML.

4.1.2 Traitement des remises

Une fois reçues du TPE et enregistrées sur disque, les remises sont *traitées* quotidiennement. Dans un premier temps, on place un sous-ensemble des informations présentes dans le brut en base de données. Ensuite, on rajoute une copie de chacun des bruts traités dans un lot. En effet, leur nombre est tel — de l'ordre de 40000 par jour sur un serveur —, qu'il serait possible dans certains cas d'atteindre le nombre maximal d'inodes* pour un système de fichiers* donné, on préfère donc les rassembler en un seul fichier. S'agissant de fichiers dont le format est fixe, il est possible de garder une trace de chacun des décalages dans un lot correspondant au début d'un brut. Pour les remises au format XML, nous proposerons une autre méthode à la section 4.3.8.

Les données présentes dans la base et dans le lot peuvent être modifiées ; ce n'est jamais le cas du brut ou du « concaténé » produit à l'exploitation.

4.1.3 Exploitation des remises

Cette étape permet de créer un fichier au format spécifié par le client, comprenant une liste de remises. Par suite aura lieu la compensation* entre banques, qui nécessite le calcul des débits et crédits globaux entre les établissements bancaires.

À l'issue de l'exploitation, pour les raisons sus-citées, on se défait des bruts individuels correspondant aux transactions exploitées, en les plaçant de manière séquentielle dans un fichier appelé « concaténé », analogue aux lots. Ce fichier est une copie de secours, et ne sera jamais modifié.

On voit qu'à partir du traitement les données sont présentes au moins en double (bruts et lots, puis concaténés et lots). Ce principe de « double écriture » permet la sécurisation des transactions financières.

4.2 Protocoles bancaires : de CBPR à UNIFI

4.2.1 Protocoles utilisés actuellement

Les protocoles bancaires permettant le transport des transactions financières utilisés couramment par AFSOL sont les suivants :

CBPR
: Ce protocole est censé être obsolète depuis 2000, néanmoins il est encore très utilisé pour des applications privatives. C'est un protocole assez simple, 40 octets suffisent pour décrire une transaction — sachant que le numéro de porteur, ou numéro de carte, occupe déjà 19 octets.

CHPR et CHPN
: Ces deux protocoles servent respectivement à la télécollecte chèque et l'autorisation chèque. Nous ne nous y étendrons guère, ayant surtout travaillé sur la partie carte bancaire.

CB2A
: C'est le protocole le plus utilisé actuellement ; il constitue la majeure partie du parc actuel de TPE. Basé sur la norme ISO 8583 [**2**], il est maintenu par le GIE cartes bancaires et, comme les protocoles sus-cités, il est tenu secret [2]. Il est divisé en plusieurs parties : CB2A fichier, CB2A autorisation, CB2A télécollecte/téléparamétrage/gestion réseau ; dans le cadre de ce travail, nous nous sommes contenté de CB2A fichier, qui contient le dictionnaire de données, et dont AFSOL utilise un sous-ensemble. CB2A est sensiblement plus complexe que CBPR. S'il est certes plus riche, il permet également de transporter les données sous quantité de formats, ASCII*, décimal codé binaire, binaire, etc., ce qui n'est certes pas pour en faciliter l'utilisation. La version 1.2 est actuellement en déploiement.

On voit que la France s'est dotée d'une autorité régulant les protocoles bancaires. Il n'en va pas de même dans d'autres pays, où dans des cas extrêmes chaque banque possède ses propres protocoles. À l'heure actuelle, l'interopérabilité en Europe est donc à peu près nulle. Cette situation ne satisfait plus un grand nombre d'intervenants, qui souhaiteraient pouvoir intervenir plus facilement sur d'autres marchés.

4.2.2 ISO 20022 et l'arrivée progressive d'EPAS

Depuis le lancement de l'euro, un certain nombre d'initiatives se mettent progressivement en place de la part de divers acteurs financiers, et pourraient aboutir à un ensemble de protocoles normalisés au sein de l'Union européenne.

2. ce qui est évidemment illusoire en termes de sécurité informatique

SEPA

Le SEPA [3] est une initiative de la Commission Européenne visant à la mise en œuvre de l'Europe monétaire. Sous sa forme la plus aboutie, elle doit pouvoir mettre en œuvre deux principes de base.

Chaque citoyen européen doit être en mesure de réaliser des paiements au sein de la zone euro avec la même facilité que ceux réalisés dans son propre pays (un compte, une carte). De plus, les paiements dans la zone euro ne peuvent pas être plus dispendieux que ceux réalisés dans le pays du client (principe de non discrimination).

Dans ce but, le règlement européen 2560/2001 stipule que les paiements en euros doivent être considérés comme des paiements domestiques au sein de la zone SEPA, et qu'il ne doit pas y avoir de discrimination entre les paiements par cartes en euros réalisés au sein de la zone SEPA, qu'ils soient réalisés au plan national ou dans un autre pays de cette zone. En fait, l'ensemble des paiements dans la zone euro doivent désormais être considérés comme des paiements domestiques. Cette réglementation s'applique à l'ensemble des paiements en euro, même si un pays n'a pas encore adopté l'euro comme monnaie nationale.

EPAS

C'est dans ce contexte qu'est né EPAS [4], un consortium de vingt deux partenaires européens. Son but est la mise au point de protocoles destinés à des TPE, comportant trois volets : un protocole acquéreur, un protocole caisse enregistreuse et un protocole de gestion du terminal de paiement. Il a pour objectif l'interopérabilité des protocoles de paiement par cartes bancaires au niveau européen.

Il se trouve que, à l'heure actuelle, l'accès aux spécifications d'EPAS reste difficile pour qui n'appartient pas au consortium. En revanche, on sait que le format des messages respectera ISO 20022, et que ce sera même le premier protocole ISO 20022 pour la carte de crédit.

ISO 20022

ISO 20022 [**3**] est une norme portant sur le format des messages financiers, mettant l'accent sur une conception « orientée objet ». Il est important de comprendre que lesdits messages restent à élaborer par les organismes qui le souhaitent. Après soumission et plusieurs étapes de validation, il seront ensuite inclus au catalogue ISO 20022 ou UNIFI [5], disponible sur le web [**6**].

Ce catalogue est maintenu par une autorité d'enregistrement (« Registration Authority »). À l'heure actuelle, il s'agit de SWIFT [6] [**5**]. Cette entité est basée en Belgique et a été fondée en 1973 par un ensemble d'établissements bancaires, sa mission principale est la normalisation des messages financiers.

Deux autres entités opèrent au sein d'UNIFI : un groupe gérant l'ensemble du processus d'enregistrement (RMG, Registration Management Group), ainsi qu'un groupe d'évaluation des standards (SEG, Standards Evaluation Group) pour chaque domaine de normalisation.

---

3. Single Euro Payments Area
4. Electronic Protocol Application Software
5. acronyme de UNIversal FInancial industry message scheme
6. Society for Worldwide Interbank Financial Telecommunication SCRL

Une fois accepté, un ensemble de messages est publié dans le catalogue, accompagné d'un schéma XML W3C [**21**, **22**, **32**, **33**]. Celui-ci n'est qu'en fait qu'un instantané des objets composant le message. À la section 4.3.3, nous effectuerons une brève présentation des schémas XML.

4.3 Migration vers UNIFI

Bien que le format exact des messages EPAS concernant les remises de transactions initiées par carte de crédit ne soit pas encore connu, il nous suffit en fait de savoir que tous les messages UNIFI peuvent être transportés au format XML et décrits par un schéma XML du W3C.

4.3.1 Écriture des bruts avant le passage à XML

Initialement, un brut est constitué (voir figure 4.2) :

- d'un bloc d'en-tête, contenant un certain nombre d'informations concernant l'ensemble de la remise, par exemple le numéro de remise, sa date de réception, ou le numéro de site acquéreur ;
- d'autant de blocs « transaction » que nécessaire, précédés de leur longueur ;
- d'un bloc de pied, indiquant la fin de la remise, et comportant par exemple le total des montants télécollectés, ainsi que le nombre de transactions en crédit, en débit, annulées ou en erreur.

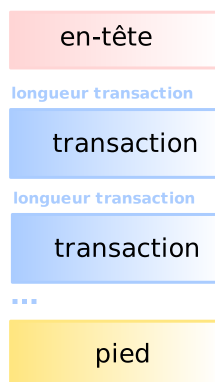


Figure 4.2 – Structure initiale des bruts

Chacun de ces blocs est décrit par une structure en langage C. Par exemple un bloc transaction pourrait correspondre au code figure 4.3.

Chacune de ces structures existe sous deux formes. En mémoire vive, on utilise une structure unique (pour un type de bloc donné), que nous appellerons structure de référence ou structure étalon. C'est la même quel que soit le type de protocole. En revanche, à l'écriture sur disque d'un bloc, le contenu de la structure de référence est transféré dans une structure propre à un protocole (CB2A, CBPR, CHPR, etc.).

Comme le montre la figure 4.4, les différentes parties de STAP communiquent avec les fonctionnalités d'écriture des bruts au moyen de la structure de réfé-

```
{
   char code; /* etat de la  telecollecte  */
   char date[8]; /* date de la transaction */
   char heure[6]; /* heure */
   char montant[30]; /* montant en centimes d'euros */
   char porteur[27]; /* numero de la carte de credit */
   ...
} trans;
```

FIGURE 4.3 – Un exemple de structure C correspondant à un bloc transaction

rence. Les structures particulières, elles, ne sont utilisées que pour l'écriture sur disque.

4.3.2 Passage à XML et compatibilité descendante

En fait, pour préparer la migration vers UNIFI, le plus important est de se donner les moyens d'écrire les remises au format XML, conformément à un schéma XML.

Ce nouveau mode d'écriture entraîne la disparition de la structure particulière. Néanmoins, il ne nous a pas semblé possible de maintenir la compatibilité avec les protocoles courants sans conserver le mécanisme impliquant l'utilisation de la structure étalon, en premier lieu parce que cela impliquait des changements drastiques dans STAP, impactant une grande partie du code source, et nécessitant en particulier la réécriture des couches de télécollecte ; en second lieu parce qu'il faut pouvoir convertir les bruts écrits à l'ancien format, ce qui ne se peut faire que grâce à ce mécanisme.

Le passage au format XML se fera protocole par protocole. Actuellement, ce changement ne concerne que les remises CB2A.

4.3.3 Outils logiciels

Cette section présente les principaux outils utilisés pour l'écriture des remises en XML, dont une compréhension globale est nécessaire avant d'aborder la section suivante.

XML

XML est un ensemble de règles permettant de structurer des données, sous une forme arborescente, sérialisées au moyen de balises lorsqu'elles sont transportées dans un fichier. En mémoire vive, elles peuvent donner lieu à la construction d'un arbre (approche DOM) ; on peut également adopter une approche événementielle : les données sont représentées par un flux générant des évènements (approche SAX). Nous avons utilisé l'approche DOM. Elle a l'inconvénient de générer des structures de données de taille importante en mémoire, de l'ordre de plusieurs centaines de mégaoctets dans STAP lorsque ce dernier reçoit en parallèle des remises d'une centaine de TPE, mais le mode de compatibilité avec les anciens bruts ne se prêtait absolument pas à l'utilisation de SAX. Il

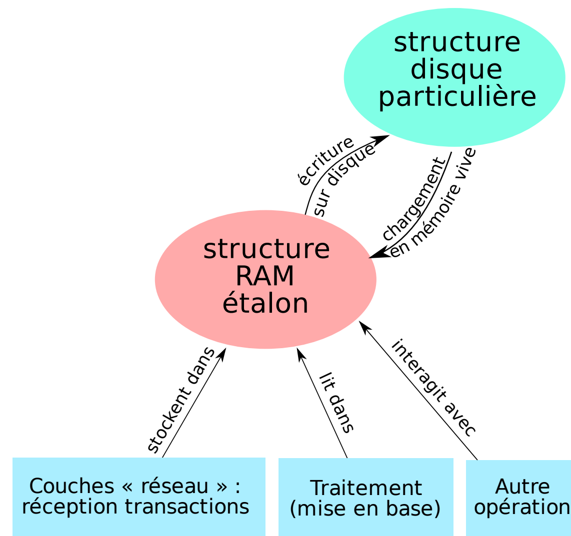


FIGURE 4.4 – Rapports entre la structure de référence et la structure particulière dans la méthode d'écriture de bruts de départ

n'est pas possible d'embrasser en un paragraphe toute l'étendue de XML. Nous conseillons la lecture de [**22**] et [**21**].

Bibliothèque XML

Une bibliothèque XML contient une collection réutilisable de fonctionnalités pour analyser et traiter des données au format XML. Nous avons considéré l'utilisation des logiciels libres* libxml2 et Xerces. Xerces est un projet de la fondation Apache ; c'est un analyseur XML (« parser ») reconnu, entièrement orienté objet*, ce qui se prête particulièrement à la nature de XML. Il dispose notamment de fonctionnalités plus puissantes pour le traitement des schémas XML que libxml2. Il est disponible en Java et en C++. Malheureusement, STAP est programmé en langage C « pur », et il est impossible de s'interfacer avec des classes C++ sans de nombreuses et périlleuses contorsions.

libxml2 est écrit en C, et possède des interfaces avec de nombreux autres langages, tels que C, C++, perl, et python. Initialement développée pour l'environnement de bureau GNOME*, elle est en fait très portable — elle s'appuie sur un nombre très réduit de bibliothèques C — et est disponible sous de très nombreux environnements. Elle s'intègre tout naturellement avec des logiciels écrits en C tels que STAP, et de fait elle était déjà utilisée pour l'analyse de certaines données XML. Dans le cadre de notre étude, on regrette cependant que — de l'aveu même des auteurs, par ailleurs — les fonctionnalités afférentes aux schémas XML du W3C soient réduites au minimum vital, et ne permettent guère plus que la validation globale de données XML. Nous verrons que nous avons contourné cette difficulté en faisant appel à des transformations XSLT. Lors du développement, nous avons utilisé la documentation en ligne du projet [**24**], qui se limite en général à un ensemble de signatures de fonctions, et n'est donc pas d'une consultation aisée.

Schémas XML du W3C

Un document XML doit être bien formé* et valide*. Le premier point signifie simplement qu'il doit respecter la syntaxe XML (fermeture et imbrication correcte des balises, etc.), le second qu'il respecte les contraintes inhérentes à une application. Les DTD* ont été le premier mécanisme de validation ; toutefois, le W3C a jugé nécessaire de concevoir un langage XML (ce n'est pas le cas des DTD), autorisant un typage plus fin des éléments, ce qui a donné lieu à une recommandation [**31**–**33**]. Relax NG et Schematron sont d'autres langages poursuivant des buts similaires. Pour plus d'informations, on pourra se reporter à [**22**] ou à [**21**].

Éditeur de schémas XML du *framework* Eclipse

L'édition manuelle de schémas XML avec un éditeur de texte, même performant, peut s'avérer fastidieuse. En effet, leur syntaxe est assez verbeuse et peut donner lieu à un enchaînement de définitions assez complexe, pour peu que l'on ne déclare pas tout en local. Un éditeur de schémas peut être très utile. L'environnement de développement Eclipse intègre un module d'éditions de schémas dont l'interface permet très rapidement de créer ou de modifier un schéma existant, ce qui facilitera les évolutions futures des schémas que nous avons élaborés.

XPath

XPath est une syntaxe non XML, faisant l'objet de recommandations du W3C [**27**, **29**], servant à désigner une portion d'un document XML. Il est embarqué dans XSLT [7], mais peut être aussi utilisé séparément pour extraire des informations d'un arbre XML. Nous avons utilisé la version 1.0, tant à l'écriture de feuilles XSL qu'au sein des fonctions développées, à travers libxml2.

Processeur XSLT

XSLT (Extensible Stylesheet Language Transformations) est une syntaxe XML, faisant l'objet de recommandations du W3C [**28**, **30**], permettant de décrire la transformation de un (ou plusieurs) documents XML en un autre document XML, un document HTML, ou encore un fichier au format texte. Nous y avons fait appel à plusieurs reprises, notamment pour transformer un schéma XML en un autre document XML. Pour une présentation précise de ce langage, on pourra se reporter à [**22**] ou [**21**], ou encore à l'excellent ouvrage de Doug Tidwell [**23**] qui y est entièrement consacré.

Un *processeur* XSLT permet de réaliser une telle transformation à partir du fichier source et d'une feuille XSL. Nous avons fait appel à l'exécutable `xsltproc`, installé par défaut sur de nombreuses distributions GNU/Linux. Ce programme est basé sur libxml2.

La version 2.0 permet de réaliser des traitements plus puissants ; en particulier, elle est capable d'utiliser les types employés par les schémas XML, ce qui ouvre la voie à une coopération toujours plus étroite entre les diverses technologies fondamentales gravitant autour de XML, et qui se serait avéré très pratique.

7. XSLT 1.0 utilise XPath 1.0, XSLT 2.0 XPath 2.0.

Elle n'est encore implémentée que dans un nombre réduit de bibliothèques XML, nous nous en sommes donc tenu à la version 1.0 implémentée dans `xsltproc`.

Flex et Bison

Flex et Bison [**35**, **40**] sont des outils permettant de générer respectivement des analyseurs lexicaux et syntaxiques en C ; ce sont des versions libres* des programmes correspondants Lex et Yacc. Un analyseur lexical sert à repérer des motifs tels que des mots-clefs, dans notre cas dans un fichier texte, tandis qu'un analyseur syntaxique [8] met en évidence la structure d'un tel texte, et permet de générer des actions correspondant à certains agencements déterminés. En général les deux sont associés, ce qui rend possible le déroulement des trois phases d'analyses lexicale, syntaxique et sémantique.

Grâce à ces outils nous avons pu écrire, de manière propre et fiable, un analyseur de code C, chargé de transformer des données décrites dans des structures C en un fichier XML, qui peut ensuite être manipulé facilement par les technologies XML usuelles telles que XSLT.

Makefiles

L'utilitaire `make`, couramment utilisé sous les systèmes de type Unix, est un moteur de production de fichiers. Il utilise des *makefiles**, qui décrivent un ensemble de cibles (souvent des fichiers), et calcule les dépendances entre ces cibles de manière à les construire ou reconstruire récursivement.

Les makefiles servent le plus souvent à la compilation de programmes, c'est ainsi que sont générés les exécutables de STAP. Ils nous ont aussi servi à automatiser la production d'un certain nombre de fichiers XML.

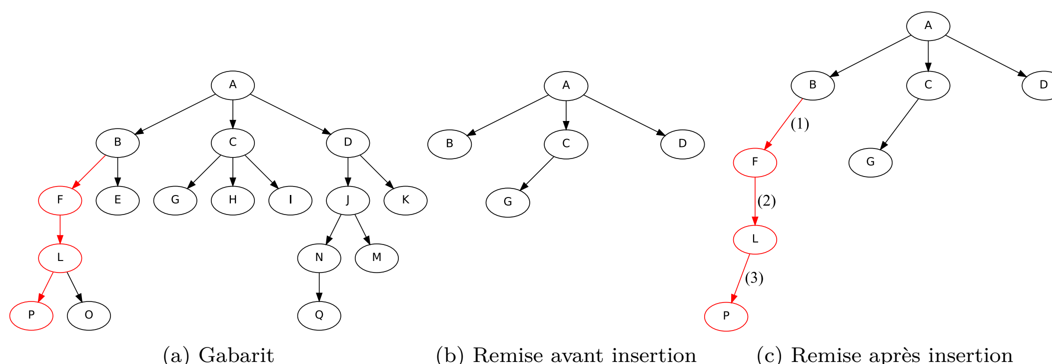
4.3.4 Génération des bruts XML

Il s'agit, dans le cas où les données nous parviennent dans un format autre que XML, de s'interroger sur la méthode qui sera utilisée pour assurer la conversion au cours de la télécollecte. Nous venons de voir que, souhaitant agir selon des standards semblables à UNIFI, nous souhaitons nous appuyer sur un schéma XML.

Un schéma XML sert généralement à valider un document (tant sa structure que le type des données scalaires, appelées types simples dans le vocabulaire du W3C), et ainsi à assurer la cohérence des données. Mais si cette application est la plus courante, il serait dommage de s'y cantonner [**22**] : le schéma peut être utilisé comme documentation [9], et peut être considéré comme le *dictionnaire des données*. Ce dernier n'est plus présent en dur dans le code de l'application, ce qui facilite son évolution ; de plus, il jouit d'une syntaxe beaucoup plus riche que ne le permet la déclaration d'un ensemble de structures C : le typage des données est beaucoup plus fin, et XML porte naturellement à l'utilisation d'arborescences, forcément plus hiérarchisées que l'ancien format « à plat ». Enfin, la validation des données est désormais confiée en majeure partie à des bibliothèques XML, allégeant encore une fois l'application.

---

8. L'anglais utilise le terme « parser ».
9. quitte à appliquer une transformation de type XSLT pour le rendre plus lisible pour le néophyte



(a) Gabarit (b) Remise avant insertion (c) Remise après insertion

FIGURE 4.5 – Copie d'un nœud du gabarit dans l'arbre XML de la remise. Les chiffres entre parenthèses de la figure de droite indiquent l'ordre de création effectif des branches.

On ne saurait s'arrêter en si bon chemin. Puisque les données sont si bien décrites dans un schéma, autant s'en servir pour générer un document XML respectant forcément une structure donnée.

Il se trouve que la syntaxe relativement complexe et protéiforme des schémas XML rend difficile une telle opération ; de plus, libxml2 n'intègre pas dans son API* de primitives suffisamment puissantes pour traiter un schéma. Néanmoins, il n'est pas interdit de recourir à un intermédiaire. Un schéma XML étant lui-même un document XML, on peut en obtenir un autre document XML, à travers une transformation XSLT. Une telle feuille XSL (baptisée `xsd2xst.xsl`) a été écrite et génère ce que nous conviendrons d'appeler un « gabarit* » [10].

Ce « gabarit » est un arbre XML composé de toutes les branches possibles, présentes chacune une seule fois.

Lorsque l'on souhaite générer un nouvel élément XML dans la remise en cours, il faut renseigner :

- Le nœud du gabarit que l'on souhaite copier dans la remise, il peut y être fait référence au moyen d'une expression XPath ;
- le nœud de la remise sous lequel doit être inséré le nouveau nœud (appelé « nœud courant de la remise ») ;
- le contenu que l'on souhaite attribuer au nouveau nœud.

L'opération consiste à remonter récursivement, à partir du nœud du gabarit, jusqu'à l'homologue du nœud courant dans le gabarit, et à recréer la branche au fur et à mesure si elle n'existe pas déjà dans la remise. Le fait de ne pas trouver, en remontant dans le gabarit, de nœud portant le même nom que le nœud courant de la remise, génère une erreur. La figure 4.5 illustre ce mécanisme de recopie d'éléments du gabarit dans la remise en construction.

4.3.5 Lien avec la structure de référence

Nous venons d'expliquer comment générer progressivement l'arbre XML d'une remise, dans le cas où cette dernière n'a pas été reçue nativement en XML. Il reste à expliquer comment le lien a pu être fait avec ce qui préexistait,

10. auquel on attribue l'extension `.xst`, `t` pour « template », c'est-à-dire gabarit.

et notamment la structure de référence, qui ne pouvait pas être occultée sans dommages considérables.

Nous sommes donc pour l'instant entre deux mondes : d'un côté un format fixe, dont la syntaxe repose sur la position des données dans un fichier ou dans une structure en mémoire vive, de l'autre un format pour lequel la notion de longueur des données n'est pas pertinente, fortement structuré au moyen de balises décrivant une arborescence.

Si l'on ne veut pas continuer à lier une partie du dictionnaire au code applicatif, le langage C ne permettant pas une « introspection » sur ses structures de données [11], il est donc nécessaire d'introduire des métadonnées décrivant la longueur de chaque champ et son décalage (« offset ») dans la structure.

Pour ce faire, un analyseur syntaxique a été écrit avec Flex et Bison [**35**, **40**]. Une grammaire du C ANSI [**37**, **38**], écrite par Jeff Lee, a servi de source d'inspiration pour écrire une grammaire beaucoup plus réduite, permettant d'analyser un fichier d'en-tête C comportant un ensemble de structures, et de générer un arbre XML renseignant, pour chacune d'entre elles, la longueur et le décalage de chaque champ.

À ce stade, nous avons légèrement avancé, dans la mesure où nous disposons d'un schéma XML tenant lieu de dictionnaire d'un côté, et des métadonnées que nous cherchions à obtenir sur les structures de l'autre, mais il reste bien évidemment à relier les deux. Le mieux serait que cette liaison fût présente dans le dictionnaire, mais cette information n'est pas aisée à introduire au beau milieu d'un schéma XML a priori... à moins de tirer parti de certains éléments de la syntaxe des schémas permettant ni plus ni moins que d'introduire des métadonnées dans ce qui est déjà des métadonnées. [**32**], section 3.13, introduit en effet le concept d'*annotations*, correspondant à la balise `xs:annotation`. Cette dernière peut contenir :

- des annotations destinées aux *humains* (`xs:documentation`), pouvant servir de documentation sur le schéma, utilisées soit directement, soit par une transformation XSLT. Ces annotations ont l'avantage par rapport aux commentaires d'avoir une sémantique plus claire, et d'être plus facilement traitables ;
- des annotations destinées aux *applications* (`xs:appinfo`), contenant n'importe quel document XML.

Ces annotations ne doivent interférer avec le processus de validation [12]. Tout au plus est-il demandé que le contenu de `xs:appinfo` contienne un fragment de XML *bien formé**, ce qui est la moindre des choses.

Nous avons fait usage du deuxième type d'annotation (figure 4.6). L'élément « brut » contient quatre informations sous formes d'attributs :

- *nom* est le nom du champ tel qu'il est déclaré dans la structure ;
- *structure* est le nom de cette structure dans le fichier d'en-tête ;
- *auto*, prend les valeurs « oui » et « non » selon que l'on souhaite que le programme recherche l'information automatiquement, ou qu'exceptionnellement on souhaite procéder « manuellement », par écriture de code C, par

11. contrairement à d'autres langages moins fortement typés, et souvent interprétés, comme PHP, qui implémente des « tableaux associatifs », que l'on peut par exemple parcourir au moyen d'un boucle.

12. [**32**] prend soin de préciser :« Annotations do not participate in validation as such. Provided an annotation itself satisfies all relevant Schema Component Constraints it cannot affect the validation of element information items. »

exemple dans des cas où l'on souhaite rajouter dans le brut une information qui n'est pas fourni lors de la télécollecte, ou encore lorsqu'il est trop difficile d'assurer automatiquement la conversion de la donnée selon le type déclaré dans le schéma ;
- enfin, *formatBrutStruct* spécifie pour quelques cas particuliers (surtout les dates, en définitive) quel est le format de la donnée dans la structure, sous une forme exploitable par une fonction de conversion.

À ce stade, on est théoriquement capable, lors de la télécollecte, d'aller piocher les éléments dans les structures de référence. Mais pour l'instant, il faut aller chercher l'information dans deux ou trois documents XML différents [13], en utilisant la syntaxe XPath. En pratique, le temps de recherche est non négligeable (de l'ordre de une à deux secondes pour une structure contenant entre dix et vingt éléments !), et ce du fait d'une implémentation de XPath dans libxml2 fort peu optimisée et de la propension qu'a cette bibliothèque d'allouer de la mémoire (qu'il faut de suite libérer) dès que l'on souhaite accéder au contenu d'un élément, en plus du temps incompressible nécessaire pour une telle recherche, même avec des outils plus performants.

Mais rien n'est perdu. Il suffit en effet d'effectuer ces opérations de recherche avant l'exécution du programme (en fait, comme on le verra, durant la phase de compilation). À partir du gabarit, des annotations du schéma et du document XML décrivant les structures (`structures.xml`), on produit un nouveau document XML, que l'on nommera le corpus (`corpus.xml`), et qui renseignera, pour chaque schéma XML utilisé, pour chaque structure, les éléments XML à générer et les métadonnées afférentes correspondant à la structure. Ce document est pris en compte au démarrage (voir la section 4.3.7 pour de plus amples renseignements), et est *parcouru* dès que l'on souhaite effectuer la conversion entre une structure et un document XML.

```
<info>
  <cb2a tag="22"/>
  <bdd champ="trnum"/>
  <brut nom="numremise" structure="edf" auto="oui"
     formatBrutStruct="AAJJMM"/>
</info>
```

FIGURE 4.6 – Fragment de XML présent dans une annotation de type `xs:appinfo`

4.3.6 Intégration à la chaîne de compilation

Le mécanisme décrit à la section précédente nécessite d'effectuer un nombre de traitements, qui sont intégrés à la chaîne de compilation. Cela permet d'automatiser le processus, dont on n'aura plus à se soucier que lors d'une évolution, mais aussi d'assurer la cohérence entre la structure de référence décrite dans un en-tête C et sa transcription en XML effectuée par l'analyseur syntaxique.

13. deux si l'on place les annotations dans le gabarit, mais cela reste peu efficace

La figure 4.7 synthétise le processus. Le fichier `brutess.h` contient (entre autres) les structures de référence. L'analyseur Flex/Bison se charge de le transformer en fichier XML décrivant chacune de ses structures et de ses champs (`structures.xml`). Par ailleurs, pour chaque schéma XML décrivant un type de remise, on effectue une transformation XSLT pour obtenir le gabarit dont nous avons déjà abondamment parlé précédemment (fichiers `.xst`). L'ensemble de ces fichiers (gabarits et `structures.xml`) subissent une nouvelle transformation XSLT qui génère le corpus (`corpus.xml`), fichier rassemblant, pour chaque dictionnaire (schéma XML), puis pour chaque structure, l'ensemble des éléments XML à générer.

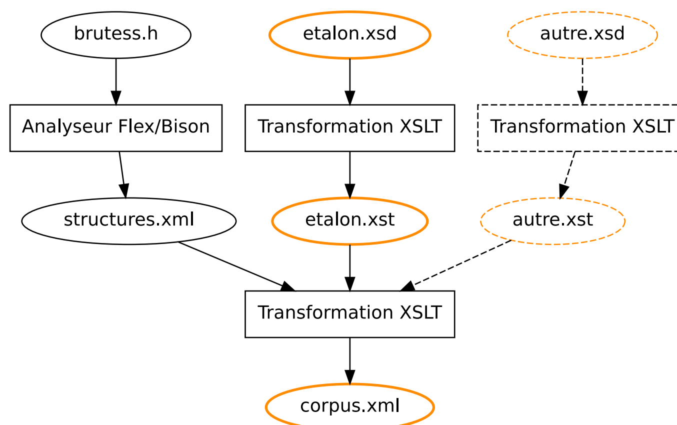


FIGURE 4.7 – Présentation générale du processus de génération du corpus et des gabarits lors de la chaîne de compilation

La branche en traits discontinus montre que ce processus fonctionne quel que soit le nombre de schémas (il peut par exemple y en avoir un pour CBPR, un pour CB2A, et un autre pour UNIFI). Les fichiers entourés en orange sont ceux qui servent à l'exécution : `corpus.xml` est chargé dès que possible, les gabarits permettent la génération des remises, les schémas restent tout de même nécessaires pour effectuer une validation une fois que l'on pense être arrivé au terme de la génération d'un brut.

La figure 4.8 est une image simplifiée du makefile* servant à générer ces fichiers. Il est légèrement plus complexe que celui présenté par la figure précédente, un certain nombre d'entre eux servant à s'assurer de la bonne marche du processus (détection des erreurs du développeur), d'autres n'étant que des intermédiaires.

On retrouve en jaune les mêmes fichiers que dans la figure précédente [14] : le dictionnaire (fichier `.xsd`), le gabarit (fichier `.xst`), et le corpus.

Les fichiers en mauve permettent d'exprimer les déclarations des structures au format XML. Le fichier `structures.y` contient une grammaire Bison, `structures.lex` la partie nécessaire à l'analyse lexicale. Ces deux fichiers, après application des exécutables flex et bison permettant d'obtenir du code

14. Pour des raisons de clarté, ce schéma ne représente qu'un dictionnaire à la fois. Il s'agit ici d'`etalon.xst`, qui sera utilisé pour CB2A.

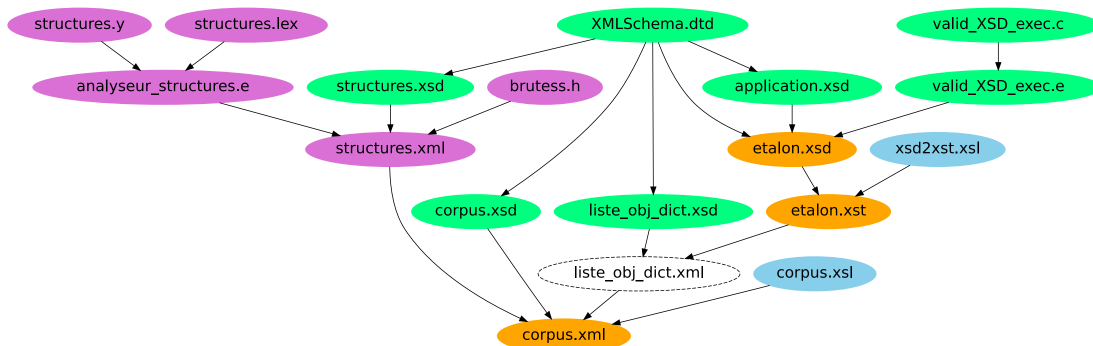


FIGURE 4.8 – Vue simplifiée du makefile permettant l'obtention du corpus et du gabarit.En vert les fichiers intervenant dans la validation du processus lui-même, en bleu les feuilles XSL, en orange les fichiers utilisés à l'exécution, en mauve les fichiers permettant d'aboutir à une transcription en XML des définitions des structures de référence. Le fichier entouré en traits discontinus n'est qu'un intermédiaire permettant d'effectuer une transformation XSLT de plusieurs fichiers à la fois.

source C, puis compilation avec gcc, sont à l'origine d'un exécutable (`analyseur_structures.e`) capable d'effectuer l'analyse syntaxique d'une structure C et de générer du XML. Il est appliqué sur brutess.h [15], qui contient les définitions des structures, et génère `structures.xml`.

Les feuilles de style XSL sont en bleu. La première, `xsd2xst.xsl`, génère un gabarit (`etalon.xst`) à partir d'un schéma XML annoté (`etalon.xsd`). La deuxième, `corpus.xsl`, transforme structures.xml et l'ensemble des gabarits, en « corpus », décrit précédemment. Cette transformation nécessite un intermédiaire entouré en traits discontinus (`liste_obj_dict.xml`). En effet, on ne peut invoquer un processeur XSLT que sur un fichier XML à la fois ; en revanche, la fonction XSLT `document()` permet de faire référence à un document XML extérieur dans le nom se trouve dans le fichier. C'est pourquoi, en pratique, on génère à la volée un petit fichier XML contenant comme seule information les noms des documents XML à traiter.

Les fichiers en vert, dont nous n'avons encore soufflé mot, servent à contrôler le processus que nous venons d'expliquer. Un certain nombre de schémas XML auxiliaires permettent de valider au fur et à mesure les fichiers XML générés. `structures.xsd` valide* `structures.xml`, une fois obtenu par analyse syntaxique de `brutess.h`. `liste_obj_dict.xsd` valide le petit fichier intermédiaire `liste_obj_dict.xml` généré à la volée dont nous venons de parler. `XMLSchema.dtd` est la DTD* écrite par le W3C pour les schémas et permet de s'assurer qu'ils sont correctement écrits.

Enfin, un outil a été spécialement écrit en C pour valider un schéma annoté, `valid_XSD_exec.c`, qui procède en deux temps. Le schéma est séparé de ses annotations, et validé avec `XMLSchema.dtd` [16], puis les annotations sont validées

15. auparavant, on invoque le précompilateur de gcc (options -E et -P) sur brutess.h, principalement afin de procéder au remplacement des macros et directives de compilation.

16. Cette DTD ne permet malheureusement pas de valider un schéma avec ses annotations,

séparément grâce au schéma `application.xsd`.

Arrivé à la fin du processus de compilation, si aucune erreur n'est retournée, le développeur est certain qu'un fichier `corpus.xml` valide et cohérent a été obtenu.

4.3.7 Modifications apportées au code préexistant

Cette section référence les ajouts au code qui ont été nécessaires pour intégrer le principe de remise XML tout en restant compatible avec les fonctions préexistantes d'écriture de bruts. L'annexe B présente le détail des appels de fonctions ainsi que des structures de données.

Fonctions afférentes aux remises XML

Ces fonctions, documentées à la section B.1, permettent d'effectuer les divers traitements génériques qui s'appliquent aux remises. Une remise est décrite par une structure C `ST_REMISEX`, renseignant entre autres le dictionnaire correspondant et son nom, l'arbre XML en construction, le gabarit, le schéma pour validation, ainsi que d'autres données telles que le nom des balises contenant une transaction ou un en-tête.

Les fonctions `creerRemiseX`, `chargerRemiseX`, `InitRemiseX`, `libererRemiseX` et `writeX` permettent la création, le chargement et la libération ainsi que l'enregistrement sur disque avec ou sans validation d'une remise. `nouvelleTransactionX` crée une transaction supplémentaire dans l'arbre en mémoire, `getChampX` et `remplirX` permettent de récupérer la valeur d'un élément ou d'en créer un. `remplitAuto` et `xmlToStruct` sont là pour convertir un bloc décrit par la structure de référence en XML et vice versa. On peut noter les fonctions `accrocher` et `fils` qui autorisent la création dynamique d'un arbre XML à partir d'un gabarit.

Fonctions de chargement du dictionnaire

Il est question dans le détail de ces fonctions à la section B.2. Un corpus, que nous avons déjà eu l'occasion de présenter, est chargé à partir du fichier `corpus.xml`, dans une structure de données appelée `ST_CORPUS_BR_XML`. La figure 4.9 utilise des éléments de la syntaxe d'UML* (Unified Modeling Language) pour présenter les éléments qui le composent. Il est important de noter

ce qui est fort dommage dans la mesure où un parseur validant n'est pas censé s'inquiéter outre mesure de ces dernières lorsqu'il analyse un schéma.

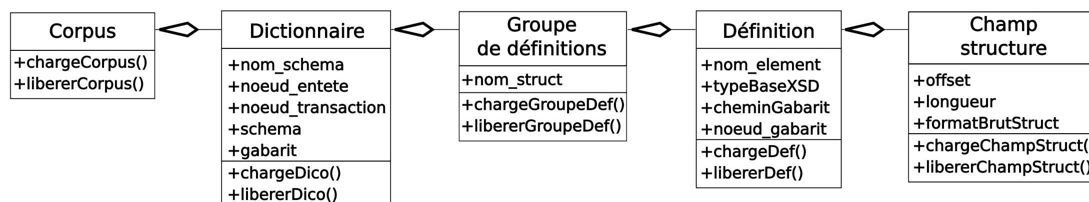


Figure 4.9 – Schéma UML décrivant la structure de données corpus

que STAP est écrit dans un langage de programmation structurée, et non orienté objet*, rendant en toute rigueur impropre l'utilisation de ce formalisme, mais nous avons trouvé commode de représenter ainsi l'agrégation des structures composant un corpus. Lorsqu'il y a lieu, nous renseignons de manière non exhaustive les fonctions (et non les méthodes) s'appliquant sur cette structure, ainsi que les champs (et non les propriétés) contenus.

Un corpus est en fait un ensemble de dictionnaires, un dictionnaire correspond à un schéma XML annoté (un par protocole). Il contient à son tour un groupe de définitions (un par structure de référence), chaque définition coïncidant avec un type simple décrit dans le schéma. Pour chacune, on décrit la position et l'offset des données en correspondance dans la structure de référence, et éventuellement un paramètre indiquant le format des données dans celle-ci, surtout utilisé pour les dates.

L'opération de chargement consiste à récupérer toutes ces données dans `corpus.xml`, et à en traiter certaines. Ainsi, pour chaque dictionnaire, on charge le gabarit adéquat, et chaque définition possède un lien vers un nœud de ce dernier. De la sorte, lors de l'opération de conversion, il n'y a pratiquement aucune recherche à effectuer.

Fonctions d'ouverture et fermeture des bruts

Nous avons vu que l'écriture des bruts en mode séquentiel sauvegardait les données bloc par bloc les données sur disque, ce qui n'est plus possible en XML. Il faut désormais créer un arbre XML à l'ouverture de la remise, et le sérialiser à la fermeture. Il a donc fallu introduire, pour permettre la compatibilité avec ce mode, des points d'entrée, là où on se contentait auparavant d'invoquer les fonctions C `open` et `close`. `typ_brut2` sert à charger une transaction existante, `openf_x` à créer une transaction, et `close_brut` doit être appelé à la fermeture. Ces fonctions sont documentées à la section B.3.1.

Fonctions de lecture et d'écriture par blocs

Des modifications, évoquées à la section B.3.2, ont dû être effectuées dans les anciennes fonctions de lecture/écriture par blocs, dénommées génériquement `readB` et `writeB`. Elles consistent principalement à introduire des appels aux fonctions de conversion `remplitAuto` et `xmlToStruct` pour les protocoles passés en XML.

4.3.8 Développements futurs

Nous avons fourni une base permettant l'évolution progressive de STAP, nous détaillons les modifications à venir en nous arrêtant sur quelques points, et nous permettons d'émettre une proposition pour le chiffrement des données sensibles demandé par PCI DSS.

Remplacement des concaténés

Les concaténés comme les lots, générés respectivement lors des phases d'exploitation et de traitement à partir de bruts à un format fixe, ne conviennent

plus à des fichiers XML. S'agissant des concaténés, nous proposons de les remplacer par une archive de type TAR, qui pourrait accueillir les bruts XML correspondant à une exploitation, compressés avec gzip.

Mise en base de données et lots

Nous proposons de supprimer le lot pour les formats migrés vers XML. Postgresql, la base de données utilisée actuellement par STAP, permet de stocker des données XML, et propose même un type dédié à celles-ci, ainsi que des fonctionnalités permettant par exemple de réaliser une extraction XPath ou une transformation XSLT au sein de la base. L'idée est donc de stocker en base l'élément XML correspondant à une transaction ainsi que tous ses descendants. Ainsi on introduit l'intégralité de la transaction, et non plus les éléments critiques, ce qui rend le lot inutile.

Auparavant, on ne stockait pas toutes les informations afin de ne pas alourdir la base. Au passage, on résout ce problème en assimilant une transaction à un seul champ. La double écriture est toujours assurée par le brut ou l'archive remplaçant le concaténé.

Mise à niveau progressive des fonctionnalités de STAP

Comme nous l'avons indiqué, la base fournie permet la compatibilité avec l'ancien mode d'écriture des bruts, et donc la mise à niveau progressive des diverses composants de STAP. Le premier composant à migrer sera le module de traitement des remises, afin notamment de pouvoir stocker l'élément XML correspondant à une transaction. Par la suite, on étendra cette migration aux couches gérant les protocoles bancaires, en premier lieu CB2A, qui utilisera de manière native les fonctions d'écriture au format XML et n'aura donc plus recours à la structure de référence.

Chiffrement des données sensibles

PCI DSS demande le chiffrement des données sensibles, notamment les numéros de porteur (ou numéros de carte), afin que seules les personnes habilitées puissent y avoir accès. De plus, il serait bon de signer les bruts dès la réception, afin de l'authentifier et d'empêcher toute modification ultérieure.

Il se trouve que le W3C a émis des recommandations dans ce domaine [**25**, **26**], permettant de chiffrer ou de signer tout ou partie d'un document XML. Ces dernières sont implémentées dans la bibliothèque XMLSec, construite sur libxml2 et OpenSSL. À première vue, il semble pertinent d'introduire ces fonctionnalités dans STAP.

4.4 Bilan

Nous avons exposé dans ce chapitre le travail effectué pour implémenter l'écriture des bruts au format XML sans remettre brutalement en cause les fonctionnalités préexistantes. Nous offrons une base permettant une évolution progressive du reste de l'application, et introduisons une nouvelle conception du dictionnaire des données qui est désormais bien séparé de l'application elle-même, facilitant la gestion des données et l'évolutivité. Enfin, les fonctions de

lecture/écriture de bruts mises au point préparent également l'adoption de protocoles UNIFI tels qu'EPAS.

Conclusion

Θάλαττα ! Θάλαττα !

Ἀνάβασις, Ξενοφῶν

CE TFE a été l'occasion d'aborder des technologies et des concepts divers, relevant de domaines aussi dissemblables que les réseaux informatiques, la cryptographie ou les technologies gravitant autour de XML ; il a donné lieu pour chaque partie à des développements concernant divers composants des applications d'AFSOL, en particulier STAP, son serveur de télécollecte.

L'implémentation de la RFC 1086 est totalement aboutie. On regrette cependant l'apparition relativement tardive dans le processus de développement, de problèmes liés aux données d'appel, limitant l'utilisation du mode RFC1086 présenté à la section 2.3.1. Face à une offre réduite en termes de matériel X.25, et à l'absence de protocole prenant en compte les données d'appel telles qu'elles sont utilisées dans les premières versions de CB2A, il semble impossible de circonvenir à ces difficultés. Néanmoins, on peut se réjouir de leur disparition dans CB2A 1.2, en cours de déploiement.

L'introduction de SSL, nécessaire pour le mode EMULRFC1086 dont il a été question à la section 2.3.2, a permis de jeter les bases de son intégration dans STAP. De solides bases théoriques, indispensables à la mise en œuvre de cette couche cryptographique, ont été jetées ; les premiers essais impliquant une communication chiffrée grâce à SSL entre STAP et un TPE ont pu être menés à bien ; finalement, nous avons ouvert la voie en dressant un premier panorama de l'infrastructure qui devra être mise en place.

Enfin, nous avons pu fournir une base permettant d'introduire le format XML dans l'écriture des remises financières, tout en restant compatible avec le système actuel, dont les caractéristiques y étaient fort peu propices. De la sorte, la migration pourra s'effectuer de manière progressive, jusqu'à la disparition totale des fonctions d'écriture séquentielles.

Le monde de la « monétique » est en perpétuelle évolution. Les nouvelles fonctionnalités XML sont une première étape qui facilitera l'adoption du protocole EPAS dès sa publication. Elles permettront également le chiffrement des éléments contenant informations sensibles, demandé par PCI DSS, qui pourrait être réalisé grâce aux recommandations du W3C exposées à la section 4.3.8 de ce rapport.

Bibliographie

Amic, fugim tanta malòria,
on viu tot home a gratcient;
del camí a l'ombra fem memòria,
mirant la vinya i l'aire pacient,
i transcrivim cent i mil meravelles
al Llibre de les Set Sivelles.

El Llibre de les Set Sivelles
Josep Sebastià Pons

Monétique et protocoles bancaires

[1] Lyra Network : *Site de la société*, 2009. Disponible sur le web à l'adresse http://www.lyra-network.com.

[2] Organisation internationale de normalisation : *ISO 8583 – Messages initiés par carte de transaction financière - spécification d'échange de messages*, 1993.

[3] Organisation internationale de normalisation : *ISO 20022 – Services financiers – Schéma universel de messages pour l'industrie financière – parties 1 à 5*, 2004.

[4] Payment Card Industry Security Standards Council : *Payment Card Industry Data Security Standard (PCI DSS)*, octobre 2008. Disponible sur le web à l'adresse http://www.pcisecuritystandards.org/.

[5] SWIFT : *Site institutionnel*. Disponible sur le web à l'adresse http://www.swift.com/.

[6] UNIFI : *ISO 20022 UNIversal FInancial industry message scheme*, 2009. Disponible sur le web à l'adresse http://www.iso20022.org/.

X.25 et TCP/IP

[7] Fabio Busatto : *TCP Keepalive HOWTO*, 2007. Disponible sur le web à l'adresse http://tldp.org/HOWTO/TCP-Keepalive-HOWTO/index.html.

[8] Cisco Systems, Inc. : *Internetworking Technology Handbook*, 2008. Disponible sur le web à l'adresse http://www.cisco.com/en/US/docs/internetworking/technology/handbook/X25.html.

[9] J. FORSTER *et al.* : *RFC 1613 – cisco Systems X.25 over TCP (XOT).* The Internet Engineering Task Force, mai 1994. Disponible sur le web à l'adresse `http://tools.ietf.org/html/rfc1613`.

[10] J. ONIONS et M. ROSE : *RFC 1086 – ISO-TP0 bridge between TCP and X.25.* The Internet Engineering Task Force, décembre 1988. Disponible sur le web à l'adresse `http://tools.ietf.org/html/rfc1086`.

[11] ORGANISATION INTERNATIONALE DE NORMALISATION : *ISO/IEC 8208 – Technologies de l'information – Communication de données – Protocole X.25 de couche paquet pour terminal de données*, 2000.

[12] Marshall T. ROSE et Dwight E. CASS : *RFC 1006 – ISO Transport Service on top of the TCP.* The Internet Engineering Task Force, mai 1987. Disponible sur le web à l'adresse `http://tools.ietf.org/html/rfc1006`.

[13] UNION INTERNATIONALE DES TÉLÉCOMMUNICATIONS : *X.25 : Interface entre équipement terminal de traitement de données et équipement de terminaison de circuit de données pour terminaux fonctionnant en mode paquet et raccordés par circuit spécialisé à des réseaux publics pour données*, octobre 1996.

Cryptographie et SSL

[14] Christophe CACHAT et David CARELLA : *PKI Open Source : déploiement et administration.* O'Reilly France, 2003. O'Reilly France ayant fermé, cet ouvrage est épuisé. Il reste disponible dans de nombreuses bibliothèques universitaires. Jusqu'à il y a peu, il était possible de l'acquérir au format PDF sur `http://www.immateriel.fr`.

[15] Pravir CHANDRA, Matt MESSIER et John VIEGA : *Network Security with OpenSSL.* O'Reilly, 2002.

[16] DIRECTION CENTRALE DE LA SÉCURITÉ DES SYSTÈMES D'INFORMATION (DCSSI) : *Réglementation, section « cryptologie »*, 2008. Disponible sur le web à l'adresse `http://www.ssi.gouv.fr/fr/reglementation/index.html#crypto`.

[17] R. HOUSLEY, W. POLK, W. FORD et D. SOLO : *RFC 3280 - Internet X.509 Public Key Infrastructure - Certificate and Certificate Revocation List (CRL) Profile.* The Internet Engineering Task Force, avril 2002. Disponible sur le web à l'adresse `http://tools.ietf.org/html/rfc3280`.

[18] INTERNATIONAL TELECOMMUNICATION UNION : *X.509 : Technologies de l'information – interconnexion des systèmes ouverts – L'annuaire : cadre général des certificats de clef publique et d'attribut*, 2005.

[19] Alfred J. MENEZES : *Handbook of Applied Cryptography.* CRC Press, 5e édition, août 2001. Grâcieusement mis à disposition par les auteurs et l'éditeur sur le web à l'adresse `http://www.cacr.math.uwaterloo.ca/hac/`.

[20] Eric RESCORLA : *SSL and TLS : Designing and Building Secure Systems.* Addison-Wesley Professional, 2000.

Technologies XML

[21] Alexandre BRILLANT : *XML : Cours et exercices.* EYROLLES, 2007.

[22] Daniel MULLER : *Transformation et échange de données structurées*, 2009. Disponible sur le web à l'adresse `http://tic01.tic.ec-lyon.fr/~muller/cours/teds/`.

[23] Doug TIDWELL : *XSLT : Mastering XML Transformations, 2nd edition.* O'Reilly, 2008.

[24] Daniel VEILLARD *et al.* : *Reference Manual for libxml2*, 1999–2009. Disponible sur le web à l'adresse `http://xmlsoft.org/html/index.html`.

[25] W3C : *XML Encryption Syntax and Processing*, 10 décembre 2004. Disponible sur le web à l'adresse `http://www.w3.org/TR/xmlenc-core/`.

[26] W3C : *XML Signature Syntax and Processing (Second Edition)*, 10 juin 2008. Disponible sur le web à l'adresse `http://www.w3.org/TR/xmldsig-core/`.

[27] W3C : *XML Path Language (XPath) version 1.0*, 16 novembre 1999. Disponible sur le web à l'adresse `http://www.w3.org/TR/xpath`.

[28] W3C : *XSL Transformations (XSLT) version 1.0*, 16 novembre 1999. Disponible sur le web à l'adresse `http://www.w3.org/TR/xslt`.

[29] W3C : *XML Path Language (XPath) version 2.0*, 23 janvier 2007. Disponible sur le web à l'adresse `http://www.w3.org/TR/xpath20/`.

[30] W3C : *XSL Transformations (XSLT) version 2.0*, 23 janvier 2007. Disponible sur le web à l'adresse `http://www.w3.org/TR/xslt20/`.

[31] W3C : *XML Schema Part 0 : Primer Second Edition*, 28 octobre 2004. Disponible sur le web à l'adresse `http://www.w3.org/TR/xmlschema-0/`.

[32] W3C : *XML Schema Part 1 : Structures Second Edition*, 28 octobre 2004. Disponible sur le web à l'adresse `http://www.w3.org/TR/xmlschema-1/`.

[33] W3C : *XML Schema Part 2 : Datatypes Second Edition*, 28 octobre 2004. Disponible sur le web à l'adresse `http://www.w3.org/TR/xmlschema-2/`.

Autres

[34] Christophe BLAESS : *Programmation système en C sous Linux : signaux, processus, threads, IPC et sockets.* Eyrolles, 2 édition, 2005.

[35] Charles DONNELLY et Richard STALLMAN : *Bison - The Yacc-compatible Parser Generator.* Free Software Foundation, Inc., 2006. Disponible sur le web à l'adresse `http://www.gnu.org/software/bison/manual/index.html`.

[36] François KOEUNE et Jean-Jacques QUISQUATER : Carte à puce : un premier tour d'horizon. *MISC hors-série*, novembre/décembre 2008.

[37] Jeff LEE : *ANSI C grammar, Lex specification*, 1985. Disponible sur le web à l'adresse `http://www.lysator.liu.se/c/ANSI-C-grammar-l.html`.

[38] Jeff LEE : *ANSI C Yacc grammar*, 1985. Disponible sur le web à l'adresse `http://www.lysator.liu.se/c/ANSI-C-grammar-y.html`.

[39] ORGANISATION INTERNATIONALE DE NORMALISATION : *ISO/IEC 8824-4 – Technologies de l'information – Notation de syntaxe abstraite numéro un (ASN.1) : paramétrage de la notation de syntaxe abstraite numéro un*, 2002.

[40] The Flex Project : *Lexical Analysis with Flex*, 2007. Disponible sur le web à l'adresse http://flex.sourceforge.net/manual/.

Annexes

Annexe A

À propos de ce document

Ce document a été élaboré avec LaTeX et BIBTeX. Le code source a été édité avec Vim, accompagné du plugin Vim-Latex. Les illustrations ont été réalisées avec Dia (éditeur de diagrammes), Graphviz (générateur de graphes) et Inkscape (éditeur d'images vectorielles). L'ensemble de ces logiciels est libre, et de surcroît disponible gratuitement. Le PDF ainsi que les sources et un fichier d'aide sont mis à disposition sur le web à l'adresse `http://set-sivelles.cat/docs/TFE/`.

Pour générer le PDF, il est nécessaire d'installer LaTeX. Les « wikibooks » francophones et anglophones fournissent l'information nécessaire selon votre système : `http://fr.wikibooks.org/wiki/Programmation_LaTeX/Installer_LaTeX` et `http://en.wikibooks.org/wiki/LaTeX/Installation`.

Les paquetages suivants sont nécessaires : `inputenc`, `fontenc`, `lmodern`, `eurosym`, `babel`, `makeidx`, `glossaries`, `graphicx`, `bibtopic`, `subfig`, `listings`, `color`, `pifont`, `cite`, `epigraph`, `chngpage`, `lettrine` et `url` ou `hyperref` si l'on souhaite générer des liens hypertexte. La majorité est préinstallée, pour les autres il faudra les récupérer sur le CTAN (`http://www.ctan.org/`) et se reporter aux liens précédents. Le style bibliographique « plain » francisé est fourni avec les fichiers sources.

Sous un certain nombre de distributions GNU/Linux, LaTeX peut être déjà installé, modulo quelques paquetages. Sur les systèmes dérivant de Debian, comme la version d'Ubuntu utilisée sur les postes clients d'AFSOL, on peut installer le paquetage `texlive-full`, qui installera LaTeX avec toutes les extensions nécessaires. Prenez garde cependant, l'espace disque requis est relativement important !

Pour compiler, se placer dans le répertoire de base et taper la commande `make` sous un système Unix (y compris OS X) ou lancer `make.bat` sous Microsoft Windows.

Annexe B

Résumé des fonctions pour l'écriture des remises au format XML à l'usage du développeur

B.1 Fonctions afférentes aux remises XML

B.1.1 Structures de données définies dans brutX.h

Remise

```
typedef struct
{
	ST_DICO_BR_XML *dictionnaire;

	xmlDocPtr doc_brut;
	xmlXPathContextPtr ctx_brut;

	xmlXPathContextPtr ctx_gabarit;

	char *elt_entete;
	char *elt_transaction;

	xmlNodePtr entete;
	xmlNodePtr *transactions;
	int nbTrans;

	xmlNodePtr racine_doc;
	xmlNodePtr racine_gabarit;
	xmlNodePtr noeud_courant;
} ST_REMISEX;
```

L'élément principal d'une remise est `doc_brut`, qui contient l'arbre XML de la remise. `racine_doc` pointe vers sa racine. En cas de chargement, `entete` pointe sur le nœud en-tête, et `transactions` est un tableau de nœuds XML pointant vers les nœuds transaction, de taille `nbTrans`. L'arbre XML du gabarit* ne se trouve pas ici, mais dans le dictionnaire. Néanmoins `racine_gabarit` est toujours renseigné (il doit y avoir moyen de s'en passer, cela est un reste d'une ancienne version du programme mais est encore demandé par une fonction de brutX).

Pour les recherches XPath, on a besoin d'un contexte libxml2 : `ctx_brut` pour la remise (utile lorsqu'on cherche une information particulière dans la remise), et `ctx_gabarit` pour le gabarit (ne sert qu'en mode « manuel », `remplitAuto` n'en a pas besoin).

Le nœud courant sert à indiquer à quel endroit dans l'arbre effectuer certaines opérations. Par exemple, lorsqu'on appelle `remplitAuto`, il doit pointer vers l'en-tête ou la transaction en cours, ou bien lorsqu'on appelle `remplirX`, il est souvent nécessaire de savoir à quelle transaction (par exemple), on doit rajouter tel nœud.

`elt_entete` et `elt_transaction` contiennent les noms des éléments correspondant à l'en-tête et à une transaction. Ceux-ci sont renseignés dans une annotation `role` dans le schéma, se retrouvent dans `corpus.xml`, et sont donc chargés en même temps que le corpus. Dans le cas du schéma `etalon.xsd`, il s'agit respectivement de `Remise` et `Transaction`.

Objets définis en global

Un « objet » remise est placé en global dans `brutX.h`. Cela est nécessaire tant que l'on souhaite assurer la compatibilité avec les fonctions d'écriture et lecture de bruts de `readB.c` et `writeB.c`, dans la mesure où on ne peut pas se permettre de changer leurs arguments (à moins de faire passer un pointeur sur une remise pour un `int` (descripteur de fichiers), ce qui est envisageable, mais « limite »...).

Un corpus (voir ci-dessous) est également déclaré en global dans le même fichier, mais il me semble que cela n'est utile que pour les fonctions d'ouverture de bruts (qui doivent être modifiables). Normalement, toutes les fonctions nécessitant le corpus le prennent en argument. Rappelons que l'idéal serait que le corpus fût chargé une bonne fois pour toutes.

B.1.2 Fonctions définies dans brutX.c

creerRemiseX

```
ST_REMISEX* creerRemiseX(const ST_CORPUS_BR_XML *corpus, const char
    *nomSchema)
```

Crée une remise XML, comme son nom l'indique. Le corpus doit avoir été chargé, nomSchema est le schéma XML correspondant au type de remise (par exemple `etalon` pour CB2A).

Cette fonction renvoie `NULL` en cas d'erreur, un pointeur vers la remise sinon. Celle-ci devra être libérée avec `libererRemiseX` (après sauvegarde avec `writeX`).

chargerRemiseX

```
ST_REMISEX* chargerRemiseX(const ST_CORPUS_BR_XML *corpus, const int
    fdX, const char *bufferX, const int taille_buffer, const char *
    fichierX)
```

Cette fonction charge une remise existante, connaissant le corpus, à partir d'un descripteur de fichier, d'un tampon ou d'un fichier.
- `fdX` : le descripteur de fichier correspondant au fichier ouvert, ou -1 si non renseigné ;
- `bufferX`, `taille_buffer` : le tampon contenant le XML et sa taille (ou `NULL` et 0 si non renseignés) ;
- `fichierX` : le nom de fichier ou NULL si non renseigné.

La fonction reconnaît le type de remise grâce au nom de schéma indiqué dans le XML : l'attribut `xsi:noNamespaceLocation="etalon.xsd"` indique par exemple que le fichier est une instance du schéma etalon.xsd. Si cet attribut n'est pas renseigné, une erreur est renvoyée.

Un pointeur sur chacun des nœuds transactions est conservé dans le tableau `transactions` de `ST_REMISEX`.

Comme précédemment, renvoie la remise allouée, ou `NULL` en cas d'erreur.

InitRemiseX

```
ST_REMISEX* InitRemiseX(const ST_CORPUS_BR_XML *corpus, const char *
    nomSchema, xmlDocPtr doc_brut)
```

Cette fonction est appelée par `creerRemiseX` et `chargerRemiseX`. Son rôle principal est d'initialiser les différents champs de la structure `ST_REMISEX` qu'elle alloue, et de chercher les éléments pertinents dans le corpus.

Si `doc_brut` vaut `NULL`, on crée une remise vide, sinon on utilise l'arbre XML vers lequel il pointe (cas de chargement).

libererRemiseX

```
void libererRemiseX(ST_REMISEX* remise)
```

Libère une remise. Attention, certains éléments ne doivent pas être libérés, dans la mesure où ils ne font que pointer ; ainsi, `dictionnaire` n'est qu'un pointeur vers un dictionnaire du corpus.

nouvelleTransactionX

```
xmlNodePtr nouvelleTransactionX(ST_REMISEX *remise)
```

Alloue de la place pour une nouvelle transaction dans le tableau `transactions`, et crée un nouveau nœud transaction qui devient le nœud courant. Renvoie `NULL` en cas d'erreur, ou le nouveau nœud courant.

Détail de programmation : j'ai appris (un peu tard) que `realloc` est assez gourmand, et qu'il ne faut pas l'utiliser pour réallouer de la mémoire de

manière incrémentale. Il vaut mieux adopter un schéma exponentiel, de manière à converger rapidement vers un espace mémoire suffisant (par exemple en doublant à chaque fois la mémoire allouée). Bien sûr, il ne faut plus alors se contenter de compter les remises : il devient nécessaire de garder une trace de la mémoire allouée.

writeX

```
int writeX(ST_REMISEX *remise, int fdX, xmlBufferPtr bufferX, char *
    fichierX, int validation)
```

Sauvegarde la remise dans un fichier connaissant son nom ou un descripteur de fichier associé, ou dans un tampon de type `xmlBufferPtr`.

- `fdX` : -1 si non renseigné ;
- `bufferX` : `NULL` si non renseigné ;
- `fichierX` : `NULL` si non renseigné.
- `validation` : si nul, demande à ce que la remise soit validée par rapport à son schéma associé (recommandé).

Valeur renvoyée :

- -1 si erreur (y compris erreur interne de libxml2) ;
- 0 si OK sans validation ;
- 0 si sauvegarde *et* validation OK si demandée ;
- valeur strictement positive si validation demandée, et le document n'est *pas* valide.

getChampX

```
xmlChar* getChampX(ST_REMISEX *remise, char *champ)
```

Permet de récupérer le contenu de l'élément `champ` (Expression XPath) de `remise`, le nœud courant étant correctement positionné si besoin est (par exemple si `champ` est une expression XPath relative, comme `//numRemise`).

remplirX

```
int remplirX(ST_REMISEX *remise, char *champ, char *valeur)
```

En fonction du nœud courant, attribue `valeur` à un nœud désigné par l'expression XPath `champ`, qui sera créé (ainsi que certains de ces ancêtres le cas échéant). Cette fonction cherche le nœud correspondant dans le gabarit*, et fait appel à `accrocher` pour le répliquer dans l'arbre XML de la remise.

accrocher

```
static xmlNodePtr accrocher(xmlNodePtr noeud, xmlNodePtr racine,
    xmlNodePtr arbre, int creation)
```

```
{
    /* Si erreur renvoyee par niveau de recursion precedent ou on est arrive
         a la racine de l'arbre de depart sans trouver ou y placer l'
        homologue de noeud */
    if(arbre == NULL || noeud == racine) return NULL;

    /* c'est le bon moment pour accrocher l'homologue de noeud dans l'arbre
         d'arrivee */
    if (!xmlStrcmp(noeud->parent->name, arbre->name))
        return fils(arbre, noeud, creation);

    /* Accrocher le noeud courant au noeud accroche un cran plus haut */
    return accrocher(noeud, racine,
                    accrocher(noeud->parent, racine, arbre, 0),
                        creation);
}
```

Réplique `noeud` dans le sous-arbre du nœud `arbre` (`racine` est la racine de l'arbre de départ, et permet de gérer un cas d'erreur). Si `creation` n'est pas nul, créer un nouveau nœud même s'il en existe déjà une instance.

`accrocher` cherche récursivement dans l'arbre de la remise le père du nœud qu'elle souhaite accrocher. Si elle ne le trouve pas, elle cherche d'abord à accrocher le père de `noeud` et ainsi de suite sur autant de générations que nécessaire. Lorsque le programme dépile, les nœuds fils sont accrochés au fur et à mesure.

Lorsqu'elle a trouvé le père, `accrocher` appelle `fils` pour placer `noeud` dans le bon ordre dans la fratrie.

fils

```
static xmlNodePtr fils(xmlNodePtr pere, xmlNodePtr fils_depart, int
    creation)
```

Rajouter un nœud portant le même nom que `fils_depart` (normalement un nœud du gabarit) au nœud `père`. Le rang dans la fratrie de `fils_depart` est respecté. Si `creation` est non nul, forcer la création même s'il existe déjà une instance de `fils_depart` dans la fratrie d'arrivée.

Cette fonction peut être simplifiée en commençant par chercher le rang de `fils_depart` dans sa fratrie, et en le comparant au rang de fratrie des fils de `pere` déjà existants.

`fils` est appelée par `accrocher`, dès qu'elle a trouvé dans l'arbre de la remise le père du nœud qu'elle souhaite accrocher.

remplitAuto

```
int remplitAuto(ST_REMISEX *remise, char *nomstruct, void *structure,
    int taille)
```

Le nœud courant étant correctement renseigné, `remplitAuto` parcourt le dictionnaire à la recherche des définitions correspondant à la structure `nomstruct`,

récupère les informations présentes dans `structure` et les utilise pour générer une partie de l'arbre XML de `remise`. `taille` est la taille de la structure.

Exemple d'appel :

```
remplitAuto(remise, "br", (void*)&br, sizeof(br))
```

permet de remplir une nouvelle transaction à partir des informations présentes dans `br`. Dans ce cas, il faut d'abord créer une nouvelle transaction avec la fonction `nouvelleTransactionX` sous peine d'écraser l'ancienne transaction.

Le format des éléments est modifié suivant le type déclaré dans le schéma et le cas échéant l'annotation `formatBrutStruct` :

- si l'élément est déclaré de type numérique, on supprime les zéros initiaux.
- s'il s'agit d'une date, on la formate en utilisant `formatBrutStruct`. Dans ce cas, deux champs de la structure peuvent donner à l'arrivée un seul élément. Par exemple, les dates étant généralement écrites au format `dateTime` dans le XML, `12012009` (12 janvier 2009) et `085431` (huit heures cinquante quatre minutes et 31 secondes) peuvent donner `2009-01-12T-08:54:31+01:00`.

xmlToStruct

```
int xmlToStruct(ST_REMISEX *remise, char *nomstruct, void *structure,
    int taille)
```

xmlToStruct réalise l'opération inverse : remplir la structure de référence à partir des éléments présents dans le XML. Les types numériques sont laissés tels quels (on ne replace pas les zéros initiaux), dans la mesure où la fonction `atoi` devrait les lire de la même manière.

B.2 Fonctions de chargement du dictionnaire

B.2.1 Structures de données définies dans dicoX.h

C'est dans ces structures de données que l'on place les informations contenues dans `corpus.xml`. Un corpus est composé de plusieurs dictionnaires (schémas XML annotés), eux-mêmes composés de groupes de définitions (un par structure de référence, donc trois en tout, pour l'en-tête, les transactions, et le pied), qui comportent des définitions (informations afférentes à un élément XML en particulier). Pour ce dernier on renseigne un ou deux éléments dans la structure de référence.

Corpus (ST_CORPUS_BR_XML)

```
typedef struct
{
        ST_DICO_BR_XML **dictionnaires;
        int nb_dictionnaires;
} ST_CORPUS_BR_XML;
```

Un corpus n'est qu'un ensemble de dictionnaires.

Dictionnaire (ST__DICO__BR__XML)

```
typedef struct
{
        xmlChar *nom_schema;
        xmlChar *noeud_entete;
        xmlChar *noeud_transaction;

        ST_GROUPE_DEF_BR_XML **groupes_definitions;
        int nb_groupes_definitions;

        xmlSchemaPtr schema;
        xmlDocPtr gabarit;
} ST_DICO_BR_XML;
```

Un dictionnaire correspond à un unique schéma, dont on récupère le nom (`nom_schema`), présent dans `corpus.xml`, et que l'on va analyser de manière à obtenir l'objet `xmlSchemaPtr schema`. Du schéma découle un gabarit*, que l'on charge dans `gabarit`. On récupère dans `corpus.xml` les noms des nœuds transaction et en-tête (`noeud_entete` et `noeud_transaction`). Il contient un tableau de pointeurs vers les groupes de définitions qui le composent.

Groupe de définitions (ST__GROUPE__DEF__BR__XML)

```
typedef struct
{
        xmlChar *nom_struct;
        ST_DEF_BR_XML **definitions;
        int nb_definitions;
} ST_GROUPE_DEF_BR_XML;
```

Un groupe de définitions correspond à une structure de référence nommée `nom_struct`, et est composé de définitions.

Définition (ST__DEF__BR__XML)

```
typedef struct
{
        ST_CHAMP_STRUCT_BR_XML **champs_struct_ref;
        int nb_champs_struct_ref;

        xmlChar *nom_element;
        enum TYPE_BASE_XSD typeBaseXSD;

        xmlChar *cheminGabarit;
        xmlNodePtr noeud_gabarit;
} ST_DEF_BR_XML;
```

La définition d'un élément XML tient de `corpus.xml` le nom de l'élément, son type de base dans le schéma XML (par exemple `xs:date`), utile pour le formatage

des données, mais aussi le chemin de l'élément correspondant dans le gabarit*
à la mode XPath, tel que /Remise/Transactions/Transaction/Montant1. Ce chemin permet, lors du chargement de la définition, de récupérer un pointeur vers ce nœud, noeud_gabarit, ainsi on n'a plus à effectuer de recherche lors de la génération d'un élément de la remise, puisqu'on a déjà le nœud du gabarit sous la main ! Pour un élément, on renseigne également un ou deux champs de la structure de référence, contenus dans le tableau champs_struct_ref.

Champs de la structure ST_CHAMP_STRUCT_BR_XML

```
typedef struct
{
        int offset;
        int longueur;
        xmlChar *formatBrutStruct;
} ST_CHAMP_STRUCT_BR_XML;
```

Un champ dans la structure de référence est défini par son décalage offset dans la structure, et sa longueur. Éventuellement (en fait, uniquement dans le cas d'une date), on renseigne son format, par exemple hhmmss pour une heure.

B.2.2 Fonctions définies dans dicoX.c

Fonctions de chargement

```
ST_CORPUS_BR_XML *chargeCorpus();
static int chargeDicos(xmlNodePtr noeud_corpus, ST_CORPUS_BR_XML *
    corpus);
static ST_DICO_BR_XML *chargeDico(xmlNodePtr noeud_dico)
static int chargeGroupesDefs(xmlNodePtr noeud_dictionnaire,
    ST_DICO_BR_XML *dictionnaire)
static ST_GROUPE_DEF_BR_XML *chargeGroupeDef(xmlNodePtr
    noeudGroupeDef)
static int chargeDefs(xmlNodePtr noeud_groupe_def,
    ST_GROUPE_DEF_BR_XML *groupe_def)
static ST_DEF_BR_XML *chargeDef(xmlNodePtr noeudDef)
static int chargeChampsStructs(xmlNodePtr noeud_def, ST_DEF_BR_XML*
    def)
static ST_CHAMP_STRUCT_BR_XML *chargeChampStruct(xmlNodePtr
    noeudChampStruct)
```

Charger chacune des entités formant le corpus. Corpus essaie d'abord de charger un groupe de dictionnaires en appelant chargeDicos, cette dernière fait appel à chargeDico pour charger chacun des dictionnaires, etc. Cette fragmentation permet par entité permet de mieux gérer les cas d'erreur (et notamment les libérations de mémoire).

Fonctions de libération

```
void libererCorpus(ST_CORPUS_BR_XML *corpus)
static void libererDico(ST_DICO_BR_XML *dico)
static void libererGroupeDef(ST_GROUPE_DEF_BR_XML *groupe_def)
static void libererDef(ST_DEF_BR_XML *def)
static void libererChampStruct(ST_CHAMP_STRUCT_BR_XML *champ_struct
    )
```

Ces fonctions libèrent chacune des entités correspondantes. `libererCorpus` est la seule qui puisse être utilisée par le reste du programme, les autres sont bien utiles pour libérer ce que l'on était en train de construire lorsque survient une erreur.

Fonction typeXSDenInt

```
static int typeXSDenInt(xmlChar *typeBrutXSD)
```

Transforme un type de base des schémas W3C en un entier (décrit dans une structure enum dans `xmlfct.h`, et ce afin d'éviter les comparaisons de chaînes lors du traitement du brut (changement de format en fonction du type de base, voir `remplitAuto`).

B.3 Fonctions d'ouverture/fermeture et de lecture/écriture par blocs des bruts

B.3.1 Fonctions d'ouverture/fermeture dans fctB.c

Les fonctions de `readB.c` et `writeB.c` écrivaient bloc par bloc en utilisant le descripteur de fichier fourni. Cela n'est plus possible : on ne peut pas écrire du XML au fur et à mesure que nous parvient l'information, non seulement parce que les formes qu'il revêt en mémoire (il s'agit d'un arbre) et sur disque (fichier résultant de la sérialisation de l'arbre) sont complètement différentes, mais aussi parce qu'en cas d'interruption on n'obtiendrait pas du XML (ne serait-ce que du fait que le document ne serait pas *bien formé**).

On est obligé de créer une remise XML à l'ouverture du brut, et de la sérialiser à la fermeture. Il faut donc obligatoirement utiliser des fonctions d'ouverture et fermeture, qui vont s'en charger ; il n'est plus possible de se contenter d'obtenir un descripteur de fichier avec `open` et de le fermer avec `close`.

Les fonctions de `readB.c` et `writeB.c` se contentent désormais de faire appel à `remplitAuto` et `xmlToStruct` (voir ci-dessous) et n'écrivent donc plus bloc par bloc (pour les protocoles migrés en XML).

typ_brut2

Charge une remise existante, et renvoie son type. S'il s'agit d'une remise XML, celle-ci est chargée, et on renvoie le type `BR_CB_XML`. S'il s'agit d'un brut à l'ancien format pour lequel un schéma XML est disponible, on réalise la conversion.

openf_x

Crée une remise (modes `w` et `w+`), correspondant au type demandé.

close_brut

`close_brut` enregistre la remise XML au moyen du descripteur de fichier fourni s'il y a lieu (le protocole est migré en XML et le descripteur correspond à un fichier ouvert en écriture), puis le ferme et renvoie le résultat de `close`.

Fonction de conversion de bruts

Lorsque `typ_brut2` rencontre un brut à l'ancien format, elle appelle cette fonction pour le convertir en XML s'il y a lieu, à grand renfort de `remplitAuto`. L'ancien brut est renommé en `.old`. Attention : le processus utilise un fichier temporaire (en `.tmp`), placé dans `/tmp`, qui n'est effacé que si l'opération réussit. Ce chemin n'est peut-être pas le meilleur, il vaudrait peut-être mieux utiliser le chemin courant, comme pour le `.old`.

B.3.2 Modification des fonctions de lecture et d'écriture par blocs dans readB.c et writeB.c

Fonctions de lecture

`read_entete`, `read_trans` et `read_pied`, pour les formats migrés en XML (auxquels on assigne le type `BR_CB_XML`), se contentent de faire appel à `xmlToStruct` pour charger le contenu du XML dans la structure de référence, après avoir pris soin de positionner correctement le nœud courant sur l'en-tête ou un nœud transaction. On utilise un compteur pour savoir combien de transactions ont été lues.

Fonctions d'écriture

`write_entete()`, `write_trans()` et `write_pied()` font appel à `remplitAuto` pour placer le contenu dans la structure de référence dans l'arbre XML, après avoir positionné correctement le nœud courant. `write_trans` appelle `nouvelleTransactionX` pour créer une nouvelle transaction, qui se charge également de positionner le nœud courant sur la transaction créée.

B.4 Fonctions ajoutées dans l'API

B.4.1 Fonctions ajoutées dans xmlfct.h

Fonctions de recherche XPath

Ces fonctions résultent de l'encapsulation de la fonction `xmlXPathEvalExpression` de libxml2.

Xml_XPath

```
xmlXPathObjectPtr Xml_XPath(xmlDocPtr doc, const char *expression,
    char *prefixeNs, char *Ns)
```

Effectue une recherche XPath dans l'arbre `doc`, selon l'`expression` fournie. Renvoie un objet `xmlObjectPtr` [1] (qu'il *faudra* libérer avec `xmlXPathFreeObject`) ou `NULL` si erreur.

Dans le cas où les noms des balises comportent un préfixe, correspondant à un espace de noms, il faut « enregistrer » ce dernier en fournissant le préfixe et l'espace de noms associé, faute de quoi XPath ne pourra pas voir ces éléments. Par exemple, on peut réaliser l'appel :

```
obj = Xml_XPath(doc, "//xs:element", "xs",
    "http://www.w3.org/2001/XMLSchema"
```

Si l'on ne souhaite pas utiliser d'espace de noms, il suffit de positionner les deux dernières variables à `NULL`.

Xml_XPath_CTX

```
xmlXPathObjectPtr Xml_XPath_CTX(xmlXPathContextPtr ctx, const char *
    expression)
```

Réalise la même opération que la fonction précédente, à ceci près que l'on ne fournit plus un arbre XML, mais ce que libxml2 appelle un *contexte* XPath, obtenu avec la fonction `xmlXPathNewContext`. Cette méthode est censée être plus efficace [2]. On la préférera dès lors que l'on doit effectuer plusieurs recherches sur un même document. De même, l'objet renvoyé doit être libéré avec `xmlXPathFreeObject`.

Si l'on souhaite utiliser des préfixes correspondant à un espace de noms, il faut l'enregistrer soi-même avec la fonction `xmlXPathRegisterNs`, par exemple de la manière suivante [3] :

```
xmlXPathRegisterNs(ctx, "xs", "http://www.w3.org/2001/XMLSchema")
```

où ctx est un contexte XPath ; `xmlXPathRegisterNs` renvoie -1 en cas d'erreur et 0 si l'opération réussit.

Xml_XPathUnique et Xml_XPathUnique_CTX

```
xmlNodePtr Xml_XPathUnique(xmlDocPtr doc, const char *expression,
    char *prefixeNs, char *Ns)
xmlNodePtr Xml_XPathUnique_CTX(xmlXPathContextPtr ctx, const char *
    expression)
```

1. voir la documentation de libxml2
2. Garder à l'esprit toutefois que XPath n'est pas vraiment optimisé dans libxml2, comme le reconnaissent les développeurs eux-mêmes.
3. après avoir inclus `xpathInternals.h` en plus de `xpath.h`

Ces fonctions sont semblables aux deux précédentes, à ceci près qu'elles renvoient non plus un objet XPath, mais un nœud XML. À utiliser dans des cas où on n'est censé trouver qu'un seul nœud. Elles renvoient le nœud escompté en cas de succès (qu'il ne faudra *pas* libérer : il ne fait que *pointer* vers un nœud de l'arbre), ou `NULL` en cas d'erreur ou si le nœud est absent ou présent plus d'une fois.

Réécriture de fonctions libxml2 absentes de la version utilisée durant le développement

Ces fonctions sont présentes dans ce qui est censé tenir lieu de documentation à libxml2, accessible à l'adresse `http://xmlsoft.org`, mais n'existaient pas dans la version utilisée en développement (antérieure à la version 2.7.2). Elles ont donc été réécrites et soumises à une directive de compilation portant sur le numéro de version de libxml2.

xmlFirstElementChild

```
#if LIBXML_VERSION < 20702
xmlNodePtr xmlFirstElementChild(xmlNodePtr parent)
#endif
```

Renvoie le premier nœud *élément* fils du nœud `parent`. Renvoie `NULL` si erreur ou si `parent` n'a pas parmi ses fils de nœud élément.

xmlNextElementSibling

```
#if LIBXML_VERSION < 20702
xmlNodePtr xmlNextElementSibling(xmlNodePtr aine)
#endif
```

Renvoie le prochain frère de `aine` qui soit un nœud *élément*. Renvoie `NULL` si un tel frère est introuvable, ou si une erreur survient.

Formatage des éléments du brut

brutVersXSD

```
char *brutVersXSD(char *valeur, enum TYPE_BASE_XSD type, char *
    formatBrut)
```

Met en forme une information du brut présente dans la structure de référence selon son type dans le schéma, l'API xmlschemastypes de libxml2 étant inutilisable a cette date (02/09). Cette fonction est utilisée par `remplitAuto` pour convertir une information de la structure de référence avant de l'ajouter à l'arbre XML de la remise.

Prend en charge :

- token, normalizedString, string ;

– nonPositiveInteger, nonNegativeInteger, negativeInteger, int, positiveInteger ;
– duration, datetime, time, date ;

L'appelant doit libérer la chaine retournée.

Retourne `NULL` en cas d'erreur.

Voir `http://www.w3.org/TR/xmlschema-2/#built-in-datatypes` pour plus d'information sur les types de base des schémas W3C.

XSDVersBrut

```
char *XSDVersBrut(char *valeur, enum TYPE_BASE_XSD type, char *
    formatBrut)
```

Cette fonction réalise l'opération inverse : formater une donnée de l'arbre XML pour la placer dans la structure de référence.

Validation avec un schéma W3C

```
int Xml_valid(xmlDocPtr doc, xmlSchemaPtr schema)
```

Valide l'arbre XML `doc` avec l'objet `schema` de type `xmlSchemaPtr` (voir l'API `xmlschemas` de libxml2). Renvoie -1 en cas d'erreur, 0 si le document est valide et une valeur strictement positive s'il est invalide.

B.4.2 Fonctions ajoutées dans dthe_fct.h

dateVersDateTimeXSD

```
char* dateVersDateTimeXSD(char *date, char *formatDate, char *fuseau)
```

Convertit une date dont on spécifie le format avec la syntaxe de `ChaineEnDate` en `dateTime` des schémas XML (sous-ensemble d'ISO 8601). Si fuseau est `NULL`, on utilise le fuseau horaire du serveur (ce cas est à tester plus avant). S'il est renseigné, il doit être de la forme `(+|-)hh[:]mm` ou `Z` (équivalent de 00:00), ou encore égal à la chaîne nulle si on ne souhaite pas l'afficher.

Renvoie la chaîne contenant la date sous la forme ISO 8601, ou `NULL` si une erreur survient.

Exemples :

```
dateVersDateTimeXSD("22041984080500", "JJMMAAAAHHMMSS", "+02:00")
```

renvoie `1984-04-22T08:05:00+02:00`, et :

```
dateVersDateTimeXSD("22/04/1984-08:05:00", "JJ/MM/AAAA-hh:mm:ss",
                    "")
```

renvoie `1984-04-22T08:05:00`.

Attention : `date` et `formatDate` sont vérifiés (ce qui génère des appels en plus...), mais *pas* `fuseau` si fourni. À l'avenir, penser à utiliser `strptime` [4], qui

4. malheureusement, son usage nécessite l'utilisation d'une directive du préprocesseur qui impactait les fonctions de date déjà utilisées dans dthe_fct.c. Il faut donc envisager une réécriture de ce fichier, ou bien de le scinder. Voir le chapitre correspondant dans [**34**]

réalise en gros le travail de ChaineEnDate, mais de manière plus générique (et qui permet d'éviter les fonctions suivantes qui ne sont que des répétitions de celle-ci).

Fonctions dérivées

```
char* dateVersDateXSD(char *date, char *formatDate)
char* heureVersTimeXSD(char *date, char *formatDate, char *fuseau)
char* dateXSDVersDate(char *date, char *formatDate)
char* timeXSDVersHeure(char *date, char *formatDate)
```

Comme nous venons de l'expliquer, les fonctions ci-dessus pourraient être évitées si l'on pouvait utiliser conjointement `strftime` et `strptime`. `dateVersDateXSD` convertit une date de la structure de référence en date ISO 8601 ; ainsi, `22041984`, correspondant au format `JJMMAAAA` sera converti en `1984-04-22`. `heureVersTimeXSD` fait de même pour l'heure ; `0805000`, que l'on déclare être au format `HHMMSS` est converti en `08:05:00+02:00`, par exemple si le fuseau par défaut est `+02:00`.

Les deux dernières fonctions réalisent l'opération inverse (mise au format pour la structure de référence). Dans ce cas, il n'est plus question de fuseau horaire, et `formatDate` ne sert plus à reconnaître le format (on sait qu'il s'agit de `xs:date` ou de `xs:time`), mais à savoir comment transformer la donnée pour la structure de référence.

Annexe C

Génération de certificats X.509 avec OpenSSL

Cette annexe présente les manipulations les plus courantes permettant la génération de certificats avec OpenSSL, grâce auxquelles on peut mettre sur pied un embryon d'ICP. Cela permet d'approfondir notre approche de SSL, mais peut aussi être utile en phase de tests.

C.1 Création d'une autorité de certification

C.1.1 Environnement de l'autorité de certification

Créer un environnement pour l'autorité de certification :

```
mkdir /opt/exempleCA
cd /opt/exempleCA
mkdir certs private
chmod g-rwx,o-rwx private
echo '01' > serial
touch index.txt
```

- Le répertoire `exempleCA` contiendra les fichiers de l'autorité de certification.
- Le répertoire `private` contiendra la clef privée de l'AC. Il est impératif de restreindre les permissions, dans l'idéal il faudrait qu'elle soit stockée à part. La perdre compromet tous les certificats émis, mais aussi les flux chiffrés dans le passé !
- Le fichier `serial` contient le numéro de série du dernier certificat signé par l'AC, il est incrémenté à chaque nouveau certificat. `index.txt` contient la liste des certificats émis. Il doit exister lors de la signature du premier certificat.

C.1.2 Configuration de l'autorité de certification

Écrire le fichier de configuration de l'AC :

```
[ ca ]
default_ca = exampleca
[ exampleca ]
dir            = /opt/exampleca
certificate    = $dir/cacert.pem
database       = $dir/index.txt
new_certs_dir  = $dir/certs
private_key    = $dir/private/cakey.pem
serial         = $dir/serial
default_crl_days = 7
default_days   = 365
default_md     = md5
policy         = exampleca_policy
x509_extensions = certificate_extensions
[ exampleca_policy ]
commonName           = supplied
stateOrProvinceName  = supplied
countryName          = supplied
emailAddress         = supplied
organizationName     = supplied
organizationalUnitName = optional
[ certificate_extensions ]
basicConstraints = CA:false
[ req ]
default_bits     = 2048
default_keyfile  = /opt/exampleca/private/cakey.pem
default_md       = sha1
prompt           = no
distinguished_name = root_ca_distinguished_name
x509_extensions  = root_ca_extensions
[ root_ca_distinguished_name ]
commonName        = Example CA
stateOrProvinceName = Virginia
countryName       = US
emailAddress      = ca@exampleca.org
organizationName  = Root Certification Authority
[ root_ca_extensions ]
basicConstraints = CA:true
```

Explication pour chaque section :

- La section `[ca]` définit le nom de l'autorité de certification par défaut (ici `exampleca`) ; il peut donc y en avoir plusieurs dans un même fichier de configuration ;
- [`exampleca`] précise le nom des fichiers, notamment le certificat de l'AC (`cacert.pem`), la clef privée (`cakey.pem`), et le répertoire dans lequel seront copiés les certificats signés (`certs`). Attention, les variables telles que « `dir` » sont locales à la section dans laquelle elles sont déclarées. Par ailleurs, la CRL est mise à jour toutes les semaines (ce qui est un peu long…), les certificats émis sont valables un an par défaut et l'algorithme

de hachage par défaut est sha1 (préférable à md5 de nos jours) ;
- [exampleca_policy] précise quelles valeurs doivent être présentes dans le certificat de l'AC ; celles-ci sont précisés dans la section req. La directive prompt=no force OpenSSL à acquérir ces informations à partir du fichier, et non de manière interactive ;
- enfin, basicConstraints=CA:true signifie que ce certificat peut servir à signer d'autres certificats. Un certain nombre de paramètres permettent un contrôle plus fin sur la chaîne de certificats ; ainsi keyCertSign permet-il de borner sa longueur.

Pour que ce fichier soit pris en compte à l'étape suivante, placer le chemin absolu du répertoire contenant l'AC dans la variable OPENSSL_CONF (il peut aussi être précisé en ligne de commandes avec l'option -config) :

```
export OPENSSL_CONF=/opt/exempleCA
```

Créer le certificat de l'AC

```
openssl req -x509 -newkey rsa -out cacert.pem -outform PEM
```

La nouvelle clef générée sera de type RSA et sera longue de 2048 bits, tel que spécifié dans le fichier de configuration. Pour une clef d'autorité de certification, 2048 est à l'heure actuelle une longueur correcte, sans plus. On peut aller jusqu'à 4096 ou 8192. Pour une clef de certificat usuel, 1024 ou 2048 sont de bons compromis.

C.2 Signature de certificat

Le cas échéant, placer la demande de certificat dans le répertoire de l'AC et renseigner la variable OPENSSL_CONF. Il ne reste plus qu'à lancer la commande :

```
openssl ca -in testreq.pem
```

et à faire parvenir le certificat signé au demandeur.

C.3 Génération d'une demande de signature de certificat

Une tierce personne souhaite effectuer une demande de signature de certificat. Il est possible d'utiliser la commande suivante à cette fin :

```
openssl req -newkey rsa:1024 -keyout testkey.pem -keyform PEM -out
    testreq.pem
```

Ne pas oublier de spécifier la longueur de la clef ! La valeur par défaut pourrait ne pas convenir. Dans le cas où on effectue soi-même cette opération, il faut supprimer la variable OPENSSL_CONF afin d'éviter que soit pris en compte le fichier de configuration de l'AC :

```
unset OPENSSL_CONF
```

C.4 Divers

C.4.1 Retirer une passphrase

Normalement, les scripts générant les certificats retirent les passphrases* des clefs privées (sauf évidemment celle de l'autorité de certification). On peut voir la différence en affichant le fichier contenant la clef privée. Une clef privée avec passphrase contiendra le mot « `ENCRYPTED` » avant le bloc de données en base 64. Pour retirer la passphrase, utiliser la commande :

```
openssl rsa -in clef_avec_passphrase -out
    fichier_ou_mettre_la_clef_sans_passphrase
```

C.4.2 Créer un conteneur PKCS#12

```
openssl pkcs12 -export -in certificat.pem -inkey clef.pem -certfile
     CA.pem -out certificat.p12 -name "Certificat TPE"
chmod 600 certificat.p12
```

À partir d'un certificat et de la clef privée correspondante au format PEM (qui peuvent éventuellement se trouver dans le même fichier), on exporte dans un fichier au format PKCS#12 (extension p12). OpenSSL demande la passphrase que l'on souhaite utiliser pour chiffrer le certificat. Suivant les TPE, on ne devrait pouvoir mettre que des chiffres. L'option `-name` est optionnelle. L'option certfile permet d'inclure un certificat supplémentaire dans le fichier PKCS#12, par exemple celui de l'autorité de certification. Il est en effet indispensable que la totalité de la chaîne de certificats soit présente dans le fichier.

Annexe D

Utilisation de stunnel

D.1 Généralités

stunnel gère le chiffrement des communications au moyen de certificats SSL. Dans notre cas, il permet d'ajouter une couche cryptographique à STAP sans en modifier le code source. Nous nommerons ce service staps (par analogie avec pops, imaps...) et lui attribuerons le port 7345.

stunnel se charge de réceptionner le flux SSL sur le port 7345, de le décrypter et de le renvoyer en clair sur le port 6504. On peut choisir de faire tourner stunnel sur la même machine que STAP, ou bien sur une machine dédiée, auquel cas il faut bien évidemment s'assurer que le lien entre les deux machines est absolument sûr. Le certificat utilisé doit être signé par l'autorité de certification installée sur le TPE. stunnel (du moins la version ubuntu) s'attend à un format particulier, structuré de la manière suivante :

- clef privée RSA en base 64,
- ligne vide,
- certificat en base 64,
- ligne vide,

sans aucune donnée humainement lisible avant le certificat. Ce fichier ressemble donc à ceci :

```
-----BEGIN RSA PRIVATE KEY-----
[Donnees encodees en base 64]
-----END RSA PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
[Donnees encodees en base 64]
-----END CERTIFICATE-----
```

Il n'est pas souhaitable de lancer stunnel avec les droits root. Il faut cependant spécifier pour les logs et le fichier de PID des noms de fichiers dans lesquels stunnel a le droit d'écrire.

D.2 Paramètres en ligne de commande

Sous Ubuntu (notamment la version 8.04) les options sont passées en ligne de commande :

```
stunnel -f -D 7 -d 7345 -r afsollx:6504 -o log -p stunnel.pem -P ~/.
    stunnel/pidfile
```

- `f` (optionnel) oblige stunnel à s'exécuter au premier plan
- `D` (optionnel) permet de choisir le niveau de journalisation (7 au maximum, 5 par défaut)
- `o` définit le nom du fichier de logs
- `d [hôte:]port` spécifie l'adresse sur laquelle écouter le flux SSL (l'hôte par défaut étant localhost)
- `r [hôte:]port` spécifie l'adresse à laquelle renvoyer les données en clair
- `p` donne le chemin du certificat à utiliser.

En double authentification (authentification du client par le serveur), l'option `-v [niveau]` oblige stunnel à vérifier le certificat du client. Le niveau 2 effectue une vérification du certificat, le niveau 3 oblige à comparer avec le certificat racine correspondant, installé sur le système, et donc à s'assurer de la validité de la signature.

D.3 Paramétrage avec un fichier de configuration

Sous Red Hat, il semble que l'on soit obligé d'utiliser un fichier de configuration :

```
; Sample stunnel configuration file by Michal Trojnara 2002-2006
; Some options used here may not be adequate for your particular
    configuration
; Please make sure you understand them (especially the effect of
    chroot jail)
; Certificate/key is needed in server mode and optional in client
    mode
cert = /usr2/stagiaire/.stunnel/stunnel.pem
;key = /etc/stunnel/mail.key
; Some security enhancements for UNIX systems - comment them out on
    Win32
;chroot = /usr2/stagiaire/
;setuid = nobody
;setgid = nobody
; PID is created inside chroot jail
pid = /usr2/stagiaire/.stunnel/stunnel.pid
; Some performance tunings
socket = l:TCP_NODELAY=1
socket = r:TCP_NODELAY=1
;compression = rle
; Workaround for Eudora bug
;options = DONT_INSERT_EMPTY_FRAGMENTS
; Authentication stuff
;verify = 2
; Don't forget to c_rehash CApath
```

```
; CApath is located inside chroot jail
;CApath = /certs
; It's often easier to use CAfile
;CAfile = /etc/stunnel/certs.pem
;CAfile = /usr/share/ssl/certs/ca-bundle.crt
; Don't forget to c_rehash CRLpath
; CRLpath is located inside chroot jail
;CRLpath = /crls
; Alternatively you can use CRLfile
;CRLfile = /etc/stunnel/crls.pem
; Some debugging stuff useful for troubleshooting
;debug = 7
output = stunnel.log
; Use it for client mode
;client = yes
; Service-level configuration
[staps]
accept = 7345
connect = 6504
;[pop3s]
;accept = 995
;connect = 110
```

cert permet de donner le chemin du certificat

pid donne le nom du fichier dans lequel écrire le pid du processus

socket spécifie des options sur les interfaces TCP, ne pas modifier a priori

output définit le fichier dans lequel écrire les logs

Dans la section `[staps]`, on spécifie les ports à utiliser :

accept port sur lequel écouter le flux SSL

connect port auquel renvoyer les données en clair

D'autres options pourraient être intéressantes :

key suggère que cette version pourrait utiliser deux fichiers séparés pour le certificat et la clef privée (mais stunnel fonctionne aussi très bien avec le format indiqué ci-dessus) ;

debug change le niveau de journalisation ;

chroot, setuid et setgid permettent de faire fonctionner stunnel dans un environnement chrooté, ce qui est intéressant pour faire tourner stunnel en tant que démon ;

verify est utile en double authentification. Utiliser le niveau 2 pour vérifier le certificat du client, et le niveau 3 pour comparer avec un certificat racine installé sur le système.

Utiliser la commande

```
/usr/sbin/stunnel [fichier de configuration]
```

pour lancer stunnel sous l'utilisateur courant, ou bien déclarer stunnel en tant que démon.

Glossary

A | **B** | **C** | **D** | **E** | **G** | **I** | **L** | **M** | **N** | **O** | **P** | **R** | **S** | **T** | **U** | **V**

A

API Application Programming Interface, spécifications permettant de s'interfacer avec un composant logiciel, par exemple une bibliothèque logicielle. 34, 47

ASCII American Standard Code for Information Interchange, norme de codage de 128 caractères sur 7 bits. Il existe des versions étendues sur 8 bits pour le codage de certains caractères accentués ou spéciaux. 12, 38, 40

B

bien formé se dit d'un document respectant la syntaxe de XML ; en particulier, il possède un élément racine, les balises sont imbriquées correctement de manière à former un arbre et sont toutes fermées, les attributs sont adjoints d'une valeur entre guillemets doubles. 45, 48, 71

C

certificat X.509 support de la clef publique d'une entité. Il comporte également certaines informations comme le nom de cette entité, les dates de validité, les extensions en fixant le domaine d'utilisation. Sa signature par une autorité de certification permet d'authentifier ces informations et de rattacher de manière fiable une clef publique à son propriétaire. 24

compensation étape succédant à l'exploitation des transactions financières donnant lieu à l'équilibre des débits et crédits entre banques émettrices. 5, 38, 39

D

DTD Document Type Definition ou Définition de Type de Document. Syntaxe non XML permettant de décrire un modèle de document SGML ou XML, dont une instance peut être soumise à validation. 45, 51

E

entropie ce concept, établi par Claude Shannon dans le cadre de ses travaux sur la théorie de l'information, peut être compris ici comme une mesure de la quantité d'information, ou de l'incertitude sur ce qui est émis par une source d'information. Ainsi les clefs cryptographiques doivent-elles être les plus aléatoires possibles. 24

G

gabarit dans le cadre de ce TFE, un modèle issu d'une transformation XSLT d'un schéma XML, permettant la génération de remises au format XML. 47, 64, 66, 69, 70

GNOME GNU Network Object Model Environment, environnement de bureau libre. 44

GPL GNU General Public License, licence publique générale GNU, licence de distribution de logiciels libres, dont Richard Stallman et Eben Moglen furent les premiers rédacteurs, visant à garantir certains droits ou libertés considérés comme fondamentaux tels que les libertés d'exécution du logiciel, d'adaption des sources, de redistribution et d'amélioration. 33

I

IETF Internet Engineering Task Force, groupe ouvert en principe à tout individu, participant à l'élaboration de recommandations et de standards pour Internet, rédigés sous forme de RFC. 11, 28

inode contraction des termes anglais « node » (nœud) et « index ». Structure de données utilisées par certains systèmes de fichiers. À chaque fichier (ou répertoire) est attribué un numéro d'inode ; sous un système de type Unix, un inode renseigne certaines métadonnées comme les droits de lecture/écriture. 39

ISO International Organization for Standardization, Organisation internationale de normalisation, organisme de normalisation international ayant pour but la production de normes industrielles, économiques et commerciales. 7, 11

L

logiciel libre logiciel dont la licence donne à chacun le droit de l'utiliser, l'étudier, le modifier, le dupliquer et le diffuser. 37, 44, 46

M

makefile fichier utilisé par le moteur de production de fichiers `make`. Il décrit un ensemble de cibles, qui peuvent être des fichiers et calcule les dépendances entre les cibles de manière à les construire ou reconstruire récursivement, en exécutant les actions indiquées pour chacune d'elles. 46, 50

monétique terme non normalisé, mais couramment employé pour désigner le traitement informatique des transactions électroniques, notamment celles initées par carte de paiement. 2, 33

N

nom distingué assemblage de tous les composants, du plus précis au plus général, d'une entrée dans un annuaire LDAP. 27

O

obfuscation méthode visant à compliquer la compréhension du code d'un logiciel ou de certaines données. 24

P

passphrase mot de passe, idéalement long, servant de clef pour chiffrer de manière symétrique une information, qui peut être une clef privée. 34, 80

programmation orientée objet la programmation orientée objet cherche à modéliser directement des objets du monde réel. Un objet résulte de l'encapsulation de données et de traitements, protégés par une interface masquant l'implémentation. 44, 53

R

remise financière ensemble de transactions financières. 4, 17, 35, 38

RFC Requests For Comment, littéralement demande de commentaires, série numérotée de documents rédigés par l'IETF ayant trait aux aspects techniques d'Internet. 11, 25

S

serveur mandataire serveur ayant pour fonction de relayer des requêtes entre un client et un autre serveur (en anglais : « proxy »). 33

socket interface de programmation pour les communications entre processus. Une socket « Internet » fournit une interface pour l'utilisation du protocole TCP/IP. 15

SQL Structured Query Language ou langage structuré de requêtes, langage permettant d'interroger une base de données relationnelle. 5

SSL Secure Sockets Layer, couche cryptographique, fonctionnant au-dessus de TCP, assurant la confidentialité et l'intégrité des données échangées, la non-répudiation, et l'authentification. TLS est la version normalisée par l'IETF. 4, 18, 19

système de fichiers méthode de stockage et d'organisation de fichiers informatiques sur un support de stockage. 34, 39

T

timeout intervalle de temps après une action au-delà duquel l'émetteur considère que la connexion a été coupée s'il n'a pas reçu d'acquittement de la part de son correspondant. 15

TLS Transport Layer Security, version de SSL normalisée par l'IETF dans la RFC 2246 après rachat à la société Netscape du brevet états-unien en 2001. 4, 28

TP0 Transport Protocol Class 0, classe la plus simple du protocole de transport des couches OSI, conçue pour des couches sous-jacentes considérées comme sûres et ne nécessitant pas de mécanisme élaboré de contrôle d'intégrité et de gestion des erreurs. 12

TPE terminal de paiement électronique, dispositif d'acquisition de transactions financières initiées par carte de paiement. 4, 10, 33, 38

TPE IP TPE utilisant de manière native le protocole réseau TCP/IP, contrairement à l'ancienne génération généralement déployée sur des réseaux X.25. 14, 33

transaction financière opération donnant lieu à un crédit, un débit (ou une annulation) entre le porteur et l'accepteur. Les transactions sont transportées par l'acquéreur. 2, 7, 35, 38

U

UML Unified Modeling Language, langage de modélisation unifié. Langage graphique de modélisation objet. 52

V

valide se dit d'un document XML bien formé respectant les règles contenues dans un modèle tels qu'une DTD, un schéma du W3C ou Relax NG. 45, 51

Index

A

AC 24–28, 31–34, 36, 37
accélération SSL 34, 35
analyseur lexical 46, 50
analyseur syntaxique 46, 48, 49, 51
annotation 48, 49, 51, 53
ASCII 12, 38, 40
auto-signé 28
autorité d'enregistrement ... 31, 32, 37
autorité de validation 31, 32

B

biclef 21, 22, 24, 28, 36
Bison 46, 48, 50
brut 4, 38–40, 42–44, 48–50, 52–55

C

CB2A .. 3, 4, 17, 40, 42, 43, 50, 54, 56
CBPR 17, 40, 42, 50
Certificate 30
CertificateVerify 27, 30
chaîne de certificats 26, 28, 33, 36
ChangeCipherSpec 31
CHPN 40
CHPR 40, 42
clef cryptographique 5, 20, 21, 24
clef de session 28–31, 35
clef privée 21–28, 30, 31, 33–36
clef publique . 21–25, 27, 28, 30, 33, 36
ClientCertificate 30
ClientHello 29, 35
ClientKeyExchange 30
code d'identification de message 23, 29
compensation 5, 38, 39
concaténé 39, 40, 53, 54
corpus 49–53
CRL 27, 31, 32, 37
cryptographie asymétrique . 21–23, 28, 29, 33–35
cryptographie symétrique ... 20, 31, 33

D

demande de signature de certificat 25, 26, 28, 31, 32, 36
dictionnaire . . 39, 40, 46, 48, 50, 52–54
données d'appel 8, 16–18, 56
double authentification 30, 35–37

E

EJBCA 37
empreinte numérique ... 23, 24, 26, 27
EMULRFC1086 .... 14, 17–19, 56
Engine 35
enregistrement 13–17
entité d'enrôlement 31, 32
EPAS 1, 5, 38, 41, 42, 55, 56
espace des clefs 20
ETCD 7, 8
ETTD 7–9
exploitation 5, 38–40, 53, 54

F

Flex 46, 48, 50
fonction de chiffrement 20
fonction de déchiffrement 20
fonction de hachage 23, 27, 29, 30

G

gabarit 47, 49–53
GPL 33, 37

H

HSM 34, 35, 37
HTTP 27, 31

I

ICP 31, 36, 37
IETF 11, 28, 31
ISO 20022 .... 1, 3, 5, 39, 41

L

LDAP 27, 31
Lex 46

libxml2 . . . . 44, 45, 47, 49, 54
lot 39, 40, 53, 54

M

makefile 46, 50, 51
monétique 1–5, 7, 32, 33, 56

N

négociation SSL 28, 29, 31–35

O

OCSP 27, 31, 32
OpenCA 37
OpenSSL 20, 24, 33, 35, 54
OpenTrust PKI 37

P

PAD 8, 10, 13
passphrase 34, 36
PCI DSS 5, 34, 35, 53, 54, 56
PKCS#11 34
PKCS#12 . . . . 28, 36
PKIX 28, 31

R

remise . . 1, 4, 5, 17, 18, 35, 38, 39, 42, 43, 47, 50, 52–54, 56
RFC 1086 1, 4, 7, 9, 11–14, 16–18, 36, 56
RFC1086 . . . . 9, 14, 18, 56

S

schéma XML 42, 43, 45–51, 53
SEPA 3, 41
ServerHello 29
ServerHelloDone 30
signature 22–28, 31–33, 36, 54
simple authentification 32, 35
sous-adresse 13, 14, 16
SSL . 1, 4, 18–22, 25, 28, 29, 33–37, 56
STAP 1–5, 7, 11, 14–18, 33–35, 37, 42–44, 46, 53, 54, 56
structure de référence . . 42–44, 48, 49, 52–54
stunnel 33–37

T

télécollecte . 1–3, 14, 16–18, 34, 38, 40, 43, 46, 49, 56
TCP Keepalive 15
TCP/IP1, 2, 4, 7–19, 28, 33–35, 37, 38
TLS 4, 28
TP0 11–17
TPE 4, 5, 10, 14, 16, 17, 33–41, 43, 56
TPE IP 14, 17, 33
traitement 5, 39, 40, 53
transaction1–3, 5–7, 10, 18, 35, 38, 40, 42, 43, 52–54

U

UNIFI 41–43, 46, 50, 55

W

W3C 42, 44–46, 51, 54, 56

X

X.121 . . . . 8, 13, 14, 17
X.25 . . . . 1, 2, 4, 7–16, 18, 35, 56
Xerces 44
XML 1, 5, 6, 38–56
XoT 16
XPath 45, 47, 49, 54
xs:annotation 48
xs:appinfo 48, 49
xs:documentation 48
XSLT 44–48, 50, 51, 54

Y

Yacc 46