CONSERVATOIRE NATIONAL DES ARTS ET METIERS

PARIS

MÉMOIRE

présenté en vue d'obtenir

le DIPLÔME D'INGÉNIEUR CNAM

en INFORMATIQUE

par

Stephen NAVARRO

« SECURITÉ ET TEMPS DANS LES SYSTÈMES RÉPARTIS»

Soutenance le 25 septembre 2002

JURY **Président :** **M. Stéphane NATKIN**

Membres : **M. Gérard FLORIN**

M. Jean-Michel DOUIN

Mme Véronique DELEBARRE

M. Alexandre FERNANDEZ

M. Pierre-Alain FOUQUE

Stephen NAVARRO

Sécurité et temps dans les systèmes répartis

Mémoire d'ingénieur C.N.A.M. Paris 2002

Les systèmes répartis sont conçus pour assurer la mise en commun des ressources matérielles ou logicielles distribuées sur un réseau. Sur Internet, le partage des actifs et la coopération des entités peuvent poser de nombreux problèmes. Les opportunités offertes sont assorties de risques significatifs.

Ce mémoire reflète l'état d'avancement d'une recherche fondamentale à l'intersection de la sécurité et du temps dans les systèmes répartis. Notre ambition est de nous intéresser à la propriété de la sécurité qui traite de la non répudiation de l'information. Le temps est considéré du point de vue de la relation de causalité qui existe entre les événements du système réparti.

Ce mémoire repose plus particulièrement sur les besoins exprimés par la jurisprudence relative à la signature électronique avancée. En outre, les travaux sur la cohérence causale développés par une équipe du C.N.A.M. [CARREZ98] ont attiré plus particulièrement notre attention.

La problématique est posée pour définir un protocole sécurisé quant à la propriété de non répudiation des événements non seulement par origine, mais aussi par destination.

L'analyse de ces travaux nous a amené à proposer un prototype de programmation écrit en java qui satisfait dans une certaine mesure à la problématique énoncée et avancée.

Mots-clefs : systèmes répartis, sécurité, ordre causal, cryptographie, signature électronique, preuve, machine virtuelle java.

Keywords : distributed systems, security, causal order, cryptography, digital signature, proof, java virtual machine.

Remerciements

Je remercie Monsieur le Professeur Stéphane Natkin, pour qui j'ai une grande admiration, de l'honneur qu'il me fait en présidant ce jury. Directeur du Centre d'Etude et de Recherche en Informatique du Conservatoire National des Arts et Métiers (CEDRIC), il a trouvé le temps nécessaire pour encadrer ce travail dont il est à l'origine. Il a toujours su lever les ambiguïtés et relancer le débat dans une direction adéquate.

Je remercie Monsieur le Professeur Gérard Florin, alter ego du directeur du CEDRIC, d'avoir insufflé la force tranquille et la sérénité à cette recherche. Jamais encore je ne m'étais rendu-compte à quel point il est nécessaire de ne pas s'opposer à un concept exprimé et tout simplement génial.

Je remercie Monsieur Jean-Michel Douin, Maître de Conférence, pour l'enthousiasme à l'égard de la machine virtuelle java et des systèmes embarqués.

Je remercie Monsieur le Professeur Jean-François Dazy, pour ses patientes explications concernant les schémas de preuve, les tableaux sémantiques et les séquents de Gentzen.

Je remercie Monsieur Roger Tauleigne, Directeur au C.N.A.M. pour les éclaircissements prodigués dans le domaine de la complexité, voire de la limite infime entre le bon ordre et la confusion chaotique.

Je remercie Monsieur Patrick Dalion, enseignant au C.N.A.M. pour les cours de droit des affaires donnés en amphithéâtre. En prenant connaissance des lois, règlements et conventions, en écoutant les éléments de la jurisprudence qui se dégagent des décisions rendues par les tribunaux, le cours de ma vie a pris une nouvelle orientation.

Je remercie les membres du jury issus du secteur privé pour avoir dégagé un créneau horaire dans leur emploi du temps afin d'assister à la soutenance de ce mémoire d'ingénieur.

Sommaire

Table des illustrations

à Christine

Ils savaient qu'ils ne se prouveraient rien, qu'ils ne parviendraient pas à effacer le passé, et ils tentaient toujours cette besogne, ils revenaient toujours à la charge, aiguillonnés par la douleur et l'effroi, vaincus à l'avance par l'accablante vérité.

Emile Zola
(Thérèse Raquin 1867)

Introduction

Dans un monde évolutif, l'Homme décida de créer des machines pour s'en servir, pour lui faciliter les tâches répétitives et pour atteindre des objectifs de plus en plus éloignés. La volonté de progresser et de communiquer au niveau mondial engendra la notion de réseaux de machines interconnectées. Ainsi, est apparu un forum incontournable dans le monde des affaires, mais aussi dans le domaine de la santé ou de l'éducation. Internet, à l'origine n'était qu'un petit programme d'une entreprise d'informatique assurant la communication entre systèmes DEC et ordinateurs d'autres constructeurs [GINGUAY81]. Le souvenir s'est estompé et l'ingratitude désinvolte de la multitude a englué DEC dans l'oubli. Deux décennies se sont évanouies, les énergies canalisées et les capitaux investis ont permis de construire un magnifique outil de communication, des autoroutes de l'information et une somme de connaissances disponibles depuis n'importe quel endroit du monde. Internet est devenu le moyen de diffusion de l'information universel, le media privilégié.

D'un point de vue financier, des magiciens emblématiques ont su trouver les mots pour balayer les inquiétudes, rassurer les investisseurs et décider les sociétés de capital-risque. Certes la bulle spéculative a gonflé au point d'éclater, mais au vingt et unième siècle, après la première crise de la net-économie, l'Homme utilise encore le formidable outil de communication, envisage sans cesse de nouvelles fonctionnalités et souhaite ardemment conserver la pérennité de l'existant. Plus de la moitié des entreprises françaises sont d'ores et déjà reliées à Internet qui est devenu le support privilégié de la publicité et du commerce. L'éducation et la formation suivent cette tendance, nonobstant certaines réticences.

Certes, les investissements requis pour se connecter semblent modiques par rapport à d'autres solutions du marché. Certes, il existe enfin une norme de fait sur laquelle les industriels peuvent s'appuyer pour proposer leur produits à une multitude de clients potentiels. Mais il faut garder à l'esprit qu'Internet, à ses débuts correspondait à un idéal de confiance et de liberté, si bien que les opportunités offertes aux utilisateurs sont assorties de risques significatifs. Les transactions financières peuvent être falsifiées, des données privées être révélées publiquement et des actifs subir des dommages importants, voire irréversibles.

Bien évidemment, il ne viendrait à l'idée de personne de proposer des bijoux en or massif dans un distributeur de fête foraine dont la tirette dépendrait de l'introduction d'une simple pièce de monnaie. Et pourtant, les moyens d'accès à l'Internet paraissent d'un coût modique et les actifs disponibles en ligne semblent importants. Ainsi, la sécurité des réseaux demeure un défi majeur à l'ère des économies mondiales interdépendantes.

Objectif du travail

Ce mémoire reflète l'état d'avancement d'une recherche à l'intersection de la sécurité et du temps dans les systèmes répartis. Notre ambition est de nous intéresser à la propriété de la sécurité qui traite de la non répudiation de l'information. Le temps est considéré du point de vue de la relation de causalité qui existe entre les événements du système réparti.

L'objectif du travail est de prémunir les systèmes informatiques de certaines attaques organisées par des opposants potentiels.

Présentation de la démarche utilisée

Le chapitre 1 regroupe les définitions informatiques et juridiques nécessaires pour exprimer l'orientation choisie. Il situe les limites imposées qui réduisent le domaine relatif à la sécurité et au temps dans les systèmes répartis.

Le chapitre 2 regroupe les spécifications du protocole que nous appelleront SETSAR, qui est l'acronyme de Sécurité Et Temps dans les Systèmes et Applications Répartis. Dès que les objectifs de ce protocole sont établis, la politique de sécurité est déterminée. Lors de la phase de conception préliminaire, les spécifications fonctionnelles précèdent l'analyse du protocole, ce qui permet de caractériser les exigences et l'architecture. La preuve statique des algorithmes principaux est établie en utilisant le formalisme des séquents de Gentzen. D'autre part, une preuve dynamique considère l'ensemble des solutions possibles d'un algorithme de l'application ; la satisfaction de certaines branches de l'arbre de preuve est envisagée. Ensuite, la conception générale et détaillée nous fournira une solution d'implantation de classes d'objets écrite en java.

Le chapitre 3 est organisé dans le but de discuter de la portée de la solution choisie et d'ouvrir le débat sur de futures évolutions et recherches. Le codage est présenté en annexe.

Pour rendre moins aride la lecture de ce document, il semble judicieux de présenter les identités fictives des acteurs qui interviennent habituellement dans les écrits relatifs à la sécurité. Alice est l'émetteur, Bob est son interlocuteur privilégié. Dans les protocoles à 3 ou 4, apparaissent les lettres C et D, Christine et David. La lettre O désigne l'opposant, citons Oscar et Onrasegratis [NATKIN02]. Estelle dont le prénom commence par un E est l'espionne qui espère casser les protocoles et ainsi devenir riche, célèbre et donc, au moins en apparences, heureuse en amour. Maurice est le plus méchant et le plus anonyme de tous. Estelle et Maurice se livrent à la crypte-analyse qui est l'art de casser les crypto-systèmes.

1 Terminologie et limites imposées

1.1 Définitions

1.1.1 Définition de la sécurité

La sécurité est un ensemble de méthodes et de moyens mis en oeuvre pour se protéger contre les défaillances résultant d'une action intentionnelle visant au vol ou au sabotage portant sur les systèmes d'information [NATKIN02].
La bonne marche d'un système informatique couvre deux domaines différents qui sont d'une part la sécurité proprement dite et d'autre part la sûreté de fonctionnement. La langue anglaise distingue "security" pour la sécurité et "dependability" pour la sûreté de fonctionnement. Le premier domaine, la sécurité, conjecture l'existence d'une intention délibérée de nuire au système, parfois de simples suppositions n'ont reçu aucune confirmation ! Le deuxième domaine, la sûreté de fonctionnement, qui suppose des alea plus naturels lorsqu'une erreur interne au système survient et provoque une faute puis une défaillance. Ce domaine relatif à la sûreté de fonctionnement regroupe les propriétés qui assurent la pérennité du système. La tolérance aux fautes, la disponibilité, la fiabilité et la maintenance sont des propriétés essentielles de la sûreté de fonctionnement.

1.1.2 Politique de sécurité

La sécurité ne peut être délimitée, mise en œuvre et fixée que relativement à des objectifs clairement définis qui sont le périmètre d'action, les droits, les attaquants et les défaillances potentielles. Ces quatre éléments sont constitutifs de la politique de sécurité.

1.1.2.1 Un périmètre d'action

Le périmètre d'action est délimité en répondant tout simplement aux questions fondamentales de toute mise en situation. Il s'agit de déterminer qui est concerné par la communication, en quel lieu se situe l'échange, la date à laquelle il peut se produire, comment se déroulent les actions, et ainsi de suite.

1.1.2.2 Les droits

Les droits sur les objets de la politique sont octroyés ou retirés aux entités du système d'information. Les droits sur les hommes ou les biens matériels ou immatériels résultent de règles visant à définir les actions autorisées ou interdites.

1.1.2.3 Les attaquants

La nature et la force des attaquants éventuels doivent être envisagées lors de la définition de la politique de sécurité. Estelle est à l'affût, et il ne faut pas sous-estimer Onrasegratis.

1.1.2.4 Les défaillances potentielles

La nature des défaillances doit être envisagée pour que le système soit en mesure de résister à des attaques menées à l'encontre de son dessein et de ses actifs.

1.1.2.5 Types de politique de sécurité

La politique de sécurité est indépendante des moyens utilisés pour l'assurer parce que la politique de sécurité résulte de la spécification de besoins exprimés. Les exigences sont d'abord énoncées en fonction de l'existence et de l'évolution de circuits d'information. Ensuite, les exigences complémentaires sont répertoriées selon une analyse des risques liés au

dysfonctionnement de ces circuits, de l'intérêt potentiel des attaquants et de contraintes juridiques. Il convient de différencier la politique de rôle et la politique nominative.

1.1.2.5.1 Politique de rôle, politique nominative

Dans le cas d'une politique de rôles, tous les droits d'une politique sont attribués aux personnes uniquement selon leur fonction au sein du système d'information (direction générale, responsable de la communication, administrateur système, administrateur réseaux, responsable de sécurité, le reste du monde…). Une telle politique doit préciser les procédures appliquées pour attribuer un rôle à une personne. À l'inverse, dans le cas d'une politique nominative, un droit est attribué à une personne au moins (intuitae personae). Chaque nouvelle déclaration d'entité nécessite l'attribution du droit. De plus, si un nouveau droit apparaît, toutes les entités du système doivent être mises à jour une par une. Par conséquent, la deuxième approche apparaît plus difficile à superviser à long terme que la première approche qui nécessite une exploitation minimale.
Il convient aussi de différencier la politique obligatoire de la politique discrétionnaire.

1.1.2.5.2 Politique obligatoire, politique discrétionnaire

Parmi tous les droits il existe une classe particulière qui concerne le droit de donner à un tiers un droit que l'on possède. Par exemple on peut autoriser un tiers à lire un document que l'on est soi-même autorisé à lire. Un tel droit est dit droit de propagation. Dans de nombreux systèmes le créateur d'un objet possède tous les droits sur cet objet y compris le droit de propagation de tous les autres droits sur cet objet. L'usage inconsidéré des droits de propagation est une des failles essentielles de certaines politiques de sécurité. Il est en effet difficile de contrôler la propagation de ce droit par le tiers autorisé. Révoquer ce droit peut s'avérer encore plus difficile.
Une politique de sécurité telle que les tous les droits de propagation sont régis par des règles indépendantes du créateur (ou du possesseur) d'un objet est dite politique obligatoire. De plus, une politique de sécurité qui n'est pas obligatoire est dite politique discrétionnaire.
Quand le propriétaire d'un objet est capable de propager les droits qu'il possède sur cet objet, alors la politique de sécurité est de type discrétionnaire, dans ce cas, la propagation des droits est laissée à la discrétion du propriétaire de ces droits.

1.1.3 Propriétés de sécurité

Des attaques existeront toujours. Elles sont dirigées contre des objectifs précis. Certaines attaques visent les propriétés d'authentification, de confidentialité ou d'intégrité. D'autres opposants visent la propriété de non-répudiation, voire les procédures d'archivage. Détaillons chacune de ces propriétés.

1.1.3.1 L'authentification

L'authentification est liée à la propriété d'identification. L'identification d'une entité est la propriété qui permet à une entité d'entrer en relation de communication avec une autre entité, qu'il s'agisse d'une personne physique, d'une machine ou d'un service. Ensuite, une procédure d'authentification permet de prouver que cette entité est bien celle que cette entité prétend être. Lors de la mise en relation de plusieurs entités, les procédures d'authentification des entités respectives sont dépendantes du temps et de l'espace de communication.
Dans le cas de l'authentification d'un message, la problématique est légèrement différente et nous renvoie à la notion de signature électronique que nous définirons au paragraphe [1.2.2].

Dès que l'écueil de l'identification est contourné en exploitant une faille du système, l'opposant peut s'attaquer à la confidentialité et à l'intégrité des données. Cependant, le viol de confidentialité ou d'intégrité ne procède pas seulement d'une infraction lors la phase d'authentification.

1.1.3.2 La confidentialité

La confidentialité est la propriété qui garantit que les informations ne sont lisibles que par les entités habilitées selon les règles d'une politique de sécurité. Si le message a été chiffré, il ne peut être déchiffré que par les personnes habilitées. La lecture des fichiers par une personne non autorisée compromet la confidentialité. Le préjudice commercial, politique ou privatif est proportionnel au degré de confidentialité des documents lus par le pirate de l'information. Mis à part le domaine de la vie privée, il existe plusieurs niveaux de confidentialité. Citons le secret commercial, le secret défense et le secret d'état.

1.1.3.3 L'intégrité

L'intégrité est la propriété de la sécurité qui garantit que les informations ne sont modifiables que par les entités habilitées, selon les règles d'une politique de sécurité. Dans le cas où un opposant au système parviendrait à modifier, voire à supprimer les informations, la propriété de l'intégrité serait bafouée.

Mais l'authentification, la confidentialité et l'intégrité ne sont pas les seules propriétés de la sécurité. La non-répudiation est une propriété moins connue, tant il est vrai qu'il semble a priori facile de confondre l'intégrité et la non-répudiation.

1.1.3.4 La non-répudiation

La non-répudiation est la propriété qui assure que l'auteur d'un acte ne peut ensuite dénier l'avoir effectué. Un client a bien passé électroniquement une commande de dix millions de scoubidous à 89 centimes pièce alors qu'il prétend que le prix était de 69 centimes seulement [TANENBAUM98]. Dans le domaine des transactions électroniques il est courant de distinguer la preuve d'origine qui implique que l'émetteur d'un ordre ou d'un message ne peut nier cette émission et la preuve de réception qui implique que le récepteur d'un ordre ou d'un message ne puisse nier à tort avoir reçu cet ordre ou ce message. Dans notre exemple, puisque l'émetteur nie avoir passé un ordre d'achat à 89 centimes, il faut apporter une preuve d'origine de cet ordre d'achat pour mettre en évidence la **non-répudiation par origine**.

Prenons un exemple pour illustrer la propriété de **non-répudiation par destination** dans le domaine de la sécurité informatique. Bob envoie une offre de prix à Maurice et Maurice s'engage à acheter 100 exemplaires d'un livre au prix unitaire de 10 Euros. Bob acquitte le bon de commande et envoie à Maurice les produits demandés. Celui-ci commercialise les ouvrages mais ne donne que 900 Euros à Bob en exhibant une offre de prix sur laquelle est inscrit un prix unitaire fixé à 9 Euros. Maurice répudie le prix initial et oppose à son créancier un autre document. La mauvaise foi de Maurice est telle qu'il affirme détenir le document intègre de toute modification. Un sérieux problème de confiance réciproque est apparu. Pour Bob, les faits sont suffisamment graves pour le motiver à porter l'affaire devant la juridiction compétente. Le juge va instruire l'affaire à charge et à décharge. Certaines preuves apportées à charge dans le dossier peuvent s'avérer innocenter totalement le présumé innocent et

réciproquement. C'est la raison pour laquelle, il vaut mieux que le juge soit unique de manière à instruire à la fois à charge et à décharge. Il va rechercher une preuve nécessaire et suffisante pour emporter son intime conviction. Pour cela il va s'appuyer sur l'auditabilité du système informatique.

1.1.3.5 L'auditabilité

L'auditabilité est la capacité à détecter et à enregistrer de façon infalsifiable les tentatives de violation de la politique de sécurité. Ainsi une personne externe au système est en mesure de contrôler l'exécution des objectifs et de faire la part des choses dans le but de prendre une décision. Dans l'exemple de Bob et de Maurice, l'auditabilité permet de déterminer avec exactitude lequel des documents est intègre, celui de Bob ou celui de Maurice. Ainsi le juge s'appuie sur l'auditabilité pour rechercher la vérité dans le système.

Les quatre propriétés principales de la sécurité sont des objectifs précis et des buts à atteindre pour les pirates de l'information. Le paragraphe suivant détaille les moyens de préserver ces propriétés pour que les objectifs soient sinon impossibles à atteindre, pour le moins difficiles à exploiter. La sécurité doit opposer à l'attaquant des obstacles infranchissables.

1.1.4 Moyens de préserver les propriétés de la sécurité

Le responsable de la politique de sécurité doit s'appuyer sur des moyens afin de préserver les propriétés fondamentales de la sécurité. [MENEZES96]

1.1.4.1 La propriété, le contrôle d'accès

Une entité a le droit de lire ou d'écrire une information à un instant précis parce que la notion primordiale de propriété existe et perdure. La propriété est un moyen de fournir à une entité les droits légaux d'utiliser ou de fructifier une chose, voire d'en abuser, c'est à dire de transférer la propriété à une autre entité. Ces droits légaux doivent être vérifiés, surveillés et contrôlés. Le contrôle d'accès permet de restreindre l'accès à des ressources au profit d'entités privilégiées en respectant les droits tels qu'ils ont été définis lors de la spécification de la politique de sécurité.

1.1.4.2 L'anonymat, la protection

L'identification des acteurs du système est la plus triviale des propriétés de la sécurité. Par conséquent il s'agit d'augmenter les obstacles afin d'empêcher l'attaquant d'être en mesure d'identifier ces acteurs. A la limite, si l'identification s'avère impossible, c'est l'anonymat qui est préservé. L'anonymat cache l'identité d'une entité impliquée dans une procédure. Puisque l'identification est rendue difficile, voire impossible, l'information est tenue secrète pour tout le monde excepté de ceux qui ont l'autorisation d'en prendre connaissance. La protection de la vie privée est alors assurée. La protection des données permet de se prémunir contre les attaques menées à l'encontre de la propriété d'authentification.

1.1.4.3 Le reçu et la confirmation

Dans les systèmes répartis asynchrones, le mode de communication privilégié voire unique est fondé sur l'envoi et la réception de message, des explications sont disponibles au paragraphe [1.3.4]. Il est tout à fait légitime de procéder à des acquittements de bon aloi à intervalles réguliers. Le reçu est tout simplement un acquittement qui prouve que l'information a été reçue. Lorsque l'acquittement concerne un service qui a été fourni, il s'agit d'une confirmation.

1.1.4.4 La validation, la certification

La validation est un moyen de fournir l'opportunité d'autoriser l'utilisation ou la manipulation d'information ou de ressources. La validation présuppose la preuve de la justesse d'agir ainsi. Dans le cas de simulation entre les parties du contrat, un acte secret est valide entre les parties. Alors c'est l'acte secret et non l'acte apparent qui régit les relations entre les parties.
La certification permet l'approbation d'une information par une entité de confiance.

1.1.4.5 L'authentification, la signature

L'authentification permet de corroborer la source de l'information.
La signature est un moyen de relier l'information à une entité. La signature doit émaner de celui à qui elle est attribuée, le mandataire ne peut donc pas signer du nom du mandant, même à la demande de celui-ci. Depuis le 27 janvier 1993, il ne sert plus à rien d'ajouter la mention « lu et approuvé ». Cette formule est dépourvue de toute portée [MERCADAL93].
La signature a la même force probante que la signature manuscrite sous seing privé, c'est la raison pour laquelle, à l'article 1326 du code civil, les mots : « de sa main » sont remplacés par les mots : « par lui-même ». [FAUSSE01].
Cette précision juridique de la portée de la signature électronique, nous amène tout naturellement à considérer la sécurité des systèmes informatiques du point de vue du juriste. Ainsi, il apparaît primordial de rappeler quelques vérités en rapport avec la législation tant française qu'étrangère en matière de sécurité informatique.

1.2 Le droit

Les lois et règlements sont décidés et énoncés par des personnes responsables dont les opinions sont différentes selon les pays et les époques. La jurisprudence est le résultat d'une évolution politique et sociale en constante mutation. Ainsi, la perception du droit diffère selon les pays.

1.2.1 La loi dite Godfrain

En France, le député Godfrain élu de l'Ariège fut l'instigateur d'une loi qui réprime le piratage informatique. Cette loi traite des atteintes aux systèmes de traitement automatisé des données. Cette loi a été instituée le 22 juillet 1992, et son décret d'application a été promulgué le 1er mars 1994. Les articles 323-1 à 323-7 du Code pénal prévoient des peines de prison, des interdictions, des confiscations et de lourdes amendes à l'encontre des personnes physiques ou morales dont les agissements ont causé des préjudices à des propriétaires de systèmes d'information. Une ordonnance du 22 septembre 2000 entrée en vigueur le 1er janvier 2002 montre que le législateur ne souhaite pas laisser cette loi pénale tomber en désuétude. Bien au contraire, il maintient les textes en conformité avec le passage à la monnaie unique de plusieurs pays de l'union européenne.
Le législateur a prévu trois catégories de délit de piratage en s'appuyant sur une typologie de la fraude schématiquement résumée par trois notions de bases : l'authentification, l'entrave au fonctionnement et l'intégrité des données. Détaillons chacun des trois cas prévus dans le cadre de cette loi pénale. Mais rappelons aussi que nonobstant la loi sur la présomption d'innocence, la bonne foi est avant tout utilisée par le législateur et les tribunaux pour tenir compte de la loyauté des intéressés (et désintéressée). La bonne foi se présume, c'est à dire que l'on n'a pas à la prouver, mais comme il s'agit d'une présomption simple, il est possible d'en apporter la preuve contraire.

1.2.1.1 L’authentification

L'accès et le maintien frauduleux total ou partiel dans tout ou partie d'un système ou délit d'intrusion est puni par l'article 323-1 d'un an d'emprisonnement et de 15 000 euros d'amende. Ainsi, non seulement l’accès frauduleux à un système protégé est un délit, mais de plus, l’utilisation du code exact par une personne qui n’a pas le droit de l’utiliser constitue aussi un délit. Enfin dans un dernier cas, celui du maintien de connexion, le législateur s’est intéressé au temps passé par la machine à effectuer des calculs sans facturation possible, là encore, le délit est caractérisé.

1.2.1.2 L’entrave au fonctionnement

L'atteinte volontaire au fonctionnement d'un système de traitement informatisé de données ou délit d'entrave est puni par l'article 323-2 de 3 ans d'emprisonnement et de 45 000 euros d'amende. Cet article de la loi dite Godfrain, vise tous les procédés utilisés pour bloquer ou brider volontairement un système en agissant sur les éléments matériels, les logiciels ou les organes de sortie, voire les imprimantes. L’emploi de logiciels permettant d'autres opérations relatives au fonctionnement, des destructions ou l’introduction de virus informatiques est englobé dans la formulation de l’entrave au fonctionnement.

1.2.1.3 L’intégrité des données

L'atteinte volontaire aux données ou délit de piratage est l'introduction frauduleuse de données dans un système, la suppression ou l'altération frauduleuse des données qu'il contient. Ce délit est puni par l'article 323-3 de 3 ans d'emprisonnement et de 45 000 euros d'amende. Cette disposition complète la précédente et protège les données elles-mêmes contre l'effacement, la modification ou le maquillage.

Ainsi, l’intrusion dans un système d’information est considérée comme un délit passible de sanctions financières et de contrainte de corps. Les modalités sont répertoriées dans le code pénal. L’état est partie prenante dans le processus et peut engager des poursuites à l’encontre des personnes physiques ou des personnes morales responsables des contrevenants. La prise de conscience peut s’avérer brutale, pour les responsables mais aussi pour certains jeunes informaticiens par ailleurs honorablement connus. L’attitude parfois insouciante, parfois insolente des pirates, pas toujours jeunes d’ailleurs, semble comprise par les juges d’instruction qui instruisent les dossiers à charge et à décharge. Mais une fois que la justice est en marche, les choses vont à leur terme. Par exemple, tombé pour l’exemple, un pirate peut passer 5 ans de sa vie dans un service informatique, puis pendant 2 ans il purge une peine de prison. La chance insolente a fini par tourner. Il apprend alors qu’il vaut mieux éviter la promenade de la cour de la prison à cause des prisonniers de droit commun. De retour sur le marché du travail, ses connaissances informatiques ne sont plus au goût du jour. De plus, les employeurs n’engagent pas volontiers des ex-prisonniers. Ainsi par excès d’optimisme ou de bravoure mal canalisée, le persécuteur est devenu une victime.

Par conséquent le législateur dispose d’une jurisprudence et de lois répressives pour protéger l’état et donc les citoyens, mais il dispose aussi de lois et de décrets dont l’objet est de faciliter et de promouvoir le développement des échanges commerciaux par le biais de l’électronique.

1.2.2 La signature électronique

1.2.2.1 La directive européenne

Le législateur annonce le terme de la démarche juridique qui prendra place dans la loi sur la Société De l'Information (SDI) [FAUSSE01].
La directive européenne de décembre 1999 vise à ce que les nouvelles dispositions répondent aux exigences légales d'une signature à l'égard de données électroniques de la même manière qu'une signature manuscrite répond à ces exigences à l'égard de données manuscrites ou imprimées sur papier. Elle comporte 15 articles et 4 annexes. Le champ d'application énonce les limites imposées et la terminologie. Les grands principes de liberté de circulation des biens et des services sont repris dans l'article de l'accès au marché. Sans détailler la loi, intéressons-nous à l'article 5 qui différencie d'une part la signature électronique simple et d'autre part la signature électronique avancée. L'article 5 précise en outre que la signature électronique simple ne peut pas faire l'objet d'une répudiation juridique globale [PIETTE01]. L'article 6 fixe les limites de la responsabilité du prestataire de service de certification. De plus, cet article énonce les dispositions concernant la valeur limite des transactions et la condition de révocation des certificats.

1.2.2.2 La jurisprudence française

En France, la démarche juridique n'est pas en retard par rapport à la directive européenne. Le code civil a été modifié. La loi du 13 mars 2000 est récapitulée dans les articles 1316-4 et suivants du code civil. A l'article 1326 du code civil, les mots : « de sa main » sont remplacés par les mots : « par lui-même ». L'article 1316-3 est ainsi rédigé « L'écrit sur support électronique a la même force probante que l'écrit sur support papier ».
Le décret d'application de l'article 1316-4 du code civil fut énoncé le 30 mars 2001, suite à réunion du conseil d'état restreint composé du Premier ministre, de la garde des sceaux, du ministre de l'intérieur, du ministre de l'économie des finances et de l'industrie, du secrétaire d'état à l'outre mer et du secrétaire d'état à l'industrie[PIETTE01]. Le décret parut en ces termes au journal officiel. Ce décret précise les conditions qui permettent à la signature électronique d'être considérée comme un procédé d'identification présumé fiable. La présomption de fiabilité repose par dessus tout sur la mise en œuvre d'une signature électronique sécurisée. Le chapitre I traite des dispositifs de création de la signature électronique. Le chapitre II traite des dispositifs de vérification de signature électronique. Le chapitre III traite des certificats électroniques qualifiés et des prestataires de service de certification électronique. Le chapitre IV traite des dispositions diverses.
Ainsi, le décret d'application de la loi précise les conditions de fiabilité du procédé de signature électronique avancée. Mais de nombreux points demeurent obscurs. Par exemple, la jurisprudence, étoffée par l'arrêt du 20 octobre 2000, ne parvient pas à déterminer de manière précise l'identification de la personne physique qui a apposé la signature électronique sur un document lorsque plusieurs personnes différentes, travaillant ensemble, ont accès aux clefs de chiffrement et au logiciel de cryptographie.
En ce qui me concerne, j'estime que toute latitude est laissée au juge pour explorer l'aspect pénal de l'affaire et permettre aux avocats et aux greffiers de prospérer. De plus, j'estime que le texte de loi comporte des définitions incomplètes et des promesses d'éclaircissement non suivies d'effet lors de la parution du décret d'application de la loi. Celui-ci renvoyant le lecteur à la lecture de la loi pour des points et des définitions qui restent ainsi à la libre interprétation des partis en présence.

1.2.3 Législation mondiale des clefs de chiffrement

Les clefs de chiffrement sont soumises à des contraintes qui varient selon les pays.

En France, le pouvoir législatif permet aux pouvoirs judiciaires et exécutifs de surveiller, sur commission rogatoire, les agissements de certaines personnes ou associations de personnes. Selon la loi du 19 mars 1999 les produits informatiques qui contiennent des clefs de chiffrement sont soumis à contingentement. Avant 1999, toute clef de chiffrement était soumise à l'approbation des autorités compétentes. La réglementation en vigueur a été considérablement assouplie. Mais elle a gagné en complexité. Des nuances existent selon la longueur des clefs utilisées. On distingue les trois cas suivants : moins de 40 bits, de 40 à 128 et plus de 128. Cette typologie a été décidée en conseil des ministres. De 40 à 128 l'utilisation est libre mais la commercialisation et l'importation doivent être déclarée. Au-delà de 128 bits, une déclaration doit être faite auprès de l'autorité compétente qui est le Service Central de Sécurité des Services Informatiques (SCSSI). Cet organisme délivre les autorisations de distribution, d'utilisation, d'importation et d'exportation.

En Chine de Pékin, l'intolérance à l'égard de la taille des clefs de chiffrement est totale. La longueur des clefs permise est égale à zéro, parce que le chiffrement est interdit ! Sans porter de jugement sur la réglementation en vigueur actuellement en Chine, il est toutefois indéniable que cette législation est de nature à constituer un obstacle à l'émergence des nouvelles techniques de traitement de l'information.

Aux Etats Unis d'Amérique, si la longueur d'une clef dépasse 56 bits, son utilisation est limitée au territoire fédéral. Son exportation est prohibée. Le délit constitué est assimilable à l'exportation de matériel militaire et sévèrement réprimé.
A titre d'exemple, considérons un protocole Internet appelé PGP pour Pretty Good Privacy, ce qui est traduisible par « bien-être chez soi ». Ce protocole n'est pas dans le domaine public parce qu'il est soumis à la législation. En effet, l'auteur de ce protocole a déposé un copyright, et de plus, PGP est disponible suivant les termes de la « Copyleft General Public Licencing » énoncée par la Fondation pour le Logiciel Libre. Le Copyleft représente une alternative. Les programmes sont distribués directement dans le domaine public, sans copyright. Si un entrepreneur distribue un de ces programmes avec ou sans modification, alors il doit aussi transmettre la liberté de transmettre ou modifier le programme.
Le jeu de mot qui existe entre copyleft d'une part et copyright d'autre part, est symptomatique d'un certain clivage qui provoque certes de l'émulation mais aussi parfois de sérieux ennuis. Le protocole PGP nécessite des clefs de chiffrement dont la longueur est paramétrable et peut dépasser le maximum autorisé à l'exportation. Les instances fédérales américaines se sont appuyées sur ce fait pour faire passer en jugement un de ses inventeurs. Phil Zimmermann a toujours souhaité rendre publics les codes de ses programmes. Cette attitude lui a valu de passer un mois de sa vie dans un établissement pénitentiaire.

1.2.4 Conclusion

En conclusion de ce paragraphe dédié au droit aussi bien national qu'international, certes les lois sont différentes selon les pays. Mais, invariablement, l'état est partie prenante dans les processus juridiques et peut engager des poursuites à l'encontre des personnes physiques qui enfreignent les dispositions qui tendent à protéger les systèmes d'information. Cependant, le législateur adapte les codes et la jurisprudence pour légitimer la force probante de la signature électronique et avaliser les grands principes de liberté de circulation des biens et des services.

1.3 Les systèmes répartis

1.3.1 Introduction à l'architecture des systèmes répartis

Des machines distantes reliées en réseaux peuvent former un système informatique. Ainsi, les différents systèmes autonomes distants sont en mesure d'échanger des messages en fonction de choix et de buts qu'ils voudraient atteindre. Ils forment alors un système réparti. Du point de vue du matériel, plusieurs machines sont reliées entre elles. Mais du point de vue de l'utilisateur, il n'existe qu'une seule entité. Le système est unique et réparti sur l'ensemble des machines. Nous emploierons indifféremment les termes répartis et distribués pour qualifier les systèmes composés de plusieurs systèmes connectés entre eux.

1.3.2 Tolérance aux pannes

Le domaine de la tolérance aux pannes n'est pas directement lié au sujet de ce mémoire, cependant, la thématique de la tolérance aux pannes est indispensable pour définir plus précisément la notion de système réparti. Dans une première approche, il semble qu'une machine quelconque soit nécessaire mais non suffisante au fonctionnement du système. Par conséquent l'arrêt d'une machine peut provoquer l'arrêt du système dans son ensemble si bien que parfois, un utilisateur du système est pénalisé par la défaillance d'une machine dont il ignorait jusqu'à l'existence. Cette notion de maillon faible est connue en terminologie britannique sous la dénomination de « single point of failure » acronyme de SPOF.
Mais, fort heureusement, il existe des parades à ce genre de situation. En effet, les systèmes comportent des dispositifs fondés sur la redondance spatiale, temporelle ou hybride. En cas de redondance spatiale il existe plusieurs voies ou chemins possibles pour transférer l'information, ainsi la probabilité que l'information soit transférée est plus élevée même si un dispositif est défectueux. Toutefois cette approche spatiale de la redondance peut procurer l'apparition de boucles. En cas de redondance temporelle la même information est envoyée plusieurs fois sur le même chemin si bien que la probabilité que le contenu du message arrive à destination est assez grande, toutefois l'échéance de délivrance étant parfois dépassée, il est possible que le message ne parvienne pas à temps. La redondance hybride mélange les deux approches de la redondance, spatiale et temporelle, pour obtenir une redondance maximale.

1.3.3 Architecture

Les machines distantes sont autant de systèmes autonomes. Chaque machine grâce bien entendu à l'utilisateur qui la manipule, prend son autonomie non seulement par rapport au système réparti mais aussi par rapport au milieu extérieur. A partir de ce milieu, le système autonome acquiert la possibilité d'entrer avec lui en relation aléatoire. Ainsi, il est tout à fait envisageable de transférer des logiciels depuis le monde extérieur jusqu'au micro-noyau du système réparti de manière aléatoire et imprévisible.
La conception et la mise en œuvre de systèmes répartis correspondent à un besoin clairement identifié. La mise en commun des ressources apporte un gain énorme. Des algorithmes répartis garantissent la cohérence des données gérées par plusieurs processus répartis sur l'ensemble des systèmes autonomes qui constituent le système réparti. Cependant, toute médaille a son revers. Le partage des données entre utilisateurs peut devenir un inconvénient. En effet, si un utilisateur peut accéder facilement aux données, il peut lire, voire modifier des données qui ne lui sont pas forcément destinées. Si d'une part il existe des moyens permettant à n'importe quel utilisateur d'accéder à n'importe quelle donnée et si d'autre part certaines données doivent rester absolument secrètes, alors il est évident qu'un sérieux problème existe et qu'il vaut mieux reconsidérer l'architecture du système. La migration de l'application vers un ordinateur isolé dans une pièce fermée et non reliée au réseau doit être effectuée dans les plus brefs délais. Mais nous ne traiterons pas de ce type d'architecture, ni d'ailleurs de ce type de sécurité. Nous explorons les possibilités liées aux systèmes répartis.

1.3.4 La communication synchrone ou asynchrone

Le dictionnaire « LEXIS » donne la définition suivante de l'adjectif asynchrone employé dans le contexte de l'informatique : « Se dit d'un ordinateur dans lequel chaque opération est initialisée par un signal provoqué soit par l'achèvement de l'opération précédente, soit par la libération des organes nécessaires. » Mais cette définition ne suffit pas à différencier les communications synchrones des communications asynchrones, puisque cette définition s'applique aussi dans le cas d'un système synchrone. Toutefois, le système synchrone est rythmé par une horloge unique alors que le système asynchrone ne privilégie pas l'horloge pour cadencer l'information. Les notions de synchrone et d'asynchrone sont très délicates à manier. On peut voir deux niveaux de définition qui se rejoignent selon que l'on utilise la notion d'horloge ou la notion de borne pour caractériser les données temporelles.

1.3.4.1 Notion d'asynchrone et de synchrone par référence à une horloge

Un système est synchrone si les événements qu'il traite sont rythmés selon une horloge.
Un système asynchrone traite les événements aléatoires selon leur instant de présentation.

1.3.4.2 Notion d'asynchrone et de synchrone par référence à des bornes temporelles

Un système est synchrone si toutes les grandeurs temporelles le caractérisant peuvent être bornées. Les délais de transmission des messages sont contingentés, la vitesse des processus est bridée, la dérive des horloges est supervisée.

Un système asynchrone traite les événements alors qu'il s'avère impossible de fixer des bornes sur certaines caractéristiques temporelles. Par exemple, une boîte aux lettres est ouverte à un instant qui reste à la discrétion totale de son propriétaire.

Dans le domaine des systèmes répartis, l'envoi et la réception des messages sont liés de manière inséparable au mode asynchrone qui est le mode naturel de communication entre ressources distantes. Cependant, c'est le point de vue métier qui perçoit le système comme un ensemble de ressources distantes. Par ailleurs, les échanges de messages sont réalisés au moyen de signaux et des deux (2) primitives principales « envoyer » et « recevoir ».

Différentes techniques de communication existent et correspondent à des besoins différents qui varient selon les protocoles.

Définition : un protocole est un ensemble de règles et d'étapes successives suivies par chaque entité reliée au canal de communication.
Classons les différentes techniques de communication par type et détaillons les successivement.

1.3.4.3 Typologie de communication

1.3.4.3.1 Diffusion sélective (multicast)

Dans le cas de la communication par diffusion sélective de type multicast, le système est composé d'un ensemble de processus qui sont séparés dans l'espace et qui communiquent au moyen d'un réseau. Ainsi un processus fournit explicitement la liste des destinataires et envoie le message simultanément à tous les processus concernés et à aucun autre.
Dans l'illustration suivante le processus A envoie le message simultanément au processus B et au processus D. Il n'envoie pas le message au processus C. [TANENBAUM95].

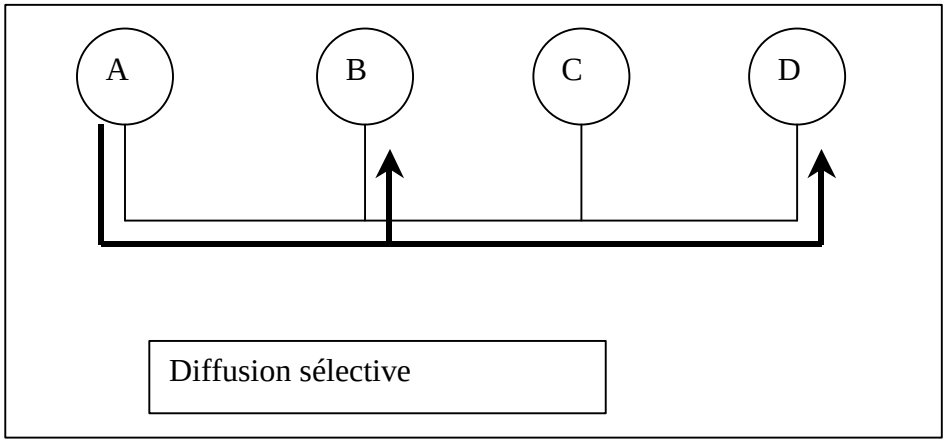

Figure 1 : Diffusion sélective (multicast)

1.3.4.3.2 Diffusion générale (broadcast)

Dans le cas de la communication par diffusion générale de type broadcast, un processus envoie le message simultanément à tous les autres processus. Même ceux qui ne sont pas concernés reçoivent le message et doivent alors détruire ce message dont ils n'ont pas besoin. Dans une approche de communication par broadcast, la sécurité est plus difficile à implanter que dans une approche de communication à diffusion sélective.

Dans l'illustration suivante le processus A envoie le message simultanément au processus B, au processus C et au processus D. Un dispositif spécial affecte le processus C au niveau du micro-noyau. Si le processus C ne fait pas partie du groupe, le message est détruit [TANENBAUM95].

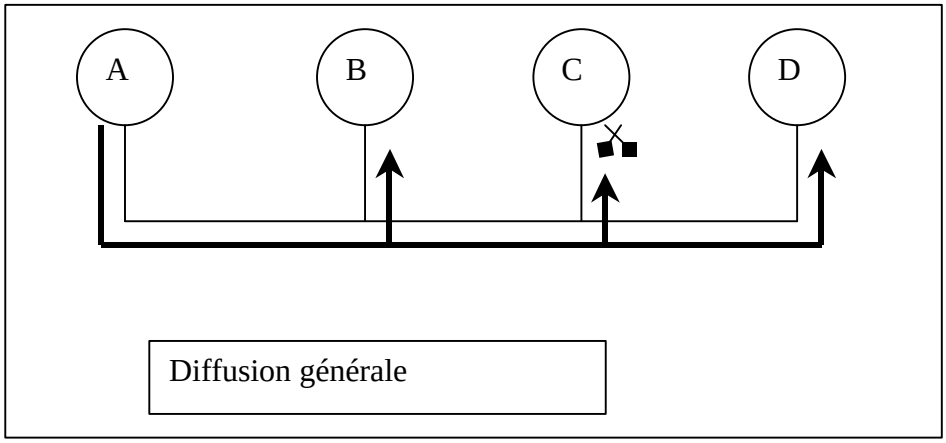

Figure 2 : Diffusion générale (broadcast)

1.3.4.3.3 Communication point à point (unicast)

Dans le cas de la communication en point à point de type unicast, un processus connait la liste des destinataires et envoie le message explicitement à chacun des processus distants concernés et à aucun autre.

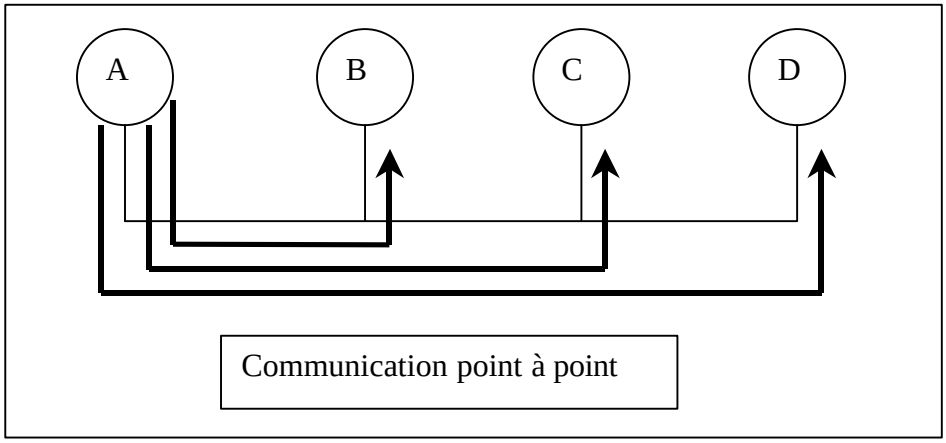

Figure 3 : Communication point à point

1.3.4.3.4 Eléments de communication

Les processus qui se comportent conformément à leurs spécifications sont réputés corrects. Cependant, certains processus peuvent être corrompus et fonctionner en mode dégradé.
Un processus est l'instance d'un programme chargé en mémoire. La suite des instructions qui le décrit doit être vue comme une séquence d'éléments indivisibles. Dans cette approche, les opérations internes à la machine autonome ne sont pas explicitées, si bien qu'il n'existe que deux types d'événements qui peuvent être exécutés. Envoyer un message à un autre processus est noté envoyer(m). Recevoir un message d'un autre processus est noté recevoir(m). Ce point est très important, les messages sont identifiés par les événements envoyer(m), le contenu du message est négligeable par rapport à la méthode. Des messages dont le contenu est identique, mais envoyés par plusieurs processus sont bien des messages différents !
C'est la raison pour laquelle, les développeurs en informatiques sont amenés à utiliser des artifices pour différencier des messages qui semblent identiques bien qu'envoyés à des instants différents. Citons brièvement deux artifices fréquemment utilisés et qui seront explicités ultérieurement : le numéro d'identification unique (nonce), calculé à chaque demande et le numéro chronologique, puisé dans un ensemble de nombres prédéfinis.

1.3.5 Les groupes

Dans le contexte architectural des systèmes répartis, il est inconcevable de maintenir un état global à un coût raisonnable, ce problème est un sérieux obstacle à la communication en univers réparti. De plus amples explications concernant ce point important sont disponibles au chapitre [1.8.2]. Pour résoudre les problèmes de datation, d'atomicité et de cohérence entre processus distants, une idée s'est imposée. Les processus distants doivent coopérer entre eux. Différents types de protocoles de communication possibles ont été énumérés et caractérisés au paragraphe [1.3.4.3].

1.4 La sécurité

Précédemment, au paragraphe [1.1.1], nous avons confronté la terminologie de la sécurité. L'authentification, la confidentialité, l'intégrité et la non-répudiation sont les propriétés logiques et des objectifs précis pour les opposants potentiels du système réparti. La politique de sécurité garantit des assertions de bon fonctionnement en présence d'attaques effectives et formelles. Mais la nature de l'attaque doit être envisagée comme étant quelconque et même non modélisable en termes de probabilités. Il convient de considérer la sécurité d'un point de vue plus qualitatif que quantitatif. De plus, nous avons énuméré les moyens de préserver les propriétés logiques de la sécurité. Intéressons-nous à présent à des principes de base de la sécurité avant de détailler les différents types d'attaques et d'opposants.

1.4.1 Principes de base de la sécurité

Des principes de base de la sécurité doivent être énumérés et expliqués parce qu'ils doivent être respectés pour mettre en place une politique de sécurité fiable avant de dégager une problématique de la sécurité et du temps dans les systèmes répartis.

1.4.1.1 Kerckhoff

Dès 1863, le linguiste Auguste Kerkhoff dégagea certains des principes fondamentaux du chiffrement moderne [STERN98]. En préambule du principe de Kerckhoff relatif à la sécurité, il est possible d'énoncer brièvement que la sécurité ne doit pas être procurée par l'obscurité et l'ignorance. Les détails du principe de Kerckhoff sont repris dans [Menezes96].
Selon ce principe, la sécurité doit résider dans les clefs de chiffrement, pas dans les algorithmes qui devraient être publics, ni dans le message chiffré qui est susceptible d'être intercepté dans sa globalité par l'attaquant. Ainsi, dans la mesure où l'attaquant possède le protocole de chiffrement et la totalité du message chiffré, il ne peut rien en faire, parce qu'il ne dispose pas du temps nécessaire pour remonter jusqu'à la source. Le message en clair reste inconnu. Par exemple, un ordre d'achat en bourse pour le jour même doit rester secret tant que la bourse est ouverte, si l'attaquant réussit à déchiffrer le message au bout de quelques jours, l'obsolescence de l'information est avérée, l'attaquant ne peut retirer aucun bénéfice du temps passé inutilement.

1.4.1.2 Serveur de confiance

Le principe du serveur de confiance (Trusted Third Party) désigne une entité qui recueille les faveurs des entités du système. Toutes les parties sont d'accord pour lui faire confiance. Concrètement, un serveur de confiance peut prendre plusieurs formes :

- Prestataire de service de certification : émet les certificats électroniques.
- Notaire: en cas de certification de document, un notaire établit un acte;
- Juge : en cas de dispute, le juge est capable de procurer une issue au conflit.

1.4.1.3 L'oracle

Le principe de l'oracle provient du mot latin « oraculum » qui signifie sacré. Un oracle garantit une réponse que les dieux envoyaient aux hommes par l'intermédiaire de leurs représentants sur terre. Ainsi, l'oracle du dieu Apollon à Delphes en Grèce antique était célèbre et très lucratif. Une personne du peuple posait une question aux dieux et recevait une réponse par l'intermédiaire de l'oracle.

Dans la théorie de la sécurité et de la complexité, un oracle est un problème A sous-ensemble d'un problème B et qui apporte le même type de solution dans un temps polynomial fini. Cela ne veut pas dire que la solution existe. Si le problème A réduit le calcul polynomial du problème B, alors d'une part B est au moins aussi difficile que A, et d'autre part A n'est pas aussi difficile que B. Par conséquent, si A est un problème bien étudié qui est reconnu infaisable, si de plus A est inférieur à B en terme de complexité, alors on apporte la preuve forte de l'infaisabilité de B [MENEZES96].
Le principe de l'oracle est à la base de nombreuses attaques mais n'est pas un type d'attaque, c'est un principe de base de la sécurité.

1.4.1.4 Le maillon faible

Le principe de maillon faible a été posé dans le chapitre dédié aux systèmes répartis [1.3], l'existence d'un Single Point Of Failure (SPOF) est acceptable sous certaines conditions. Ce principe s'applique également à la politique de sécurité. Si une faiblesse existe quelque part, et s'il n'existe aucun dispositif redondant pour doubler la protection à cet endroit du système, alors la sécurité s'écroule et le système est investi. De plus, le niveau de sécurité d'un système dépend du niveau de sécurité de son maillon le plus faible.

Nous avons par conséquents exprimés des principes de base de la sécurité en termes nets.
Ces hypothèses reconnues et communes à tous les systèmes dits sécurisés, devraient être envisagées et assimilées avant de concevoir, réaliser et mettre en place un quelconque crypto-système. Ce système sera de toute manière attaqué, que ce soit de l'intérieur ou de l'extérieur.

1.4.2 Type d'attaques contre le système

Des attaques existeront toujours, établissons une liste des attaques possibles contre le système. Tout d'abord, il convient de différencier l'attaque passive de l'attaque active.

1.4.2.1 Attaque active, attaque passive

L'attaque active a lieu quand l'opposant essaie de modifier, voire de détruire l'information. L'opposant peut également attaquer activement le système en s'attaquant au canal de transmission. Dans ce type d'attaque, les propriétés visées sont l'intégrité et l'identification.
L'attaque passive a lieu quand l'opposant enregistre les messages qui transitent sur le canal de transmission dans le but de lire les messages. L'attaque passive est dirigée contre la propriété de confidentialité. Elle est dangereuse dans la mesure où elle peut passer complètement inaperçue. Le propriétaire des données peut ignorer que la confidentialité a été brisée.
Quand une personne dirige une attaque passive à l'encontre des deux personnes qui partagent une relation d'échanges, le canal de communication n'est pas brisé, les données ne sont pas altérées, mais la confidentialité n'est pas assurée.

1.4.2.2 Attaque du milieu

1.4.2.2.1 Introduction

L' attaque du milieu est aussi appelée attaque par interception. C'est un type d'attaque active. L'opposant reste anonyme, il écoute les messages, les interprète et les renvoie dans le système en se faisant passer pour des entités du système.

1.4.2.2.2 Définition

Dans le cadre d'une communication bipoints entre A et B, l'attaque du milieu repose sur le principe suivant : Estelle se fait passer pour Bob par rapport à Alice, et Estelle se fait passer pour Alice par rapport à Bob. Ceci permet à Estelle de détourner des actifs à son profit.

1.4.2.2.3 Commentaire

Quand une attaque du milieu est effectuée par un opposant, chaque interlocuteur du système croit dialoguer directement avec son interlocuteur privilégié, alors qu'il n'en est rien. Chaque message envoyé par un des acteurs du système est intercepté par l'opposant puis renvoyé immédiatement à son destinataire licite. Peut importe, si l'auteur de la mascarade retire un quelconque bénéfice de cette attaque par interception, le comportement de l'intrus est imprévisible, sujet à critique et contraire à la confidentialité des contrats. Regarder par le trou de la serrure, écouter aux portes est symptomatique d'un comportement répréhensible. L'attaquant peut apprendre beaucoup d'informations sur le système, dans le but de prendre l'identité d'une entité du système. Quand la supercherie est découverte, il est souvent trop tard. Sur Internet, personne ne sait vraiment qui vous êtes, quelle est votre identité, voire qui vous représentez. Le besoin de se mesurer à autrui, la volonté de dépasser son adversaire, voire l'espoir d'un gain facile représentent autant de motivations différentes.

1.4.2.2.4 Exemple : la fraude à la carte bancaire

La fraude à la carte bancaire est un exemple d'attaque du milieu. Ce type d'activité est nuisible et relève de l'escroquerie. Le site du département de la justice américaine (D.O.J.) explique avec moult détails le mécanisme de la fraude à la carte bancaire dans le but de sensibiliser les utilisateurs, de les rendre plus méfiants et de les inciter à souscrire des contrats intégrants des protocoles sécurisés de l'Internet (SSL par exemple). Le principe de cette escroquerie est le suivant :

Maurice prend l'identité d'un honnête commerçant et fait de la prospection publicitaire. Maurice propose une caméra vidéo à Alice, il promet de livrer rapidement une caméra payable à la livraison. Alice, trouve l'offre alléchante, le produit merveilleux et le prix compétitif, elle se laisse donc séduire par Maurice. Celui-ci, de son côté, se fait maintenant passer pour Alice ; Maurice commande une caméra vidéo à Bob en utilisant un numéro de carte bancaire volé par Estelle. Il obtient ainsi un matériel neuf qu'il échange au domicile d'Alice contre de l'argent liquide ou un chèque bancaire qu'il n'a plus qu'à déposer sur un compte bancaire situé dans un paradis fiscal. Dans cet exemple, maintenant que la fraude est avérée, Alice en toute bonne foi détient un matériel non payé au fournisseur, et celui-ci est en droit d'engager des poursuites.

L'illustration suivante modélise la fraude à la carte bancaire :

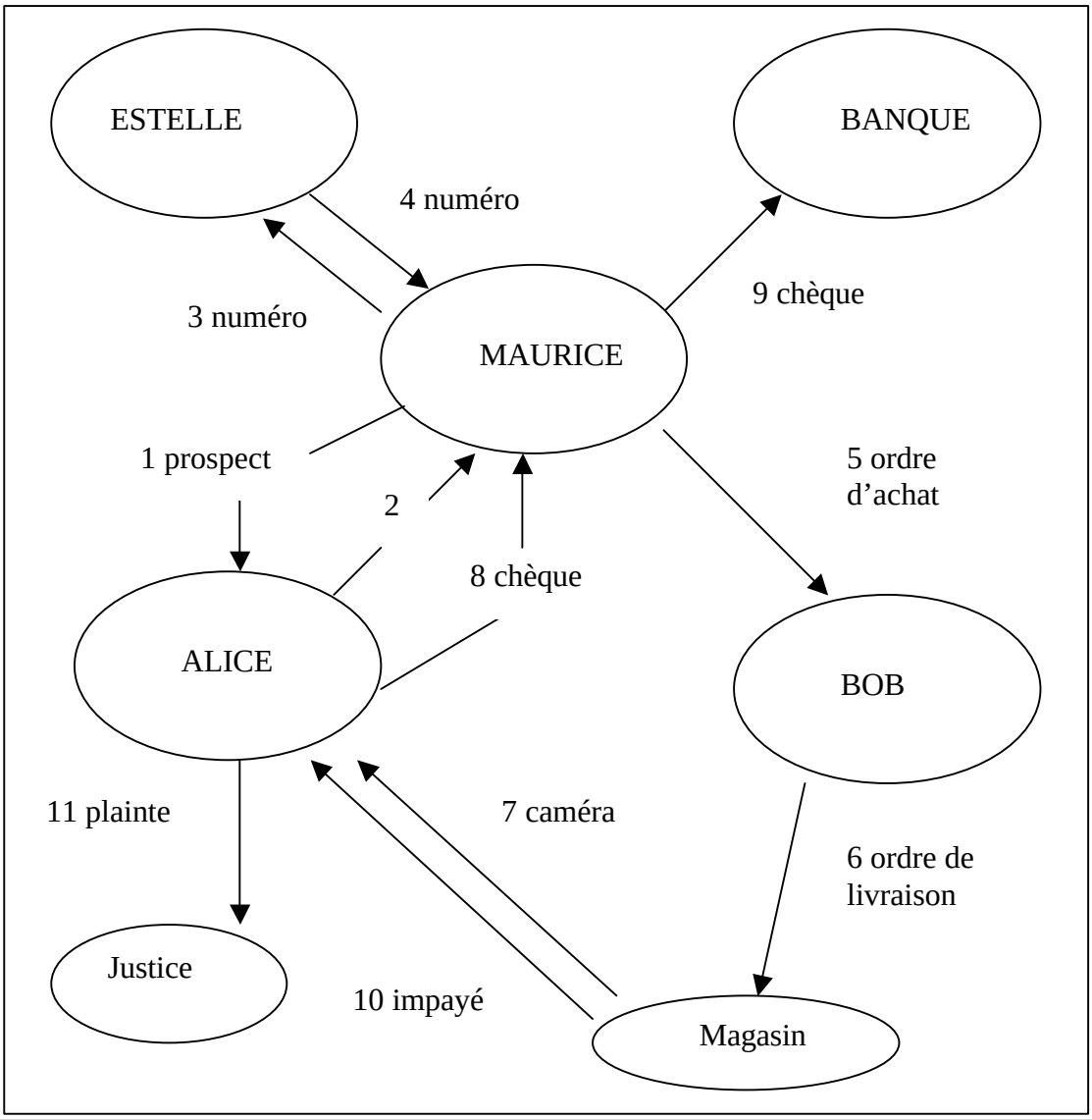

Figure 4 : Fraude à la carte bancaire

1. Maurice envoie un prospectus à Alice
2. Alice envoie un accord de principe à Maurice
3. Maurice demande à Estelle un numéro de carte bancaire
4. Estelle fournit à Maurice un numéro de carte bancaire dérobée
5. Maurice envoie à Bob un ordre d'achat au nom d'Alice
6. Bob émet un ordre de livraison
7. La livraison de la caméra est effectuée au domicile d'Alice
8. Alice donne un chèque bancaire à Maurice
9. Maurice envoie le chèque à sa banque pour encaissement
10. David, dans le magasin se rend compte d'une erreur. David émet un avis d'impayé
11. Alice porte plainte en justice

Alice, Bob et David devraient être plus méfiants à l'égard d'Estelle et de Maurice.
Ils devraient souscrire des contrats intégrants des protocoles sécurisés de l'Internet.

Nous avons considéré un exemple d'attaque par interception, nous allons maintenant nous intéresser à différents types d'attaques utilisés en crypte-analyse. Certes, l'attaque sur le chiffrement semble être une activité privilégiée par l'opposant, mais les types d'attaques varient en fonction du degré de connaissance du système et du type de chiffrement utilisé.

1.4.2.3 Attaque directe sur le chiffrement

L'attaque directe sur le texte chiffré est la plus triviale des attaques. Dans ce cas de figure, l'opposant trouve directement le texte en clair qui correspond au texte chiffré. Si l'attaque directe sur le chiffrement est une réussite, cela prouve que l'insécurité du système est maximale. Dans ce cas, la politique de sécurité est donc très faible et doit être revue.

1.4.2.4 Attaque à partir de texte connu

Si l'opposant possède des textes en clairs et les textes chiffrés correspondants, alors l'opposant peut procéder à une attaque à partir de texte connu. Il cherche à mettre en évidence la méthode de chiffrement.

1.4.2.5 Attaque à partir de texte en clair choisi

Dans ce type d'attaque, l'opposant ne connaît pas le texte chiffré qui correspond au texte en clair choisi. Mais si l'opposant connaît le type protocole utilisé, il peut procéder à une attaque à partir de texte en clair choisi. Il génère des textes chiffrés à partir des textes en clair qu'il a choisi. Prenons un exemple. Pour attaquer le système de chiffrement de Blaise Vigenère qui date du dix-huitième siècle, l'attaquant choisit le texte constitué uniquement à partir de lettres « a ». Puisque la lettre « a » correspond au chiffre zéro qui est l'élément neutre de l'addition sur l'ensemble des nombres entiers, en faisant fonctionner la machine à chiffrer, l'attaquant obtient en sortie un texte qui correspond à la clé de chiffrement.

1.4.2.6 Attaque adaptative à partir de texte en clair choisi

Dans ce type d'attaque, l'opposant choisit les textes en clair en fonction de textes chiffrés qu'il possède, et il essaie par approximations successives d'obtenir le texte en clair qui correspond au texte chiffré qu'il possède.

1.4.2.7 Attaque à texte chiffré choisi

L'opposant choisit un texte chiffré et génère le texte en clair correspondant. Ce type d'attaque suppose que l'attaquant possède les moyens nécessaires pour décoder l'information en clair à partir du texte chiffré.

1.4.2.8 Attaque adaptative à texte chiffré choisi

L'opposant choisit un texte chiffré en fonction de textes en clairs obtenus précédemment au moyen d'une attaque à texte chiffré choisi. Dans ce type d'attaque, l'opposant procède par approximations successives jusqu'à obtenir les moyens nécessaires pour décoder l'information en clair à partir du texte chiffré.

1.4.2.9 Attaque aléatoire

Dans ce type d'attaque, l'opposant envoie des messages successifs à l'oracle qui lui répond sommairement, soit par oui ou par non, soit en procurant un comportement déchiffrable. L'opposant essaie de briser le crypto-système ou plus prosaïquement d'acquérir des indices, d'apprendre des choses qui concernent les propriétés de la sécurité.

Bien entendu, il est possible de combiner plusieurs types d'attaques.

1.5 Techniques cryptographiques

1.5.1 Introduction

L'objectif désigné pour les techniques cryptographiques est de fournir des outils fiables pour que les propriétés de la sécurité soient satisfaites pendant le déroulement des protocoles.
Définition : un protocole est un ensemble de règles et d'étapes successives suivies par chaque entité reliée au canal de communication.
Lors de la spécification d'un protocole sécurisé, il faut partir du principe de base qu'une entité va essayer de le casser et que cette entité est très puissante. La capacité du système à résister à des attaques menées à l'encontre de son dessein doit être maximale, c'est la sécurité maximale. Si la métrique utilisée pour mesurer cette capacité est le temps exprimé en secondes mis par l'attaquant pour trouver une brèche et l'exploiter, alors cette quantité de secondes doit tendre vers l'infini.
L'objectif de ce chapitre n'est pas d'établir une liste exhaustive ou une taxinomie des protocoles de cryptographie. Le lecteur intéressé par une telle approche trouvera dans [MENEZES96] toutes les informations utiles à cet effet. Notre propos réside plutôt dans une brève explication des techniques cryptographiques crypto-graphiques utilisées ou évoquées dans ce mémoire : protocoles symétriques, protocole asymétriques, algorithmes à fonctions de hachage et technique du nonce.

1.5.2 Protocole symétrique

Dans un protocole symétrique, la clef utilisée pour chiffrer le message avant de le transmettre, est identique à la clef utilisée pour déchiffrer le message reçu à l'autre bout du canal de communication. Les deux entités utilisatrices du protocole doivent donc se mettre d'accord préalablement sur la nature de la clef de chiffrement qui est aussi utilisée lors du déchiffrement. Le protocole symétrique est utilisé dans ce mémoire, voir à ce sujet le paragraphe dédié aux fonctions de hachage [1.5.4].

1.5.3 Protocole asymétrique

Le protocole asymétrique est fondé sur la théorie des nombres premiers et plus particulièrement sur les travaux de Diffie-Hellman. La signature électronique sécurisée est construite en partie en utilisant ce type de technique cryptographique. Ce type de protocole résout très bien le problème d'Alice qui veut envoyer un message à Bob en étant quasiment sûr que personne d'autre que Bob ne puisse le lire. De plus, Bob est presque sûr qu'Alice le message a été émis par Alice et personne d'autre.

1.5.4 Algorithmes à fonction de hachage

Au plus haut niveau de classification, les fonctions de hachage peuvent être divisées en deux groupes : les fonctions sans clef, dont les spécifications ne requièrent qu'un paramètre en entrée (le message); et les fonctions de hachage avec clef dont les spécifications requièrent deux paramètres en entrée (le message et la clef). Dans ce mémoire, nous considérons exclusivement les algorithmes à fonction de hachage à sens unique et sans trappe (sans clef).
La fonction de hachage calcule un résumé ou empreinte, de longueur fixe à partir du message d'origine. Quelque soit la longueur du message d'origine, une fois que le calcul effectué par la fonction de hachage est terminé, la longueur du résumé est toujours la même. Par exemple, dans le cas du protocole sécurisé de hachage SHA-1, la longueur du résumé est de vingt (20) octets. De plus, si le message d'origine est digéré une deuxième fois par cette fonction, le résultat est identique à celui obtenu la première fois.
La deuxième caractéristique fondamentale des algorithmes à fonction de hachage assure qu'une infime variation du texte d'origine entraîne une modification quasi totale de l'empreinte qui résulte de la fonction de hachage.
Le type de protocole qui utilise des fonctions de hachage pour chiffrer les données induit des temps de réponse bien plus courts que les protocoles asymétriques (RSA par exemple) dont les calculs effectués lors du chiffrement et du déchiffrement sont coûteux en énergie.

1.5.4.1 Fonction à sens unique

Une fonction de hachage est une fonction à sens unique. Les fonctions à sens unique sont construites à partir du DES ou de n'importe quel bloc de chiffrement E qui se comporte comme une permutation de code aléatoire.

Soit x le message
Soit y le résumé
Soit E le bloc de chiffrement (DES)
Soit k une fonction de cryptographie
Soit O l'opérateur ou exclusif

$Y = f(x) = E\ k(x)\ O\ x$

Pour un y quelconque, trouver x et la clef E est difficile parce qu'il n'y a pas de meilleur choix pour x que le choix aléatoire. À partir du résultat il est très difficile de reconstituer le message d'origine. La difficulté réside dans la complexité induite par les grands nombres, mais surtout par celle de la reconnaissance d'un texte intelligible. En effet, une attaque exhaustive permettrait de reconstituer plusieurs messages, mais comment être sûr de choisir le bon message ? Le fait de poser la question est une réponse en soi. C'est un problème difficile.
Les protocoles MD-5 et SHA-1 sont des protocoles à fonction de hachage. MD est l'acronyme de Message Digest traduisible par Résumé de message. SHA est l'acronyme de Secured Hached Algorithm traduisible par Algorithme de hachage sécurisé.
La type de méthodes choisi pour construire ces fonctions de hachage spécifique (MD5 et SHA-1) utilise les techniques de la cryptographie symétrique. Les deux fonctions les plus utilisées sur l'Internet sont des améliorations du MD4 qui fut inventé par Rivest en 1990 [NATKIN02]. En effet, SHA-1 s'appuie sur des opérations d'addition, de décalage et de ou exclusif qui sont faciles à câbler dans un circuit intégré et ne consomment pas beaucoup d'énergie. De plus amples descriptions concernant les fonctions de hachage sont disponibles dans [NATKIN02]. Le nombre d'opérations effectuées, les variables utilisées, les choix d'initialisation et le type de résultat obtenu sont répertoriés et précisés.

1.5.4.2 Fonction qui résiste à la collision

Une fonction de hachage doit résister à la collision, ce qui signifie qu'il est très difficile de trouver deux messages différents qui produisent le même résumé. De nombreuses recherches poursuivent cet objectif. À titre d'exemple, la méthode qui s'appuie sur le paradoxe des anniversaires est beaucoup plus rusée que l'attaque par force brute. Le paradoxe des anniversaires expose que la probabilité qu'il existe deux personnes partageant la même date d'anniversaire dans une assemblée de 23 « personnes aléatoires » est au moins un demi [STINSON96]. Le paradoxe n'en est pas un, mais le calcul par intuition est éloigné de 23. Après tout, un enfant candide, sait qu'une année est faite de 365 jours, il estime alors que 365 jours génèrent 365 possibilités pour une date d'anniversaire donnée. Un autre calcul surprend l'intuition. En effet, il faut réunir aléatoirement 183 personnes pour obtenir plus d'une chance sur deux de trouver une personne dont la date anniversaire coïncide avec la date choisie au préalable. Cette méthode permet d'envisager des calculs qui durent beaucoup moins longtemps que les calculs fondés sur la force brute. Une parade possible consiste à augmenter la taille minimale du résumé généré par la fonction de hachage.

1.5.4.3 Sécurité des fonctions de hachage

La taille du résumé est directement en relation avec le type de sécurité procuré par la fonction de hachage, comme le précise la définition de la sécurité idéale suivante de [Menezes96]:
Définition : Une fonction de hachage sans clef qui fournit un résumé sur n bits présente une sécurité idéale si les deux caractéristiques suivantes sont toutes les deux réunies :

- À partir d'un résumé, obtenir un texte pré-image et un deuxième texte pré-image nécessite à chaque fois 2^n opérations.
- À partir d'un résumé, obtenir une collision nécessite environ $2^{n/2}$ opérations.

Par conséquent, nous pouvons affirmer que les algorithmes à fonction de hachage fournissent un niveau de sécurité tout à fait acceptable. Une implantation de ce type d'algorithme dans le protocole de ce mémoire est tout à fait envisageable.

1.5.5 Le nonce

Le nonce désigne un nombre généré de manière aléatoire, et utilisé à une seule occasion.
Dans le contexte des systèmes distribués, le terme de fraîcheur d'un message s'applique pour qualifier un message dont l'émission précède la diffusion d'au plus quelques pulsations d'horloge. Plus la durée est courte et plus le message est considéré comme étant frais. Dans le contexte des systèmes répartis et de la relation causale, un message est frais dans le déroulement d'un protocole à condition que son contenu ne soit pas apparu précédemment lors des échanges, des étapes successives effectuées dans un ordre conventionnel précis. Le plus souvent, quand la fraîcheur est étudiée dans le contexte de la cryptographie, c'est dans le but d'empêcher les attaques par répétition. Un intrus répète des messages précédents du protocole dans l'espoir de convaincre les participants qu'il est l'un d'eux. Ainsi il espère en apprendre un peu plus sur le système, et il peut même se substituer à l'un des participants. Cette technique porte le nom de mascarade. La parade de la mascarade est le nonce. Ce nombre est fabriqué pour être utilisé seulement une fois, pour le temps présent et à cette occasion exclusivement. Le générateur de nombre aléatoire doit être choisi de manière judicieuse pour garantir que le nombre fabriqué et vraiment un nonce. Dans le domaine de la cryptographie, le fait que la probabilité de générer deux fois le même nombre soit infime est satisfaisant. Ainsi, la probabilité de fabriquer une deuxième fois le même nombre est si faible que cette éventualité est qualifiée d'impossible. Prenons un exemple d'utilisation du nonce. Alice fabrique un nonce au moyen d'un générateur de nombre aléatoire. Elle cache le nonce dans un message. Alice envoie un message à Bob. Celui-ci le reçoit et envoie un accusé de réception dans lequel il a recopié le même nonce. Ainsi, Alice peut raisonnablement être convaincue de la sincérité de Bob, dans la mesure où elle est sûre que Bob ne rejoue pas un message précédent. Toutefois, la technique du nonce ne suffit pas à se prémunir de l'attaque du milieu. En effet, si Estelle intercepte le message, le lit et renvoie le nonce à Alice, celle-ci estime que c'est Bob qui lui écrit alors qu'il n'en est rien.
Par conséquent le nonce permet de se prémunir des attaques par mascarade, mais le nonce ne suffit pas à légitimer l'échange de messages entre deux entités.

1.5.6 Chaînes non modifiables

Dans une première approche, imaginons succinctement un protocole dans lequel chaque message qui circule contient un horodatage calculé à partir d'une base de temps de confiance selon les normes de l'organisation Internet Engineering Task Force (IETF). Cet horodatage (publié à intervalles réguliers) contient lui-même une empreinte des horodatages précédents. Ainsi, la chaîne des messages est non-modifiable. Zn est le nième document à horodater. H() est une fonction de hachage. Ln est un résumé de longueur fixe. L'autorité d'horodatage (serveur de confiance) calcule Ln = H (Zn, Ln-1). Nous obtenons l'illustration suivante selon [BOURROUILH01] :

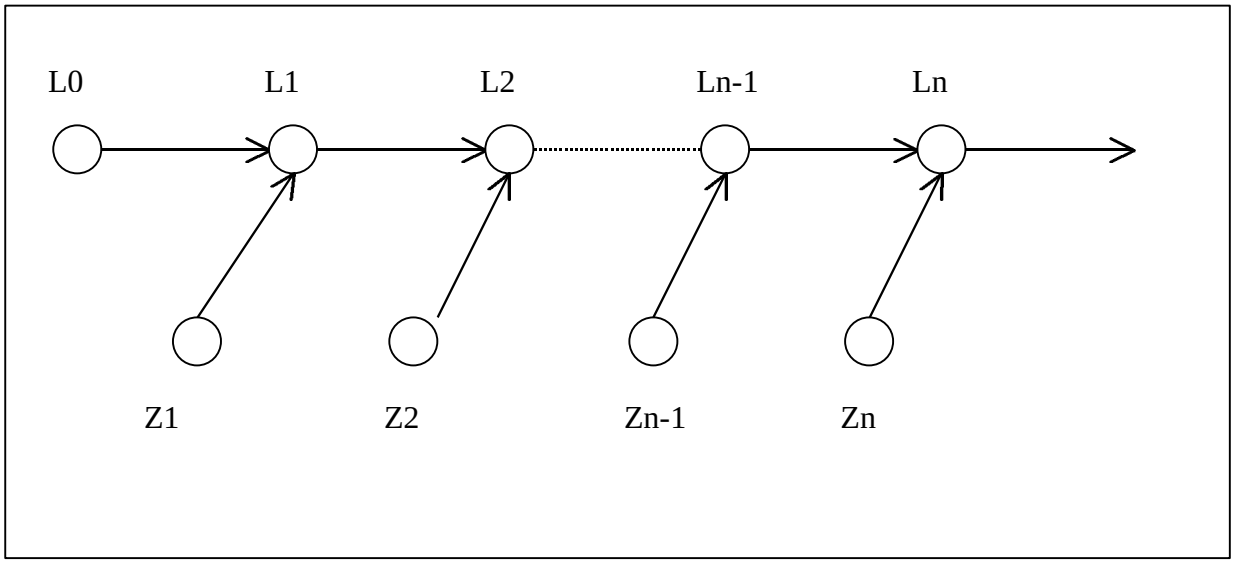

Figure 5 : Chaînage linéaire des documents

Dans une deuxième approche, imaginons maintenant un protocole dans lequel chaque message qui est émis contient, en plus du message lui-même, une empreinte. Cette empreinte (ou résumé) est calculée à partir de ce message et de tous les messages précédents au moyen d'une fonction de hachage à sens unique et sans collision.
Dans l'illustration suivante, le message M_1 est réduit pour obtenir le résumé R_1. De plus, le message M_1, le résumé R_1 et l'historique H_0 sont réduits pour construire l'historique H_1, le message M_1 et l'historique H_1 sont alors émis. Ensuite, lors de l'itération suivante du protocole, le message M_2 est réduit pour obtenir le résumé R_2. De plus le message M_2, le résumé R_2 et l'historique H_1 sont réduits pour construire l'historique H_2, le message M_2 et l'historique H_2 sont alors émis. Ainsi, de proche en proche, l'empreinte est modifiée par le résultat du calcul relatif au message de l'itération en cours.
En ce qui concerne la dernière itération de l'illustration suivante, le message M_n est réduit pour obtenir le résumé R_n, de plus, le message M_n, le résumé r_n et l'historique H_{n-1} sont réduits pour construire l'historique H_n qui sera envoyé avec le message M_n.

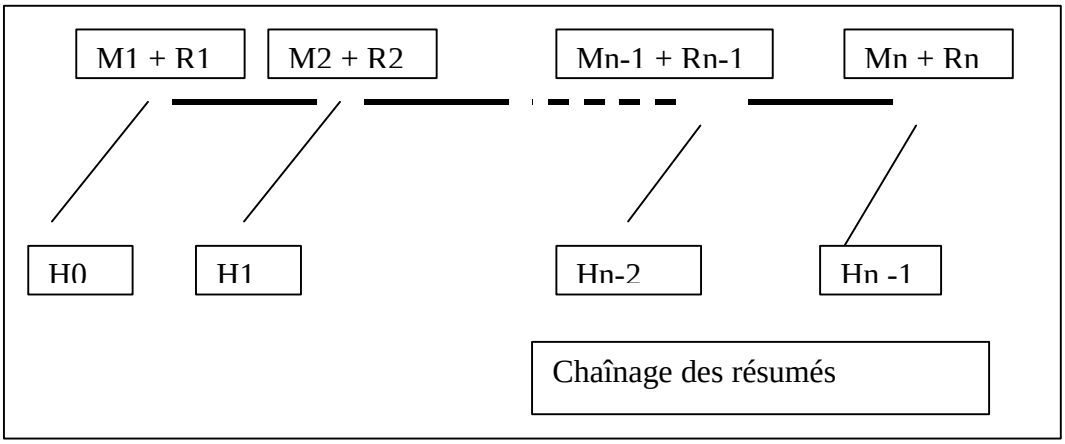

Figure 6 : Chaînage des résumés

Le principal avantage de cette solution du chaînage des résumés par rapport à la solution du chaînage des documents réside dans le fait que la relation de causalité des messages est enregistrée. Dans la première solution, deux messages indépendants peuvent paraître a priori liés, dans la deuxième solution par contre, les messages sont forcément liés si le résumé les contient tous les deux.

1.6 La problématique

En matière de sécurité, les propriétés liées à l'authentification et à l'intégrité des données sont très souvent explorées lors de la mise en place d'une politique de sécurité. La propriété de non-répudiation est souvent confondue avec la propriété d'intégrité. Cet amalgame et la notion d'intégrité telle qu'elle est exprimée en droit des affaires nous amène logiquement à envisager la sécurité sous l'angle de la propriété de non répudiation des événements qui se sont produits dans le passé.

L'envoi de courriers électroniques signés non répudiables constitue l'un des services les plus prometteurs et des plus demandés par le marché. Mais avant de songer à appliquer notre recherche à un quelconque débouché commercial, il est préférable d'envisager une recherche fondamentale sur le sujet qui nous préoccupe, c'est à dire la sécurité et le temps dans les systèmes répartis.

Le mode message constitue le mode privilégié de la communication asynchrone en système réparti. D'autre part, maintenir une heure identique sur toutes les machines caractérise une solution très coûteuse qui peut causer de graves désagréments. Des explications sont disponibles au paragraphe [1.8]. La datation doit être envisagée à partir d'un marquage et du point de vue de la relation causale qui existe entre les événements du système.
La problématique de la sécurité et du temps dans les systèmes répartis est donc posée. Les définitions qui permettent de construire des protocoles de diffusion de messages entre les entités du système doivent être modifiées de manière à éviter les problèmes liés à la propriété de non-répudiation des messages.
Par conséquent, il s'agit de trouver un moyen d'énoncer différemment les définitions de communication ordonnée pour que la propriété d'ordre ne soit répudiable ni par origine, ni par destination.

1.7 Sécurisation de l'ordre causal

Les événements du système peuvent être liés par une relation d'ordre causal ou bien peuvent être indépendants entre eux. Des explications sont disponibles au paragraphe [1.8]. La détection de l'ordre causal doit faire partie des préoccupations majeures en vue d'assurer la sécurité du système réparti. Une fois que l'ordre causal est mis en évidence, sa préservation doit être envisagée car l'opposant qui ne doit être sous-estimé en aucun cas, va mettre en œuvre des moyens pour s'attaquer à certains objectifs en liaison avec l'ordre causal.

Nous introduisons deux notions qui expriment ces objectifs. La destruction de la causalité existante d'une part, la construction d'une causalité artificielle d'autre part sont les deux moyens que l'attaquant peut utiliser pour détourner le système de son dessein.
Dans la littérature, le terme denial est utilisé pour qualifier la notion de destruction de la causalité, le terme de forgery est utilisé pour qualifier la notion de fabrication de la causalité [REITER99]. Pour définir plus exactement les notions de denial et de forgery, il est préférable dans un premier temps de modéliser les algorithmes de détection de la causalité en utilisant un prédicat qui agit sur une paire de messages. Nous considérons qu'un processus évalue le prédicat pour déterminer si un message m précède un message m' en respectant un ordre de relation causale.

Un processus P dispose d'un historique d'événements qui induit des connaissances sur l'ordre causal. Notons, suivant [REITER99]
C(m1,m2) est vrai si, à partir des informations qu'il possède, le processus estime que m1 précède m2. Notons que si le processus est fiable alors m1 précède m2.

C(m1,m2) est faux si, à partir des informations qu'il possède, le processus estime que m1 ne précède pas m2. Notons que si le processus est fiable alors m1 ne précède pas m2. Cela peut vouloir dire aussi que m1 et m2 sont indépendants. Alors, on ne peut rien conclure quant à la précédence entre m1 et m2.
Le message m1 précède le message m2 est noté m1 --> m2
Le message m1 ne précède pas le message m2 est noté m1 -/-> m2
Nous choisissons un comportement pour le prédicat, tel que
C(m1,m2) = { vrai si m1--> m2, faux sinon }
A partir du prédicat C, nous sommes en mesure de définir les notions de « denial » et de « forgery » qui peuvent se produire à l'encontre de l'ordre causal, si le système est faible à l'encontre de ces notions, qu'il s'agisse d'attaques accidentelles ou d'attaques malignes.

1.7.1 Preuve faible et preuve forte

Dans ce mémoire nous abordons une approche de preuve que nous baptisons schéma de preuve faible et schéma de preuve forte.

PREUVE FAIBLE : le protocole de communication de la signature fournit un mécanisme de preuve faible à Alice qui lui permet de prouver que en ce qui la concerne, elle a bien agi.

PREUVE FORTE : le protocole de communication de la signature fournit un mécanisme de preuve forte à Alice dans la mesure où toutes les entités du système acceptent d'appliquer ce protocole. Alors, si une des entités quelconque du protocole a mal agi, cette entité ne peut pas prouver avoir bien agi et s'exclut elle-même du protocole. Si Bob à mal agi, il n'est pas en mesure de prouver qu'il a bien agi.

Soit un protocole P, mettant en jeu N participants. Supposons que P s'appuie sur une voie de communication fiable, c'est à dire qui tolère les défaillances transitoires et signale les défaillances permanentes. Parmi les participants, nous distinguons les participants de confiance qui exécutent P selon sa spécification et les participants byzantins dont le comportement est imprévisible. Nous dirons qu'une exécution de P est correcte si tous les participants exécutent P selon sa spécification. Une exécution incorrecte est dite byzantine.

Soit e et e' deux événements définis dans le cadre d'une exécution de X de P. Il peut s'agir d'une émission ou d'une réception de message. Nous notons O(X,e,e'), le fait que e précède causalement e' ou que e' précède causalement e dans l'exécution E de P.
Soit i un participant de confiance, notons Ci(E,e,e') l'assertion qui est vraie si i déduit O(X,e,e') à partir des messages (ordre et valeur) qu'il a reçus et envoyés lors d'une exécution X de P et de la spécification de P. Notons Ci(e,e') si Ci(X,e,e') pour toute exécution X correcte.
Considérons une exécution XB de P byzantine.
Nous définissons que cette exécution détruit la causalité s'il existe au moins un couple d'événements e et e' et un participant de confiance i qui déduit (non Ci(XB,e,e')) alors que Ci(e,e'). Autrement dit i déduit que e et e' ne sont pas ordonnés alors que toute exécution correcte l'aurait amené à conclure le contraire.
Nous définissons que cette exécution fabrique la causalité s'il existe au moins un couple d'événements e et e' et un participant de confiance i qui déduit Ci(XB,e,e')) alors que non O(XB,e,e'). Autrement dit i déduit que e et e' sont ordonnés alors que dans cette exécution ces messages ne sont pas ordonnés.

Soit X une exécution de P, G une coalition (sous-ensemble) de participants, CAUS(X,G) une assertion construite comme une propriété logique du premier ordre dont les termes sont les assertions Ci(X,e,e'), i appartenant à G. CAUS(X,G) est une propriété que G peut affirmer comme étant vraie en montrant comme élément de preuve tous les éléments permettant de déduire chaque Ci(X,e,e'). Si CAUS(X,G) est vraie pour toute exécution correcte de P, alors CAUS est une des propriétés de la spécification de P. Ceci sera noté CAUS(G).

Considérons un protocole P qui rend un certain service fonctionnel (le courrier électronique par exemple). Sécuriser P du point de vue du temps revient à construire un protocole S(P) qui rend le même service mais qui permet de prouver certaines propriétés de type CAUS(X,G), non prouvables dans P.
Plus formellement considérons un protocole S(P) déduit de P. Les participants de P sont un sous-ensemble des participants de S(P). S(P) doit-être tel qu'il existe par une injection des événements E de la spécification de P vers les événements de E' de la spécification de S(P). A tout événement e de E est associé un événement de E' noté S(e). Et à toute assertion CAUS(X,G) de P est associée une assertion S(CAUS(X,G)) de S(P) en substituant S(e) dans tous les termes Ci(X,e,e').
Soit G une coalition de participants de confiance. Nous dirons que S(P) est une sécurisation faible de P pour une assertion CAUS(G) si CAUS(X,G) est vrai pour toute exécution correcte de P alors S(CAUS(G)) pour toute exécution de S(P). Autrement dit les membres de G pourront toujours prouver qu'ils ont exécuté S(P) selon un ordre qui a vérifié l'assertion.
Soit deux messages m et m' envoyés par deux participants de G. Supposons que pour toute exécution correcte X de P CAUS(X,G) implique que l'émission de m précède l'émission de m'. Nous voulons maintenant que les byzantins ne puissent nier avoir reçu les messages ou déclarer les avoir reçus dans un ordre inversé.

A partir de CAUS(X) construisons une assertion RECEPTION(X,RG) telle que pour toute émission e de message m apparaissant dans CAUS(X) il existe une réception de m dans RECEPTION(X,RG) et RECEPTION(X,RG) traduit que l'ordre de réception est isotone à l'ordre d'émission. RG est constitué des participants récepteurs des messages émis par G.

Nous dirons que S(P) est une sécurisation forte de P pour une assertion CAUS(G) si S(P) est une sécurisation faible de P pour CAUS(G) et pour RECEPTION(X,RG). Autrement dit les membres de G pourront toujours prouver qu'ils ont exécuté S(P) selon un ordre qui a vérifié l'assertion et aucune coalition de byzantins ne pourra exhiber une preuve falsifiée qui permette de supposer qu'ils n'ont pas reçu les messages ou qu'ils les ont reçus dans le désordre.

1.7.2 Destruction de la causalité

Une relation causale est détruite au sens du prédicat C(m1,m2), s'il existe des messages m1 et m2 tels que m1 précède m2, mais pour un processus correct, C(m1,m2) est évalué à faux.
Dans l'exemple suivant, le message m_3 précède le message m_4, mais pour Maurice, C(m_3,m_4) est évalué à faux.
Alice donne l'ordre d'acheter des actions auprès des courtiers Bob et Christine, Alice émet le message m_3 vers Christine et le message m_1 vers Bob. Christine n'a pas assez d'actions pour faire face à la demande d'Alice et s'adresse à Maurice. Christine envoie le message m_4 à l'intention de Maurice. Le cours de l'action va avoir tendance à monter conformément au principe de l'offre et de la demande ; Maurice, frauduleusement va s'enrichir en niant avoir reçu le message m_4, et il affirme que le message m_2 est antérieur à l'émission du message m_1. Ainsi, il s'efforce de bénéficier d'un cours de l'action plus avantageux pour lui.

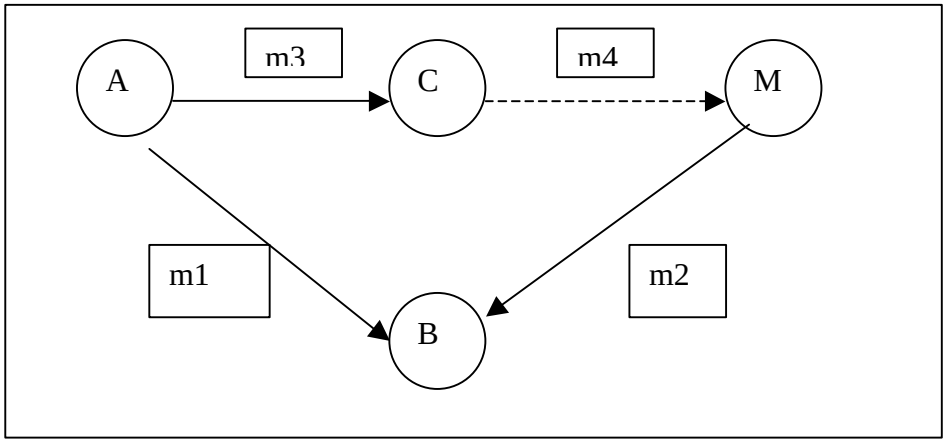

Figure 7 : Destruction de la causalité

Les messages sont totalement ordonnés, mais Maurice fait croire que la causalité temporelle est fausse. Pour cela, il nie avoir reçu le message m_4.

Des algorithmes existent pour empêcher la destruction de la relation d'ordre causal. Ces algorithmes doivent vérifier la relation suivante :
Si m_1 --> m_2 alors C(m1,m2) est vrai pour toutes les livraisons de m1 et de m2.

Pour empêcher l'occurrence d'apparition de la destruction de la causalité, nous envisageons deux sortes d'algorithmes. Le premier est fondé sur l'existence d'un serveur de confiance, le deuxième est fondé sur l'approche conservatrice dans laquelle la diffusion locale est imposée.

1.7.2.1 Prophylaxie par serveur de causalité

Un serveur de causalité est un serveur de confiance qui agit comme un intermédiaire, entre les processus du système réparti. Chaque processus envoie chacun des messages au serveur de causalité qui est implanté de manière centralisée sur un des hôtes du réseau. Celui-ci est garant de l'ordre causal puisque tous les messages passent par lui. Il agit comme un dispositif de routage, car il ré-émet chacun des messages à l'intention de son destinataire effectif. Ainsi, un processus n'est pas en mesure de communiquer directement avec son interlocuteur, il doit s'adresser à l'interlocuteur privilégié, le serveur de causalité. Cette approche est une approche de communication centralisée. La relation de causalité C(m1,m2) est garantie.

Dans cette approche, la suppression d'un message est impossible puisque le serveur de causalité est le garant de l'ordre causal. Les messages sont ordonnés selon l'ordre d'apparition dans le canal de communication dont le centre est constitué par le serveur de causalité. Toutefois, si une entité refuse de reconnaître avoir reçu un message, la destruction de causalité est tout de même effective.

Reprenons l'exemple des courtiers en bourse. Alice souhaite envoyer le message m1 à Bob. Cette fois, le message doit transiter par le serveur de causalité qui le renvoie à Bob en ajoutant une estampille. Le serveur envoie le message m1s à Bob. De même, Alice qui souhaite envoyer le message m3 à Christine, doit adresser m3 au serveur de causalité, qui après avoir ajouté une estampille, envoie le message m3s à Christine.

Christine souhaite envoyer le message m4 à Estelle. Le message doit transiter par le serveur de causalité qui le renvoie à Estelle en ajoutant une estampille. Le serveur envoie le message m4s à Estelle. Celle ci prend connaissance du message mais décide de le passer sous silence. Elle nie l'avoir reçu, et envoie le message m2 à Bob demandant à bénéficier de certaines conditions avantageuses, hors de propos depuis l'annonce du message m4s. Le serveur de causalité transmet à Bob le message m2s.

Bob reçoit les messages m1s et m2s. Estelle affirme ne pas avoir reçu m4. Que peut-on opposer à la mauvaise foi de Estelle qui s'efforce de provoquer un deni de causalité ?
Le serveur de causalité peut affirmer, qu'en ce qui le concerne, les estampilles ont été apposées aux messages et qu'il les a ensuite envoyés aux destinataires respectifs. C'est un schéma de preuve faible. En tout état de cause, rien ne permet de conclure qu'Estelle a reçu ou non le message m4s.

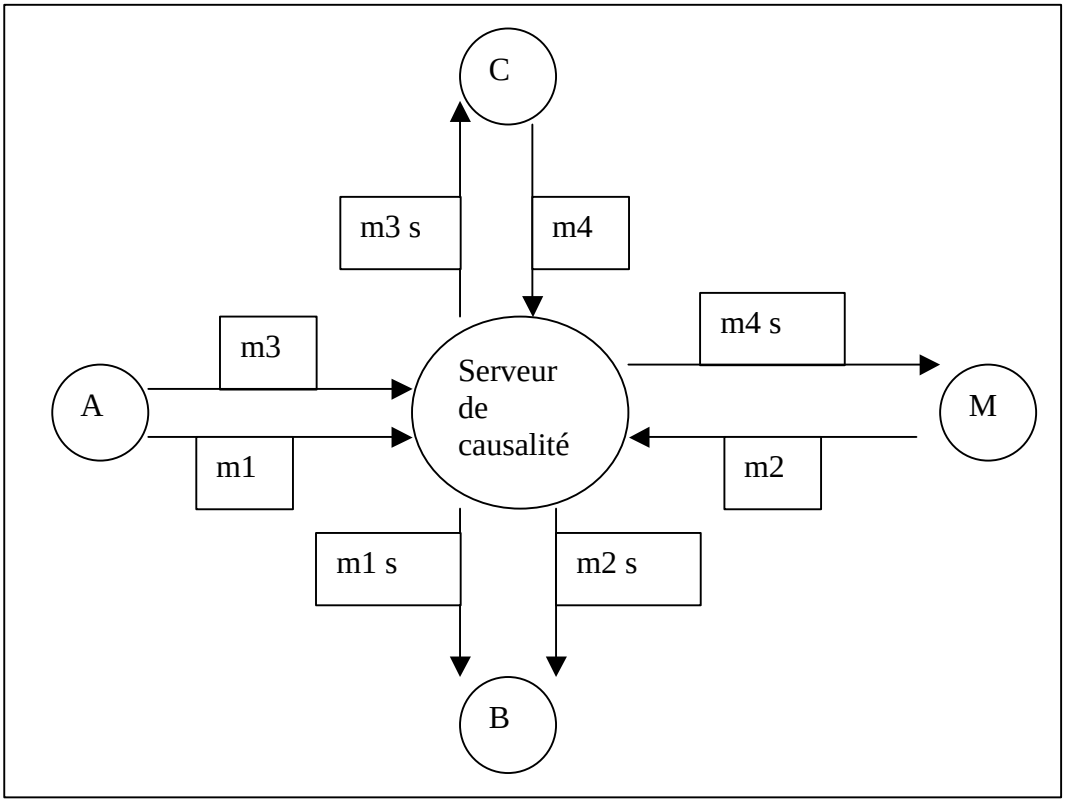

Figure 8 : Serveur de causalité

Le premier inconvénient de cette approche réside dans l'apparition d'un goulot d'étranglement. Mais de plus, la performance du système est moins bonne parce que le nombre de messages est multiplié par deux. Ainsi, les messages peuvent s'empiler si le serveur est incapable de les gérer plus vite qu'ils ne se présentent.
Le deuxième inconvénient de cette approche réside dans l'apparition d'un maillon faible. Si le serveur de causalité est investi par un attaquant, l'ordre causal n'est plus sécurisé du tout.
Le troisième inconvénient est l'existence d'un schéma de preuve faible, car en ce qui le concerne le serveur de causalité peut affirmer avoir envoyé les messages, les entités peuvent affirmer ne pas les avoir reçus.

Nous allons envisager un deuxième type d'algorithme pour garantir la sécurisation de l'ordre causal en empêchant sa destruction, est fondé sur l'approche conservatrice.

1.7.2.2 Prophylaxie par approche conservatrice.

Pour éviter la destruction de la relation d'ordre causal, il est tout à fait envisageable de recourir à une solution fondée sur l'ordre total qui garantit que les événements sont diffusés conformément à leur ordre d'émission. Il faut songer à surcharger le protocole avec un mécanisme d'acquittement faisant partie d'un protocole sous-jacent.
Dans notre exemple de courtiers en bourse, chaque entité du système attend d'avoir reçu l'acquittement du message précédent avant d'envoyer un nouveau message si bien que nécessairement, Bob reçoit m1 avant de recevoir m2, mais ce n'est pas l'objectif principal de cette approche. Le grand intérêt réside dans le fait que Maurice ne peut pas nier avoir reçu m4 puisque d'une part, Alice détient une preuve que Christine a reçu le message m3, et que d'autre part, Christine a la preuve que Maurice a reçu m4 puisque celui-ci lui a envoyé un acquittement.

Par conséquent, le fait d'imposer l'ordre local semble suffisamment prophylactique pour que Maurice ne puisse pas s'enrichir frauduleusement. Même si Maurice n'envoie pas l'acquittement, il ne peut pas détruire la relation de causalité.

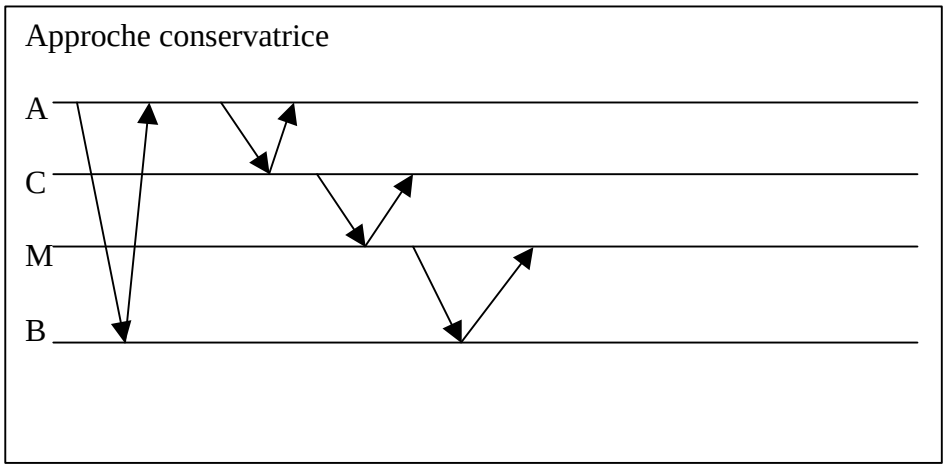

Figure 9 : Approche conservatrice

L'un des inconvénients de cette approche est la lenteur induite par l'attente obligatoire qui existe entre deux envois de messages, conditionnés par la réception d'un message d'acquittement. De plus il faut ajouter des temporisateurs pour exclure l'interlocuteur qui mettrait trop de temps à répondre.

Par conséquent, la destruction de la causalité s'avère plus difficile si le système réparti est construit en intégrant les notions de serveur de confiance et d'approche conservatrice. Mais, nous avons montré que ces deux algorithmes ne sont pas exempts de défauts et qu'ils ne permettent pas de manière optimale une sécurisation de la causalité. La sécurité mise en place ne permet pas de se prémunir contre la destruction de cette causalité.

Considérons à présent le cas où l'opposant ne cherche pas à détruire la causalité existante, résultante et vraie ; mais au contraire, l'attaquant s'efforce de construire une causalité fausse.

1.7.3 Fabrication d'une causalité

Définition : Une relation causale est fabriquée conformément au prédicat C s'il existe des messages m1 et m2 tels que m1-/->m2 mais pour un certain processus, C(m1,m2) est vrai.

Dans notre exemple, la causalité telle qu'elle existe n'est pas satisfaisante pour Estelle. La causalité actuelle va en l'encontre de son désir de s'enrichir frauduleusement, c'est la raison pour laquelle elle cherche à prouver que par causalité, le message m2 précède le message m1, ce qui est totalement faux.

Ainsi, selon les allégations d'Estelle m1-/->m2, mais C(m1,m2) est vrai

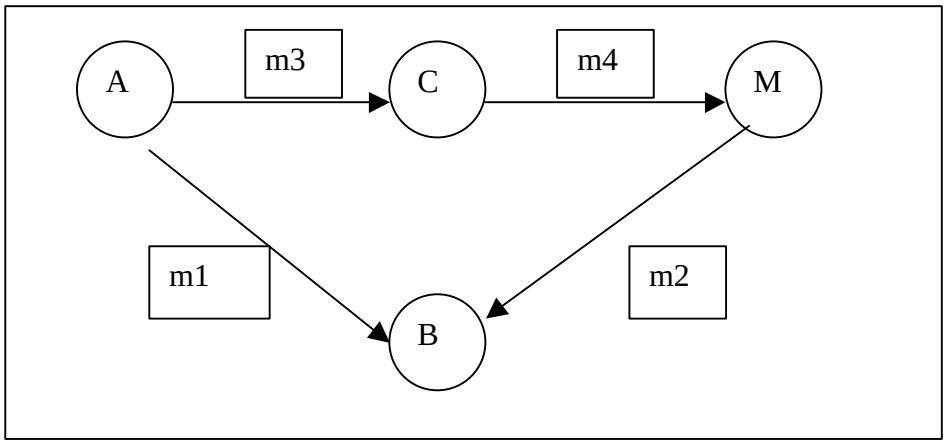

Figure 10 : Fabrication de la causalité

Estelle fabrique une relation causale en affirmant que m2 précède m1. Si on considère les allégations d'Estelle, m2 précèderait m1. En ce sens, les messages ne sont pas totalement ordonnés, mais Estelle fait croire que la vraie causalité temporelle est celle qu'elle a fabriquée et qui correspond à ses objectifs.
Pour empêcher la fabrication de causalité fausse (forgery), intéressons nous à deux à algorithmes susceptibles de garantir que la sécurisation de la causalité est effective en présence d'une tentative de fabrication de causalité. Ces deux algorithmes ont un point commun, ils utilisent un protocole à clé publique pour assurer l'authenticité des messages. Si Alice envoie un message à Bob, elle signe le message avec la clé privée de Alice (KA). Ainsi, Bob pourra vérifier que le message a été signé par Alice en déchiffrant la signature au moyen de la clef publique de Alice (ka). De plus amples détails concernant les protocoles de chiffrement asymétriques sont disponibles au paragraphe [1.5.3].
Le premier des deux algorithmes est fondé sur l'utilisation des horloges logiques vectorielles.

1.7.3.1 Prophylaxie par horloges logiques.

La terminologie d'horloge logique correspond à des techniques de datation des événements.
En 1978, Lamport proposa une solution d'horloge logique scalaire pour laquelle un seul compteur est associé à chaque événement du système. Le compteur est transmis sur le réseau en même temps que chacun des messages qui provoque son incrémentation.
L'algorithme de datation par **horloge logique scalaire** est le suivant :

1.) Pour chaque site I, chaque processus gère une liste de hi qui est initialisé à zéro.
2.) Pour chaque événement local sur le site I
   - Hi := Hi + 1
   - E est daté par Hi

   Chaque message est estampillé par la date de son émission.
   Estampille = (m, E(m)), avec E(m) = H(émission(m))

   A la réception d'un message (m, E(m)) sur le site I
   - Hi := max (Hi, E(m)) + 1)

Mais, cette solution d'horloge logique scalaire ne convient pas à la datation de messages indépendants car issus de processus concurrents sur des sites distants. Ainsi est apparue une solution évolutive, l'horloge logique vectorielle. Chaque processus Pi maintient une horloge logique vectorielle ti = (t1, t2, ...,tn) avec n = nombre total de processus
L'algorithme de datation par **horloge logique vectorielle** est le suivant :

1. Quand un processus commence à s'exécuter, tous les ti sont mis à zéro.
2. Le processus P incrémente ti avant d'exécuter un événement
3. Le processus P envoie l'horloge logique vectorielle en même temps que le message
4. Lors d'une réception de message, une comparaison est effectuée, entre la valeur reçue et la valeur locale. Pj met à jour son horloge logique vectorielle. Pour chaque tj, la valeur maximum est choisie.

A chaque fois qu'Alice génère un événement, elle incrémente le compteur qui lui est dédié parmi tous les compteurs de l'horloge logique vectorielle qu'elle détient. Puis elle ajoute une représentation de son horloge logique vectorielle dans le message qu'elle émet à l'attention de Bob. Lors de la livraison du message, Bob, qui gère lui aussi une horloge logique vectorielle, compare l'horloge d'Alice qu'il vient de recevoir avec le vecteur d'horloge logique vectorielle de Bob qu'il détient. Il compare d'une part la valeur de chaque compteur de chaque processus référencé par Alice avec d'autre part la valeur de chacun des compteurs qu'il détient. Si une tentative frauduleuse de fabrication de causalité a eu lieu quelque part dans le système, il est possible qu'il ne s'en rende pas compte, mais à un moment où à un autre, un processus s'en rendra compte.
Cette technique est très efficace et permet de concevoir un système réparti dans lequel la dépendance causale est gérée de manière fiable. Malheureusement, ce type d'approche nécessite autant de compteurs que d'entités dans le système. Si le nombre d'intervenants est grand, le nombre d'horloges est grand également et devient par conséquent lourd à gérer. [REITER 99].
Le type d'algorithme fondé sur l'utilisation des horloges logiques semble donc mal adapté à la gestion dynamique et aux communications qui comportent de nombreux participants. Par conséquent, il est préférable de considérer un autre type d'algorithme dans le cadre de ce mémoire.

1.7.3.2 Prophylaxie par piggybacking.

Envisageons un algorithme fondé sur le piggybacking (à dos de cochon) pour éviter qu'une entité du système ne fabrique une causalité qui viendrait à l'encontre de la relation causale. Dans cette approche, quand un processus émet un message, il ajoute au message un historique de tous les autres messages qui ont précédé et suscité l'envoi du message actuel. En définitive, chaque message est garant de l'historique des messages précédents.

L'algorithme de piggybacking présenté et prouvé dans [REITER99] est le suivant :

1.) Chaque processus gère une liste de hi qui est initialement vide.
2.) Si Pi exécute l'instruction envoi(m), Hm = hi est envoyé avec m
3.) Si Pj exécute l'instruction recevoir(m), hj est mis à jour. hj = hj $\cup$ Hm $\cup$ m

Pour détecter la relation causale, les processus appliquent le prédicat de causalité :
C(m1,m2) = {vrai si m1 $\in$ Hm2, faux sinon }
Dans ce prédicat, m1 $\in$ Hm2 veut dire qu'un message au contenu identique à m1 est inclus dans l'historique du message m2.
Intuitivement, l'algorithme de piggybacking est simple, dans l'illustration suivante, la lettre h signifie historique. Dans cette représentation, Alice envoie d'abord le message m1 à Bob. Ensuite, Alice envoie le message m0 à Christine en ajoutant m1 dans l'historique des messages. Puis, Christine envoie le message m2 à Bob en ajoutant m0 et m1 dans l'historique des messages. Bob reçoit donc deux messages différents en provenance d'émetteurs différents. Bob ne remarque la présence de m0 que dans le message que lui a envoyé Christine. Bob en conclut que le message m1 précède le message m0.

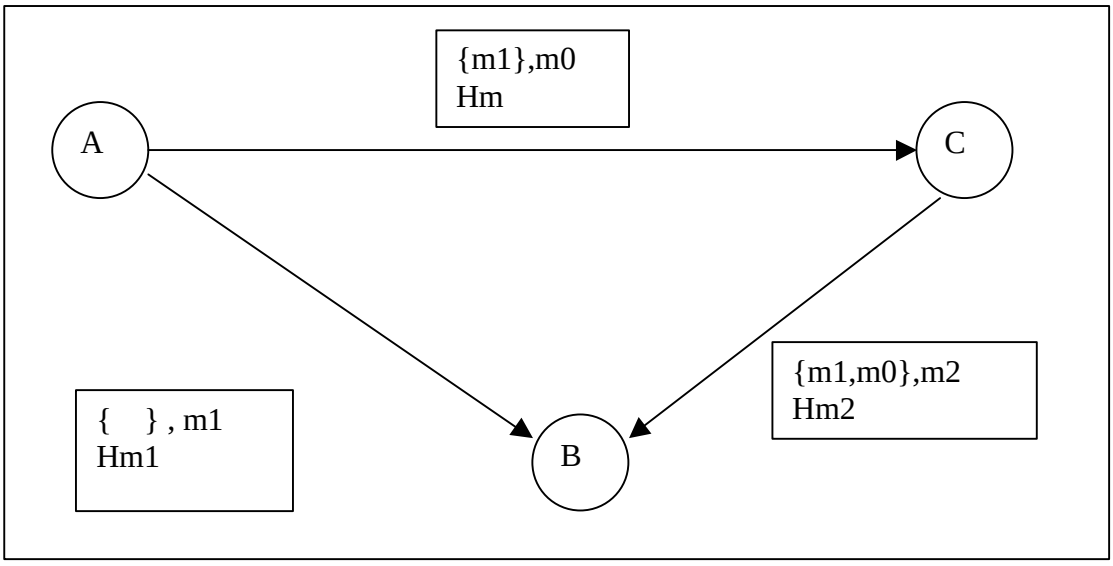

Figure 11 : L'algorithme de piggybacking

La technique de piggybacking est coûteuse en ressource, mais elle est très intéressante dans la mesure où la sécurisation de l'ordre causal devient tout à fait envisageable.
Intéressons nous maintenant plus précisément au temps et à l'ordre causal dans le but de concevoir un protocole de sécurisation de l'ordre causal qui corresponde à la problématique exprimée au paragraphe [1.6].

1.8 Le temps

1.8.1 Introduction

Depuis la nuit des temps, celui-ci – insaisissable – fuit à chaque seconde, inexorablement. Certes, le poète Lamartine affirma : « Oh, temps suspends ton vol ! » Mais le temps continue de s'écouler. Même si les horloges suisses, les mesures d'Einstein ou encore les efforts successifs de normalisation constituent des preuves de la volonté des êtres vivants qui essaient coûte que coûte de superviser, de contrôler ou de suspendre le temps. Ils ne peuvent jamais réellement le fuir, puisqu'il englobe tous les événements. Dans ce paragraphe, nous considérons deux artifices éprouvés permettant d'envisager une datation des événements : l'heure physique et la relation d'ordre causal.

1.8.2 L'heure physique

Dans un système réparti, obtenir exactement la même heure sur toutes les machines signifie obtenir un état global. En choisissant un point de vue purement mathématique, ce problème est impossible à résoudre. En choisissant un point de vue technique, pour atteindre et conserver un état global, les coûts induits en réservant les ressources systèmes et réseaux deviennent de plus en plus élevés. L'état obtenu s'approche d'un état que l'on pourrait qualifier de global, nonobstant certains détails et incertitudes désignés comme négligeables.

Mais il est temps de se poser la question fondamentale suivante : " Qu'est ce que l'heure et le temps ? "Les informaticiens répondent à cette question en précisant à quoi il sert et comment il est mesuré. Il sert à ordonnancer et cadencer les instructions à l'intérieur du microprocesseur. Son unité de mesure est la seconde dont la durée correspond à la transition entre deux niveaux de l'atome de césium 133.

D'un point de vue technique, il semble que la solution la plus judicieuse et la moins coûteuse consiste à contrôler précisément l'heure de chaque machine par rapport à une heure de référence. Mais, quel que soit le protocole choisi dans le but de maintenir une heure précise sur toutes les machines, il existe des moyens et des pistes pour modifier l'information recherchée avec la plus extrême des rigueurs. Prenons un exemple simple. Une horloge de confiance (mais quelle confiance ? et confiance en quoi ?) à l'écoute du bip de Radio France Internationale aurait un coût mesurable et raisonnable. L'ordinateur PC ainsi configuré coûterait plus cher qu'actuellement. Mais rien n'empêche d'imaginer une interface radio malicieuse qui ferait dériver le temps fourni en retard voire en avance. Ainsi, par cet exemple, nous réfutons la confiance et la crédulité. N'importe quel autre exemple servirait notre propos. La confiance existe jusqu'à l'instant de la trahison. Ralentir le temps est une approche très dangereuse pour le système et très prometteuse pour l'opposant qui ouvre ainsi une brèche dans la sécurité du système.

Puisque la notion d'heure physique choisie pour caractériser le temps ne suffit pas pour garantir la sécurité du système réparti, il faut absolument caractériser le temps en choisissant une autre approche différente de la notion du temps perçue habituellement. La datation des événements logiquement les uns par rapport aux autres est une approche encourageante.

1.8.3 La relation d'ordre causal

La relation d'ordre causal s'appuie sur la discipline mathématique de la topologie qui définit des relations et plus particulièrement des relations d'ordre partiel et total. Cette approche consiste à dater les événements du système indépendamment du temps physique. Les événements sont horodatés au moyen et à cause des événements qui les ont précédés dans le système et dans le temps passé. Peu importe l'heure physique, la cause produit un événement puisque n'importe quel fait a une cause et que les mêmes faits dans les mêmes conditions produisent les mêmes effets.

Sur chaque événement qui survient dans le système, une estampille distinctive peut prouver son authenticité. L'horodatage est réalisé en s'appuyant sur une relation d'ordre causale qui existe entre les événements eux-mêmes. Les propriétés d'ordre sont au cœur de la problématique de la sécurité et du temps dans les systèmes répartis.

Définition de Lamport : Soit N systèmes informatiques communiquant par messages, on considère que chaque site exécute un calcul séquentiel. On appelle événement, la fin d'une opération de calcul et en particulier une émission de message ou une réception de message.
Les relations seront des ordres partiels sur les événements.
- a et b sont exécutés sur un même site et a précède b dans le calcul.
- a est une émission d'un message m d'un site i vers un site j et b est la réception du même message.

On appelle ordre causal l'ordre partiel sur les événements qui est la fermeture transitive des deux relations précédentes.

Deux événements différents du système peuvent être reliés par une relation de précédence. E1 est intervenu avant E2. Cet événement E2 est intervenu après E1. De plus, si E1 fait partie des causes possibles de l'événement E2, alors par nécessité, E2 est apparu après E1. Dans ces conditions, le temps logique est exprimé au moyen d'une donnée du système prenant des valeurs différentes. Cette variable système est mise à jour au moyen d'une fonction dans le cadre d'un protocole qui précise la suite d'opérations exécutées par les différents protagonistes du système. Une typologie des diffusions ordonnées et quelques définitions sont nécessaires pour déterminer plus précisément la notion d'ordre causal.

1.8.3.1 Livraison ordonnée

La relation de livraison ordonnée existe entre deux messages M et M', lorsque ces deux messages sont livrés dans cet ordre aux destinataires [CARREZ96].

$M \rightarrow_{do} M' \Leftrightarrow$ M est livré avant M' sur chaque entité destinataire de M et M'

Note : l'instant de la livraison est différent et précède celui de la réception.

1.8.3.2 L'ordre local

L'ordre local correspond à l'ordre qui existe entre les événements sur le processus d'origine des événements. Ainsi, la relation d'ordre locale (notée $\rightarrow_l$)existe quand une entité A émet un message M avant d'émettre un autre message M'. Quand l'ordre local est assuré, les messages sont délivrés de manière ordonnée conformément à la relation locale qui a eu lieu sur le site d'émission initial. C'est une extension de l'ordre FIFO (premier entré, premier sorti). Le canal de communication est conçu de telle sorte que si un message M arrive avant un message M', alors nécessairement, l'émission de M a précédé l'émission du message M'.

$M \rightarrow_l M' \Rightarrow M \rightarrow_{do} M'$

L'ordre local qui existe entre deux événements est nécessaire mais non suffisant pour mettre en évidence leur ordre causal. L'ordre causal est plus strict que l'ordre local.

1.8.3.3 La diffusion locale

La diffusion locale est la plus simple à réaliser, elle ordonne les messages conformément à l'ordre local d'émission.

$\forall$ M, M' , M $\rightarrow_l$ M' $\Rightarrow$ M $\rightarrow_{do}$ M'

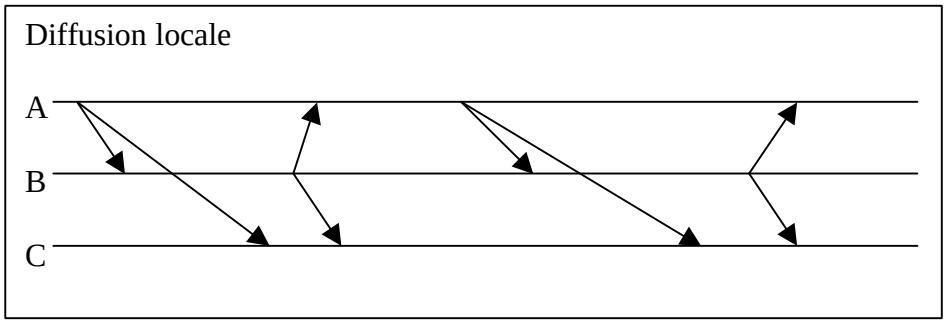

Figure 12 : Diffusion locale

1.8.3.4 L'ordre total

L'ordre total qui existe entre les événements d'un système réparti s'appuie sur une relation d'ordre total qui peut exister entre les événements du système.

Définition 1 : Soit A(M) l'ensemble des sites adressés pour un message M
Soit M et M' deux messages distincts du système.
Soit D la fonction de livraison des messages.
Soit i la variable de désignation des différents sites.

$\forall i, \in$ A(M) $\cap$ A(M'), on a Di(M) $\rightarrow$ Di(M') ou Di(M') $\rightarrow$ Di(M)

Définition 2 : Soit Rg la relation de délivrance ordonnée.
M et M' deux messages distincts du système
On note $\rightarrow_{do}$ la relation de délivrance ordonnée.

on a M Rg M' $\Leftrightarrow$ M $\rightarrow_{do}$ M' ou M' $\rightarrow_{do}$ M

De manière informelle, quel que soit le site concerné par la communication par diffusion de deux messages distincts, la relation totalement ordonnée garantit que si le message M est délivré avant l'autre message M' sur un des sites, alors M est délivré avant M' sur tous les sites du système.
De même si M est délivré après M' sur un des sites, alors la relation d'ordre total garantit que M est délivré après M' sur tous les sites.
Note importante : l'ordre d'émission initial de M et de M' n'est pas utilisé pour définir la relation de communication totalement ordonnée.

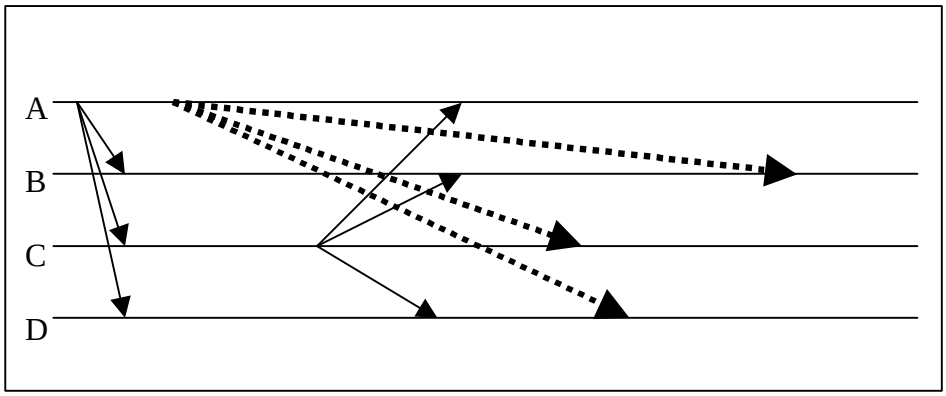

Figure 13 : Diffusion totale

1.8.3.5 La relation causale directe

La relation causale directe est une relation de précédence entre les messages.

$M \rightarrow_{cd} M' \Leftrightarrow$ M est livré à l'entité émettrice de M' avant que celle-ci n'émette M'

1.8.3.6 La diffusion causale

Dans l'illustration ci-dessous, la diffusion causale est respectée parce que la fermeture transitive des relations locales et causales directes est satisfaite.

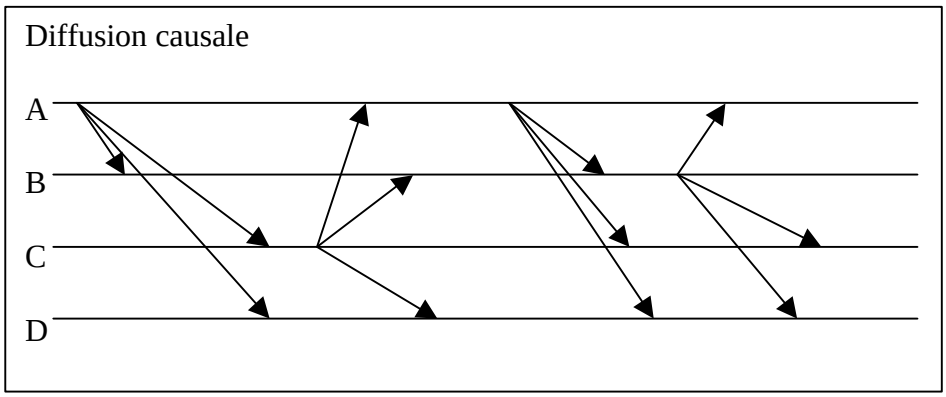

Figure 14 : Diffusion causale

Dans un chronographe, un observateur PEUT obtenir un ordre local et un ordre causal transitif direct et un ordre identifié mais ne pas obtenir d'ordre causal. Cet effet de bord apparaît toujours quand il n'y a pas d'entité qui est chargée de mémoriser tous les messages qui ont lieu lors de la relation causale transitive.

C'est la raison pour laquelle, les protocoles qui implantent des solutions fondées sur l'ordre causal, doivent englober des notions d'archivage et d'adressage à soi-même.

Théorème : Un ordre total et local avec une entité d'archivage et un mécanisme d'adressage à soi-même fournit un ordre total et causal [CARREZ98].

1.8.3.7 Perspectives

Il est donc tout à fait envisageable de dater les événements d'envoi de message à partir de l'ordre causal à condition de prendre en compte certaines imperfections. L'identification par énumération, l'archivage et l'adressage à soi-même sont des artifices éprouvés permettant de rendre causale une diffusion locale et totale.

Dans [CARREZ96], des résultats sont définis de manière à rendre possibles des protocoles à diffusion causale, deux résultats sont plus particulièrement exploitables.

1.) Dans un protocole avec adressage à soi-même, la station chargée de l'émission du message adresse le message aux stations nécessaires ainsi qu'à elle-même. La station émettrice participe au ré-ordonnancement des messages qu'elle émet, même si elle ne les délivre pas. Par cet approche, la diffusion locale et identifiable avec adressage à soi-même procure une diffusion causale et totale.

2.) Dans un protocole avec archivage et adressage à soi-même, une station particulière, archive reçoit tous les messages même si aucune entité d'application n'est destinataire de cette station archive. La station archive participe à l'ordonnancement de tous les messages reçus. De plus une station émettrice fait un adressage à soi-même. Nous pouvons affirmer que la diffusion locale et totale avec archive et adressage à soi-même fournit une diffusion causale et totale.

Par conséquent, la datation des événements d'envoi et de réception de message semble une approche plus convaincante que la datation par rapport à une référence commune à tous les processus. Mais cette approche de datation des événements non pas au moyen d'une base de temps universelle mais en utilisant la relation d'ordre causale qui existe entre les événements, n'est pas exempte d'attaques de la part d'un opposant mal intentionné à l'égard du système. C'est la raison pour laquelle, l'ordre causal fait l'objet de protections et d'une sécurisation appropriée.

En matière de sécurité, les propriétés liées à l'authentification et à l'intégrité des données sont très souvent explorées lors de la mise en place d'une politique de sécurité. La notion d'intégrité de la personne humaine telle qu'elle est exprimée en droit des affaires nous amène naturellement à envisager la sécurité sous l'angle de la propriété de non répudiation.

L'envoi de courriers électroniques signés non répudiables constitue l'un des services les plus prometteurs et des plus demandés par le marché.

La problématique de la sécurité et du temps dans les systèmes répartis est donc posée pour concevoir et réaliser le protocole SETSAR.

2 SETSAR

2.1 Introduction

SETSAR est l'acronyme de sécurité et temps dans les systèmes répartis.
Dans le cadre de travail de ce mémoire, il s'agit de fabriquer un protocole de communication de la signature de la relation d'ordre par propagation successive de la signature. Ce protocole est ajouté en plus d'un protocole de communication existant. On signe n'importe quel contenu, donc, on peut prouver à posteriori que ce contenu n'a pas été altéré. Par rapport à la relation d'ordre qui existe entre les événements du protocole existant, le protocole de signature permet de signer n'importe quelle relation d'ordre pour prouver à posteriori qu'elle a été celle-là. La preuve de sécurité est un plus par rapport aux propriétés du protocole de communication qui existe par ailleurs.

La preuve de sécurité est assurée par la mise en place d'un mécanisme de marquage au moyen de fonctions de hachage, ce qui permet de créer un résumé qui mémorise toutes les signatures antérieures. La sécurité ne fait pas marcher le protocole de communication qui existe par ailleurs. Il empêche que quelqu'un arrête de faire marcher le protocole quelle que soit la propriété assurée par le protocole de communication.
De plus, si une des entités refuse de coopérer en ne signant pas, ou en signant mal, un mécanisme exclut cette entité du système.

Différencier les messages qui possèdent le même contenu apporte de la complexité supplémentaire. Le message M s'appelle M tout au long de son cheminement à travers le système réparti. Le couple (Pi, M) suffit pour assurer la différence entre les messages. Dans le cas de la communication par diffusion totale, il est nécessaire de différencier les messages de contenu identique mais présents dans différents endroits du système, dans le cas de la diffusion sélective, il n'est pas nécessaire de le faire.

2.2 Enoncé des besoins

Deux entités ou deux groupes d'entités doivent communiquer, nonobstant leur comportement qualifié de byzantin. En effet, ils peuvent changer d'avis sans prévenir, voire trahir inopinément la confiance qu'ils inspirent. Si bien que le protocole de communication doit procurer l'existence d'une preuve faible et d'une preuve forte.

Le protocole doit s'appuyer sur la synergie de deux couches protocolaires qui sont d'une part les services d'enregistrement et d'autre part, la signature de la relation causale.

L'entité signataire doit être en mesure de prouver – en ce qui la concerne – qu'elle a bien agi. Ce qui est une preuve faible.

La preuve forte quant à elle doit être procurée par la signature causale. En chaînant les résumés successifs de la relation, les entités du protocole sont liées entre elles par une obligation réciproque et mutuelle, ainsi un contrat synallagmatique existe entre les entités. Si quelqu'un agit au détriment du système, il ne peut pas prouver avoir bien agi.

De plus, si une entité refuse d'utiliser le protocole, elle se met dans une situation inacceptable. Alors, un mécanisme doit l'exclure du protocole.

Chaque entité doit disposer d'un service lui permettant de vérifier la conformité des éléments qu'il vient d'obtenir lors du déroulement du protocole de communication.

En cas de litige, Alice et Bob, prennent les preuves, les fichiers, les données persistantes en leur possession et les confient à un acteur impartial qui assure l'arbitrage du litige.

Un acteur impartial que nous appellerons le juge, doit disposer d'une procédure fiable pour déterminer laquelle des deux entités a causé un préjudice à l'autre entité.

Le juge doit être capable d'évaluer l'intégrité de la relation d'ordre causale.

Le juge doit être capable de disposer d'un mécanisme pour traiter de la non-répudiation par origine et de la non-répudiation par destination.

Le juge doit être capable de déterminer avec exactitude l'ordre d'apparition des messages.

2.3 Spécifications fonctionnelles

2.3.1 Objectif

Dans une démarche de spécification orientée objet, il convient d'exprimer le plus précisément possible les besoins de l'utilisateur et de modéliser le système d'information. Les acteurs principaux du système veulent communiquer. Un accord d'utilisation doit exister entre les tiers. Un contrat synallagmatique existe entre les parties, ce contrat ne peut être répudié ni par l'un, ni par l'autre des partis. L'objectif principal que doit atteindre ce protocole est englobé dans une fonctionnalité que le système doit fournir aux utilisateurs par rapport à la propriété de la sécurité qui concerne la non-répudiation. On se place dans un protocole de communication fondé sur la confiance étayée par l'existence d'un schéma de preuve faible et d'un schéma de preuve forte.

2.3.2 Politique de sécurité

Dès la phase de spécification du protocole, l'analyse doit partager la terminologie de la sécurité au lieu de considérer une politique de sécurité qui s'adapte à un produit existant.
Il s'agit de répondre aux questions suivantes, sachant que cette liste n'est pas exhaustive :

- Qui peut attaquer ? Les entités du système réparti et le reste du monde.
- Quels sont les actifs que nous protégeons ? Les messages constituent des actifs.
- Quels actifs additionnels procurons nous ? Le résumé qui résulte des messages successifs.
- Quels risques potentiels encourent ces actifs ? La destruction ou la modification représentent des menaces significatives et intolérables.
- Quelles menaces spécifiques pèsent sur cette architecture ? En tout premier lieu, une des entités du protocole peut attaquer le système de l'intérieur en répudiant un message que ce soit par origine ou par destination ; en second lieu, un opposant peut intercepter un message et tenter de s'immiscer dans la conversation.
- Quels contrôles permettent d'annihiler ces risques et ces menaces ? Le nonce et l'enregistrement du nonce dans une structure de données dédiée semble satisfaisant.
- Quel est le niveau de risque résiduel ?
- Quand va se produire l'attaque ?
  Avant ou pendant l'envoi du message caractérise la préméditation ;
  Après caractérise la destruction (deni) de causalité.

- Combien de temps les données devront-elles persister en mémoire ? Toute la durée de la conversation est une durée minimale de la persistance des données.
- Combien de temps le contrat d'obligations mutuelles devra-t-il persister ? La durée maximale de la persistance des données dépend du type de contrat souscrit et de la volonté des contractants.
- Comment va procéder l'attaquant ? Le type d'attaque choisi par l'opposant qui ne doit en aucun cas être sous-estimé, dépend des choix d'implantation technique.

2.3.3 Environnement technique

L'environnement désigne Internet, selon la terminologie en vigueur dans les RFC de l'IETF.
Le protocole s'ajoute par exemple plus spécifiquement aux échanges de messages par courriers électroniques.

2.3.4 La fonction principale

La fonction principale du protocole est d'établir une relation durable entre deux entités dont le comportement byzantin est avéré et prouvé au paragraphe [2.5.1]. Malgré ce lourd passif, ils doivent obtenir l'assurance de sceller leur contrat en toute quiétude. Cette relation doit être fondée sur la confiance entre les deux entités qui doivent respecter les spécifications du protocole et des modalités de fonctionnement des outils informatiques mis à leur disposition. Par le biais de la signature électronique, une relation d'ordre causale est établie entre les messages successifs que s'envoient les associés de ce contrat d'obligations mutuelles.

2.3.5 La fonction secondaire

La fonction secondaire du protocole est de fournir un mécanisme de validation et de vérification du message. Cette fonction de validation s'applique soit à un seul échange dans le cas d'une vérification unitaire, soit à un ensemble de messages dans le cas d'une vérification de la globalité des messages relatifs à un contrat désigné et enregistré. Il s'agit de contrôler que l'émetteur a bien envoyé ce qu'il prétend avoir envoyé, et que l'ordre de la relation locale et causale est conforme à l'ordre que les entités prétendent avoir construit.

2.3.6 La fonction d'enregistrement

Le protocole doit fournir une fonction dont l'objet est de procurer une propriété d'auditabilité.
Quand un acteur principal veut envoyer un message à un autre acteur du système, il prépare ce message, ensuite le système doit traiter ce message en assurant des fonctions d'archivage, d'estampillage et d'historique. Alors, l'utilisateur principal peut envoyer le message à son interlocuteur.
Quand un autre acteur principal obtient le message qui lui a été envoyé précédemment, il doit prendre connaissance des informations et les enregistrer. L'historique fait partie de ces informations qu'il convient d'enregistrer avec le plus grand soin parce que cet historique qui lui a été envoyé permet d'établir la relation de causalité.
Quand un litige intervient entre deux acteurs principaux du système qui ont enregistré soigneusement les données, une autorité de confiance doit trancher le litige et déterminer si l'ordre est conforme d'un point de vue local et d'un point de vue causal.
Le succès ou l'erreur d'une vérification ne doit pas dépendre de la volonté, de la présence ou de l'existence du signataire ou plus généralement de n'importe qui d'autre que le vérificateur. [GOSCINSKY91]

2.3.7 La fonction de non modification de l'ordre

Le protocole doit fournir une fonction dont l'objet est de se prémunir d'une éventuelle modification de l'ordre dans lequel les messages ont été envoyés ou sont rédigés, les uns par rapport aux autres. C'est la raison pour laquelle, le système doit procurer une fonction très importante, l'horodatage. Bien qu'il puisse sembler trivial d'utiliser l'heure universelle, ce n'est pas l'option qui a été choisie. Le système ne doit pas être exclusivement tributaire de

l'heure universelle. Mais cela ne veut pas dire qu'il n'existe pas d'horodatage. Cette fonction doit utiliser la relation de causalité qui existe entre les envois de messages successifs qui matérialisent la communication entre les entités du système. Les événements sont datés les uns par rapport aux autres.
Ainsi, une fonction d'horodatage doit procurer une estampille qui ne caractérise qu'un message et un seul. Toutefois un message peut faire référence à d'autres estampilles que la sienne, mais alors ces estampilles caractérisent d'autres messages. Par conséquent, le système doit établir une relation de type application mathématique bijective entre l'ensemble des messages d'une part et l'ensemble des estampilles d'autre part. Il faut absolument que l'estampille ne soit pas prédictible dans le but de se prémunir des attaques menées à l'encontre du dessein du système.

2.3.8 Les rôles

Les fonctions principales et secondaires du système étant identifiées sommairement, il convient de définir plus précisément les différents rôles que peuvent endosser les entités impliquées dans la relation de communication. Mais commencer par identifier les rôles des entités qui entrent en relation avec le système, permet aussi de définir la notion de composant et le comportement des objets.

Le rôle est un des trois axes d'un environnement à trois dimensions, les deux autres axes étant la notion et le comportement. La notion définit les attributs et les méthodes d'un objet du domaine de l'application. Le comportement représente une évolution réutilisable dans le domaine de l'application. Le rôle modélise différents états transitoires et les transitions correspondantes. C'est au sein d'un rôle que s'effectue la combinaison d'une notion et d'un comportement pour obtenir un modèle complet d'objet.
Le rôle anime la notion. Le rôle concrétise le comportement.
Les rôles identifiés sont les suivants :

L'auteur : il écrit un message pour l'envoyer, il rédige un message et demande au système de le préparer, l'auteur envoie le message signé.

Le destinataire : il est concerné par le message que l'auteur a rédigé à son attention et qui lui a été envoyé, le destinataire obtient le message signé et l'enregistre.

Le vérificateur : il intervient après réception et enregistrement d'un message, le vérificateur peut lancer un programme qui lui prouve que son interlocuteur a agi conformément avec les messages précédents. Ce rôle est secondaire, il existe, mais dans une relation fondée sur la confiance, il est facultatif.

Le juge : il lance un programme qui lui permet de lever les ambiguïtés et de déterminer l'ordre dans lequel les messages ont été communiqués entre les entités du système. Ce rôle est secondaire, le juge intervient en cas de litige entre les entités du système.

Le reste du monde : N'importe quelle entité qui ne correspond pas aux rôles d'auteur, de destinataire, de vérificateur ou de juge obtient automatiquement le rôle de reste du monde.

Les rôles définissent les acteurs du système qui doivent être liés entre eux par une convergence d'opinion au sujet des modalités d'utilisation qui sont explicitées dans les scénarios d'utilisation.

2.3.9 Les cas d'utilisation

Les cas d'utilisations correspondent à des « Use Cases » dans la terminologie du langage de modélisation unifié (UML). Les cas d'utilisation ont été formalisés par Ivar Jacobson. Ils décrivent, sous la forme d'actions et de réactions, le comportement d'un système du point de vue de l'utilisateur. Ils permettent de définir les limites du système et les relations entre le système et l'environnement.

Une même personne physique peut jouer le rôle de plusieurs acteurs. [MULLER00]

Dans notre protocole, un acteur peut tenir plusieurs rôles successivement, mais ce n'est pas obligatoire. Par exemple, Alice peut tenir le rôle d'auteur, mais peut aussi endosser celui de destinataire. Alice peut aussi si elle le souhaite agir en temps que vérificateur. De même, l'acteur Bob, peut tenir les rôles d'auteur, de destinataire ou de vérificateur. Un même acteur peut jouer plusieurs rôles à l'égard du système, mais tous les acteurs ne correspondent pas forcément à ce critère. L'auteur d'un message est aussi le récepteur d'un autre message. Le récepteur d'un message est aussi l'émetteur d'un autre message. Le récepteur d'un message est aussi un vérificateur potentiel du message qu'il vient de recevoir. L'auteur potentiel d'un message, avant de décider de devenir réellement l' auteur d'un message peut décider de tenir le rôle de vérificateur du message qu'il a reçu précédemment de la part de l'acteur avec qui il est lié par la relation de communication. Ainsi, dans la mesure où la vérification ne détecte aucune anomalie, l'auteur peut rédiger son message en toute quiétude par rapport à la relation de communication précédente.

Considérons maintenant le rôle du juge. Cet acteur ne doit pas être confondu avec les autres acteurs de l'application. Alice ou Bob ne peut pas endosser ce rôle. Le juge est unique et solitaire au sein de l'application puisqu'un adage du droit nous oblige à considérer que « Nul n'a le droit de faire à soi-même justice » [ROLAND99]. Certes, la jurisprudence admet l'inexécution du contrat synallagmatique à l'encontre du débiteur négligeant en cas de détection de dol ou de lésion. Mais, l'adage signifie que la reconnaissance ou la mise à exécution d'un droit passe nécessairement par la justice de l'État. Si Alice estime que le comportement de Bob est contraire aux engagements réciproques, elle est en droit de suspendre dans une certaine mesure l'exécution du contrat, mais en aucun cas, Alice n'est habilité pour engager une procédure privée, Alice doit porter l'affaire devant la juridiction compétente. Il serait présomptueux de croire que tous les problèmes peuvent être résolus au moyen de spécifications, d'algorithmes et de calculs. La complexité des textes de loi, la pondération des éléments de jurisprudence et la grande sagesse des adages nous incite à la modération, voire à la modestie. L'adage « Juge unique, juge inique » [ROLAND99] nous entraîne à envisager que l'unicité du juge peut être source d'iniquité pour plusieurs raisons : le risque de mal juger en commettant une erreur, la partialité envers une des parties et la dépendance causée par les sollicitations. Le juge n'étant pas exempt de la tentation de frauder, le mécanisme de jugement doit être impartial. Etant donné les faits qui lui sont rapportés, le juge résout le problème qui lui est soumis en scindant ce casse-tête en deux énigmes distinctes, pour lesquelles, si possible deux solutions existent et deux seulement. D'une part la réponse est oui, l'auteur a répudié son message, et d'autre part la réponse est non, l'auteur n'a pas répudié son message. Cette fonction de jugement correspond au principe de base de la sécurité répertorié sous le terme d'oracle. Le juge doit aussi être en mesure de déterminer, en examinant deux messages, lequel des deux messages a été traité le premier par le système et lequel des deux a été traité le deuxième par le système.

Dans la terminologie UML, un scénario est une instance d'un cas d'utilisation. Envisageons les cas d'utilisation qui correspondent à des scénarios qui reflètent un comportement idéal et intègre des utilisateurs.

2.3.9.1 Cas d'utilisation n° 1

Envoi sur setsar
Acteur : Auteur
Résumé :
Ce cas d'utilisation permet à un utilisateur de préparer un message en vue de l'envoyer à l'autre contractant de la relation synallagmatique. Des fonctions d'archivage sont implicites et transparentes.
Contexte d'utilisation :
L'auteur veut envoyer un message à son associé avec lequel il est en relation contractuelle.

Pré-condition :
Une structure de données de type h_state existe, c'est une structure de données évolutive qui subit et enregistre les modifications [KOPETZ97].
Description du déroulement :
Le système gère les objets dans le but de préparer les messages qui seront archivés et ceux qui seront envoyés.
Les interactions sont les suivantes :

- mise à disposition d'un message pour l'application ;
- lancement de l'application ;
- récupération des données en vue de les envoyer par Email.

Exception :
Le message n'est pas mis à disposition, le programme s'arrête.

Post-condition :
La structure de données a été modifiée.

2.3.9.2 Cas d'utilisation n° 2

Réception sur setsar
Acteur : Destinataire
Résumé :
Ce cas d'utilisation permet à un utilisateur d'obtenir et d'enregistrer les messages qui lui ont été envoyés.
Contexte d'utilisation :
Le destinataire veut enregistrer les messages qu'il a reçu de la part de son associé.
Pré-condition :
Une série de messages a été reçu.
Description du déroulement :
Le destinataire enregistre les messages reçus. La pile de nonce est mise à jour automatiquement.

Exception : aucune

Post-condition :
Les messages ont été enregistrés

2.3.9.3 Cas d'utilisation n° 3

Vérification sur setsar
Acteur : Vérificateur
Résumé :
Ce cas d'utilisation permet à un utilisateur de vérifier les messages qu'il vient d'obtenir.
Contexte d'utilisation :
Un utilisateur peut vérifier en utilisant les données en sa possession que l'autre entité a agi en conformité avec les messages précédents.
Pré-condition :
Une série de messages a été enregistré.
Description du déroulement :
Le système gère les objets dans le but de vérifier si les messages enregistrés sont liés par une relation d'ordre causale. Si aucune anomalie dans la relation n'est détectée, alors le contrat synallagmatique a été respecté par l'auteur qui lui a envoyé le message. Sinon, une anomalie est détectée, le vérificateur peut engager une procédure de litige.
Les interactions sont les suivantes :

- lancement de l'application
- récupération des données en vue de décider du litige éventuel.

Exception : L'utilisateur avait envoyé un message, mais il n'a toujours pas reçu de réponse. Si bien que, s'il lance le programme de vérification, ce programme s'arrête.
Post-condition : aucune

2.3.9.4 Cas d'utilisation n° 4

Jugement sur setsar
Acteur : Juge
Résumé :
Ce cas d'utilisation permet au juge d'arbitrer et de déterminer lequel parmi deux messages a été envoyé le premier. Ce cas d'utilisation permet aussi de trancher en décidant si oui ou non la relation de causalité entre les messages a été brisée.
Contexte d'utilisation :
Un litige existe entre deux acteurs du système. Ils ont apporté leurs données respectives au juge, celui-ci exploite les données pour déterminer si les acteurs ont bien agi respectivement. Si un des acteurs a mal agi, le juge doit pouvoir mettre en évidence que cet acteur ne peut pas prétendre avoir bien agi.
Pré-condition : une série de messages a été confiée au juge.
Description du déroulement :
Le système gère les objets dans le but de reconstruire peu à peu le déroulement de la relation de causalité telle qu'elle s'est produite entre les messages successifs qui ont été échangés entre acteurs du système. En fonction des données qui lui ont été confiées, le juge doit pouvoir déterminer si la relation de causalité entre les messages est établie conformément aux spécifications et si le contrat entre les auteurs successifs est encore valide ou si la relation de causalité entre les messages a été brisée. Une fois que la relation entre les messages a été reconstruite, le juge est en mesure de déterminer exactement lequel des deux messages a été envoyé le premier, et par conséquent lequel des deux acteurs a raison dans le litige qui l'oppose à l'autre associé du contrat synallagmatique.
Exception : Aucune.
Post-condition : Le message 1 a été envoyé avant le message 2 ou le message 2 a été envoyé avant le message 1

2.3.10 Diagramme d'activité de l'application

Le diagramme d'activité regroupe les différents rôles possibles et les cas d'utilisation correspondants. Il convient de ne pas confondre le rôle tenu par l'acteur et l'activité potentielle générée. Le rôle concrétise le comportement, ne le remplace pas.
Les rôles figurent en haut des colonnes, les activités possibles, quant à elles sont représentées à l'intérieur des colonnes. Un point noir indique un point d'entrée dans le protocole, un point noir entouré d'un rond blanc indique quant à lui une terminaison du protocole. Les losanges optionnels représentent des choix de comportement, si la condition est vraie, la branche de gauche est utilisée, sinon l'activité se dirige vers la droite du losange.
Note : la colonne Destinataire n'a pas de point noir, une réception ne débute pas l'application.

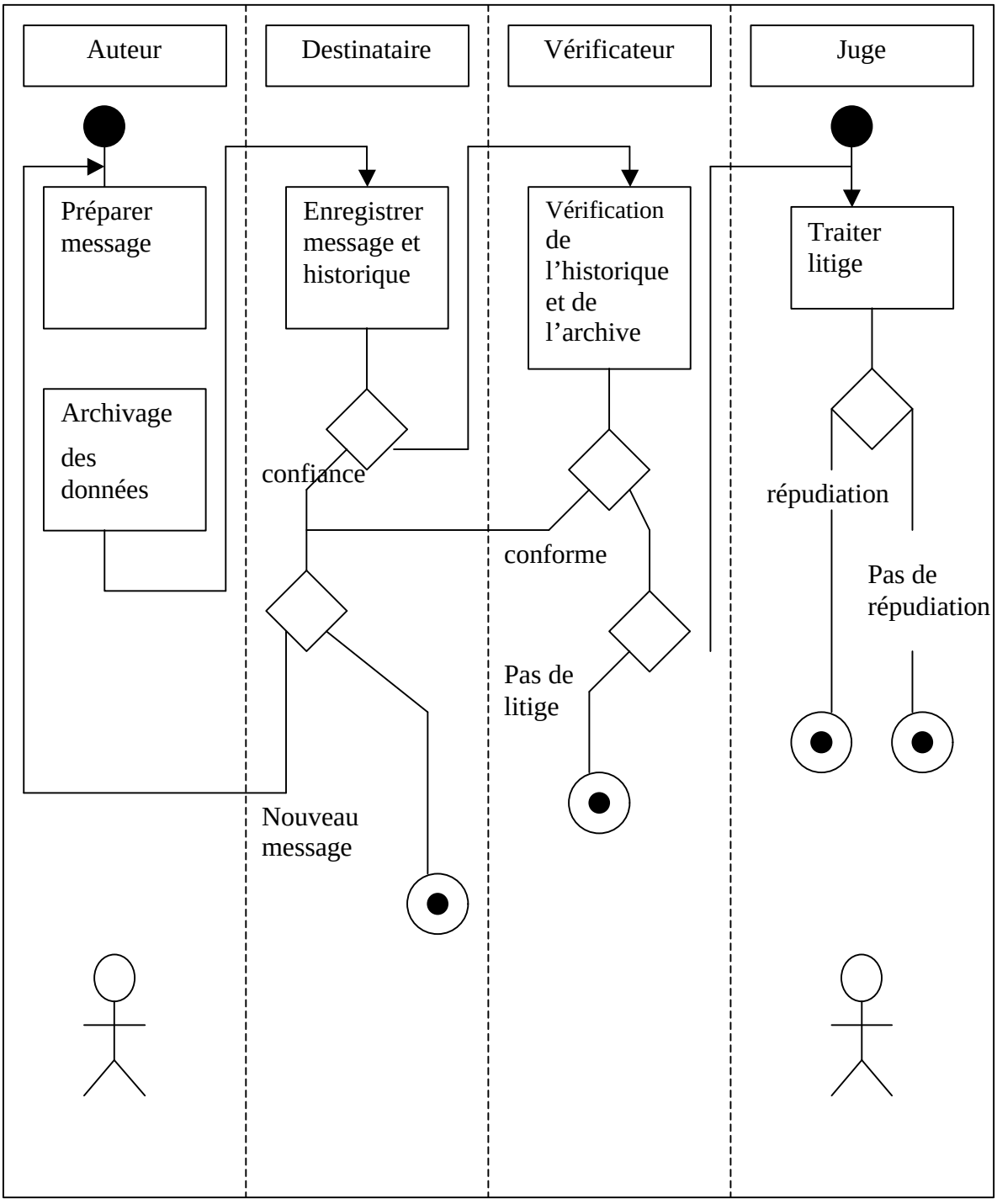

Figure 15 : Diagramme d'activité

Si le destinataire a confiance en l'auteur dont il vient de recevoir un message, alors le diagramme d'activité présente une branche qui aboutit sur un nouveau losange. Si le destinataire décide de préparer un nouveau message, alors le diagramme d'activité nous montre que le l'acteur va changer de rôle, le destinataire se métamorphose en auteur. Mais si le destinataire préfère ne pas envoyer de nouveau message, alors le diagramme d'activité présente une branche qui se termine sur un rond blanc. C'est la fin de l'application.

En ce qui concerne le rôle du juge, dans la mesure où il est sollicité par un des acteurs du protocole, il va entreprendre une activité en vue de traiter le litige. Le losange indique le litige que le juge doit trancher. L'auteur dont il est question a répudié sa signature, ou bien l'auteur dont il est question n'a pas répudié sa signature. Le juge doit fonder sa décision à partir des données qui lui ont été fournies par les acteurs du protocole.

2.3.11 Les interfaces

2.3.11.1 Introduction

Ce protocole gère des structures de données d'archivage.
Ce protocole gère des structures de données d'historique.

Ces deux structures de données doivent être distinctes.

2.3.11.2 Description

Les interfaces sont les suivantes :

- Archive : utilisée par l'auteur et le vérificateur, ainsi que pour le logiciel de mailing ;
- Courrier : située après le logiciel de mailing et utilisée par le destinataire ;
- Historique : utilisée par le destinataire et le vérificateur.

Les interfaces Archive et Historique sont aussi utilisées par le juge.
Chaque acteur principal du système possède à la fois une interface d'archive et une interface d'historique.

2.3.11.3 Conditions d'implantations

L'archivage et l'historique sont des structures de données de type h_state [KOPETZ97].
Ces deux interfaces subissent l'évolution des modifications provoquées par le calcul. Chaque itération du système, correspond à une modification de l'historique. Dans notre exemple de système qui ajoute un historique à des échanges de courrier électronique, chaque message envoyé contient un historique différent. Si on considère qu'il existe n messages dans l'historique et que l'historique vérifie la relation d'ordre causal à l'ordre n. Si on montre que le message existe aussi à l'ordre n+1 et qu'il est aussi conforme à la relation d'ordre causal à l'ordre n+1, alors il est possible d'appliquer un raisonnement par récurrence. Cependant, il ne faut surtout pas oublier de poser le cas de base de la récurrence. Quand n=0, il faut prévoir un historique pour fixer les bases du raisonnement. Sans vouloir brûler les étapes de l'analyse, il est nécessaire de se préoccuper dès l'étape de l'analyse fonctionnelle, de l'impérieuse nécessité de fixer les bases du raisonnement par récurrence. Par exemple, si l'instance du modèle fonctionnel choisi comporte des fichiers plats, alors un fichier nommé init, contenant le mot init, permet d'initialiser la fonction de récurrence. De plus, il faut enregistrer en mémoire stable les fichiers émis et il faut enregistrer en mémoire stable les fichiers reçus.

2.4 Analyse de faisabilité.

2.4.1 Introduction
La spécification fonctionnelle est terminée, il convient de se poser la question de la faisabilité du projet en considérant tous les aspects de l'analyse fonctionnelle. Les coûts induits par la conception et la réalisation d'une application conforme aux besoins du client peuvent en effet s'avérer difficile à concilier avec les délais de mise à disposition. Dans ce cas il vaut mieux étudier les solutions qui vont être proposées, en argumentant sur les coûts induits avant de proposer une solution adaptée. Celle-ci s'avère certes moins ambitieuse que la solution qui pourrait être dégagée directement des spécifications fonctionnelles, mais elle a le mérite de contenter tous les intervenants du projet.

2.4.2 Solution 1
La solution 1 comporte un protocole de mode message asynchrone en point à point (unicast).

2.4.3 Solution 2
La solution 2 utilise un protocole de messagerie au moyen d’outils comme Netscape, Pine, MAPI ou SMIME.

2.4.4 Solution 3
La solution 3 envisage un protocole à messages en diffusion générale (broadcast).

2.4.5 Solution 4
La solution 4 envisage un protocole à messages protocole en diffusion sélective (multicast).

2.4.6 Conclusion
Dans le cadre de ce mémoire et du développement de SETSAR, la solution 1 et une partie de la solution 2 ont été explorées. Les outils de messagerie Netscape et Pine ont été utilisés. Mais les protocoles de messagerie MAPI et SMIME n'ont pas été explorés. Ces points pourraient faire l'objet d'un travail ultérieur.
En ce qui concerne les solutions 3 et 4, les connaissances académiques exprimées dans le chapitre 1 (terminologie et limites imposées) n'ont pas été explorées dans le chapitre 2 (Setsar), si bien qu'une recherche ultérieure sur ces sujets est tout à fait envisageable

2.5 Conception préliminaire

2.5.1 Eléments de preuve du protocole

La preuve d'un programme est envisageable à plusieurs époques de son cycle de vie. Une telle démarche peut être effectuée soit à l'issue de la conception préliminaire, soit à l'issue de l'aboutissement de l'énoncé de l'architecture. De plus, si le raffinement est plus poussé, un choix de preuve peut s'avérer utile au niveau de la spécification détaillée du logiciel. Dans ce dernier cas, la preuve permet de générer des schémas de code voire des tests unitaires.

Le but ultime d'un protocole est de ne pas tomber dans un schéma de preuve qui aboutisse à un dilemme. Si le protocole est incapable de choisir entre deux solutions contradictoires qui présentent toutes les deux des avantages comparables ou bien des inconvénients équivalents, la limite de la preuve est atteinte. Il convient alors de reconsidérer la preuve, le protocole voire les deux à la fois. Mais surtout, il faut éviter de tomber dans le syndrome du doigt accusateur. Le responsable de la preuve montrant du doigt les imperfections du protocole, tandis que le responsable du protocole rejette la responsabilité des erreurs et des fautes sur le schéma de preuve.

Dans ce chapitre dédié à la conception préliminaire, plusieurs paragraphes traiteront de la preuve, nous considérons successivement deux algorithmes issus de la conception préliminaire. Nous choisissons donc délibérément de rester le plus indépendant possible à l'égard du langage de programmation qui sera utilisé pour construire de l'information interprétable par une machine.

Le raisonnement par réfutation sera privilégié tout au long des démonstrations de ce chapitre. Nous choisirons des cas d'utilisations dans lesquels les actifs du système ont été attaqués, et pour lesquels il semble que la sécurité du protocole ait été mise en défaut. Nous mettrons alors en évidence une absurdité ou exhiberons une preuve qui réfute la thèse avancée par l'opposant.

Le raisonnement par induction permet à partir de fait choisis de remonter jusqu'aux concepts. Le raisonnement par déduction permet à partir de faits avérés de prouver le bien fondé d'autres faits qui résultent de ces hypothèses.

Certes, le raisonnement par syllogisme est envisageable et permet de fonder une conclusion sur des propositions supposées vraies; toutefois, sommes nous sûrs de la vérité de ces propositions ? Il convient d'être prudent, c'est la raison pour laquelle, la preuve qui repose sur un raisonnement par syllogisme doit être maniée avec précaution et parcimonie. Prenons un exemple de syllogisme utile à notre propos et à la compréhension de notre protocole.

Soit à prouver formellement que si tous les utilisateurs de courrier mail ont un comportement byzantin, et que de plus si Bob est un utilisateur de courrier mail, alors indubitablement, Bob présente un comportement byzantin.

Une preuve formelle spécifiée $\Gamma \vdash \Delta$ est une suite d'applications de règles d'inférence commençant par $\Gamma \vdash \Delta$ et se terminant par des axiomes.
Γ est l'ensemble des règles de gauche permettant de raisonner sur des hypothèses.
Δ est l'ensemble des règles de droite permettant de raisonner sur la formule à prouver.

Preuve formelle :

U(-) Etre un utilisateur de mail
B(-) Etre byzantin
Bob constante

$\forall x : (U(x) \rightarrow B(x))$	Tous les utilisateurs de courrier mail ont un comportement byzantin
U(Bob)	Bob est un utilisateur de courrier mail
$\vdash$	Si alors
B(Bob)	Bob présente un comportement byzantin.

Figure 16 : Preuve du comportement byzantin de Bob

$\Gamma , \forall x : (U(x) \rightarrow B(x)) \wedge U(Bob) \vdash B(Bob) , \Delta \quad \forall \vdash$
$\Gamma , \forall x : (U(x) \rightarrow B(x)) \wedge (U(Bob) \rightarrow B(Bob)) \wedge U(Bob) \vdash B(Bob) , \Delta \quad \wedge \vdash$
$\Gamma , \forall x : (U(x) \rightarrow B(x)), (U(Bob) \rightarrow B(Bob)) , U(Bob) \vdash B(Bob) , \Delta \quad \rightarrow \vdash$

$\Gamma ,U(Bob) \vdash U(Bob) , B(Bob) , \Delta \quad$ aff $\vdash$ aff
$\Gamma ,U(Bob) \vdash U(Bob) , \Delta \quad$ axiome

$\Gamma , \forall x : (U(x) \rightarrow B(x)) \wedge U(Bob) \wedge B(Bob) \vdash B(Bob) , \Delta \quad \wedge \vdash$
$\Gamma , \forall x : (U(x) \rightarrow B(x)), U(Bob) , B(Bob) \vdash B(Bob) , \Delta \quad$ aff $\vdash$
$\Gamma , B(Bob) \vdash B(Bob) , \Delta \quad$ axiome

Certes, les axiomes ne sont pas décrits, mais nous devons les admettre si bien que nous avons fermé avec succès toutes les branches possibles de l'arbre de preuve. Par conséquent, nous estimons juste d'admettre que Bob, à l'instar de tous les utilisateurs de mail, présente un comportement byzantin.

Mais il serait présomptueux de croire et d'affirmer que ce type de preuve formelle par syllogisme permettrait de prouver indubitablement n'importe quel problème. L'ensemble des règles de gauche (gamma) qui permet de raisonner sur les hypothèses doit être délimité et exprimé de la manière la plus rigoureuse possible. Si des hypothèses temporelles sont envisagées, alors le problème posé devient différent, intéressons nous à un développement informel à ce sujet.

Que Bob soit un utilisateur de mail semble un fait acquis qui n'admet aucune contestation. Toutefois, Bob a pu résilier son abonnement auprès de son fournisseur habituel au cours de la nuit précédente, ou bien le fournisseur d'accès au Mail a subi une attaque de la part d'un pirate et ses serveurs sont indisponibles. Ainsi Bob n'est plus utilisateur de mail pour une période de temps non déterminée, nous nous heurtons ici, dès le départ à un problème technique provoqué par des contingences temporelles et économiques.

Considérons maintenant le début du syllogisme. Que tous les utilisateurs de courrier mail aient un comportement byzantin semble déjà plus contestable parce que cette phrase n'est pas un fait élémentaire, mais une juxtaposition de faits. La proposition qui en résulte semble intéressante à plus d'un titre. Le premier terme de la phrase possède lui même un bon potentiel de césure. On peut estimer, par exemple, que certains utilisateurs de mail seraient au dessus de tout soupçon et auraient un comportement exemplaire. Nonobstant la présomption d'une attitude correcte, le comportement byzantin serait alors causé par l'environnement technique soumis à des alea indépendants de la volonté de l'utilisateur mais qui viendraient à l'encontre de son comportement exemplaire.

Mais nous devons nous engager et déterminer un fait, une hypothèse de départ. Le syllogisme est prouvé et nous admettrons que Bob a un comportement byzantin, ce qui devient un fait sur lequel nous pouvons nous appuyer pour démontrer d'autres raisonnements.

La preuve statique examine les faits tels qu'ils se présentent alors que le programme est arrêté. Les pré-conditions et le déroulement du protocole ont généré des traces. La preuve Statique s'intéresse à une étape atomique de programme, puis à une suite d'étapes analogues uniques qui correspondent à la séquence suivante : pré-condition, action, post-condition. La preuve statique envisage les différents cas possibles pour répondre à la question cruciale de savoir si l'ensemble satisfait aux exigences du protocole.

La preuve dynamique s'intéresse aux relations d'ordre partielles générées par une exécution en environnement parallèle et distribué. La preuve dynamique est plus compliquée que la preuve statique. Le schéma de preuve dynamique considère plusieurs branches de l'arbre qui représente l'ensemble des résultats possibles de l'exécution du programme. Sans connaître précisément les valeurs des branches de l'arbre, quatre types de conclusions sont possibles, énonçons les :

1°) le fait est prouvé et réalisable pour toutes les branches de l'arbre;
2°) il est prouvé que le fait n'est pas réalisable pour toutes les branches de l'arbre;
3°) le fait est prouvé et réalisable pour certaines branches de l'arbre;
4°) le fait n'est pas prouvable.

La preuve dynamique considère les choix possibles binaires. Si une branche n'est pas exploitable, elle est considérée comme fausse et tous les choix qui en découlent sont déclarés faux. Par contre si une branche est exploitable, la preuve est applicable aux branches et feuilles de l'arbre qui dépendent de l'existence de cette ramification déclarée vraie.

2.5.2 Diagramme de flux

L'objectif poursuivi en dessinant ce diagramme de flux est de montrer ce qui permet de relier les itérations successives du protocole de ce mémoire.

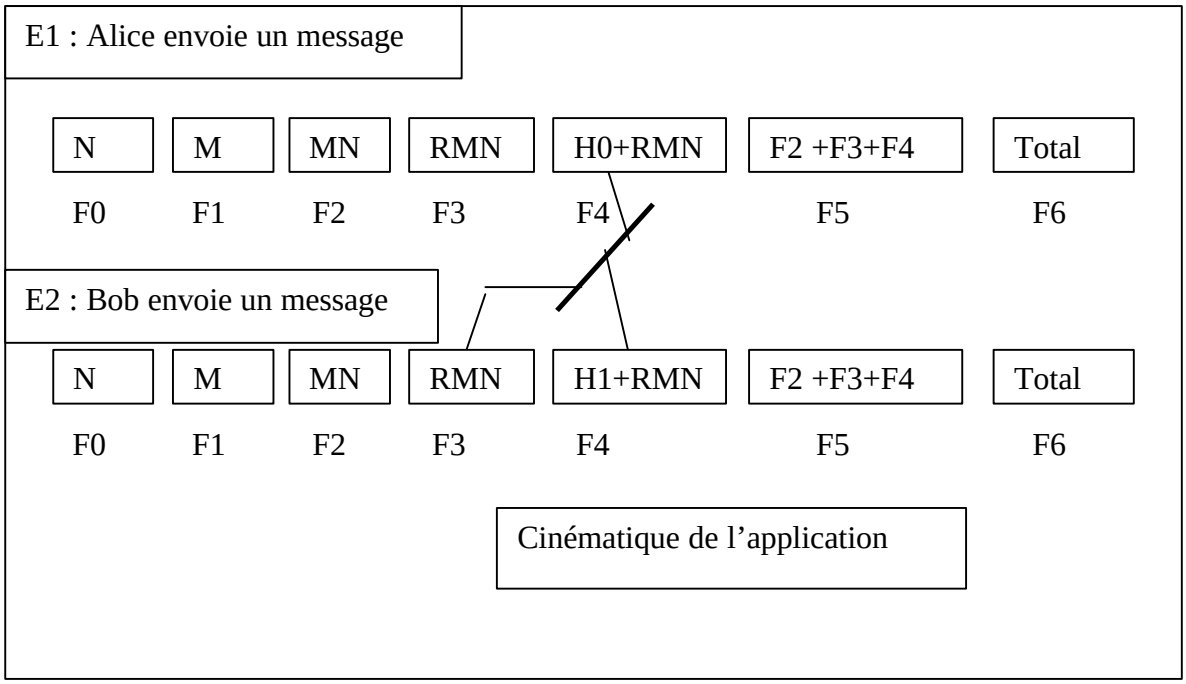

Figure 17 : Cinématique de l'application

Expliquons la notation utilisée dans cette illustration :

- F0, F1, F2, F3, F4, F5, F6 désignent des fichiers générés dans le déroulement du protocole;
- M désigne le message ;
- N désigne le nonce ;
- MN désigne le message plus le nonce ;
- RMN désigne le résumé résultant du hachage du message plus le nonce ;
- H0 désigne l'historique existant lors de l'itération 0 ;
- H1 désigne l'historique existant lors de l'itération 1, c'est le résumé qui résulte du hachage de H0 plus RMN.

Si le fichier F4 de Alice existe et si le fichier F3 de Bob existe, alors il devient possible de générer le fichier F4 de Bob, qui sera l'historique du prochain événement. Ainsi, nous utilisons à la fois une sorte de formalisme des réseaux de Pétri à propos de l'algorithme de Piggybacking.

Lors de la prochaine itération du protocole (E3), H2 résultera du hachage de H1 et de RMN.
L'utilisation du nonce répond à deux types de besoins. Le nonce permet d'une part de nommer de manière unique les fichiers, d'autre part le nonce permet aussi de contrer un certain nombre d'attaques. En effet, le fait de concaténer le nom du fichier avec le nonce généré à l'occasion unique de l'événement « envoi de message », permet de différencier sans équivoque possible les fichiers issus d'itérations successives de l'application. Dans notre exemple, le fichier F2_1359 qui correspond à l'événement généré par Alice qui donne rendez-vous à Bob lundi, est différent du fichier F2_5591 qui correspond à l'événement généré par Bob qui répond qu'il est d'accord pour se rendre au rendez-vous proposé par Alice.

2.5.3 Les interfaces

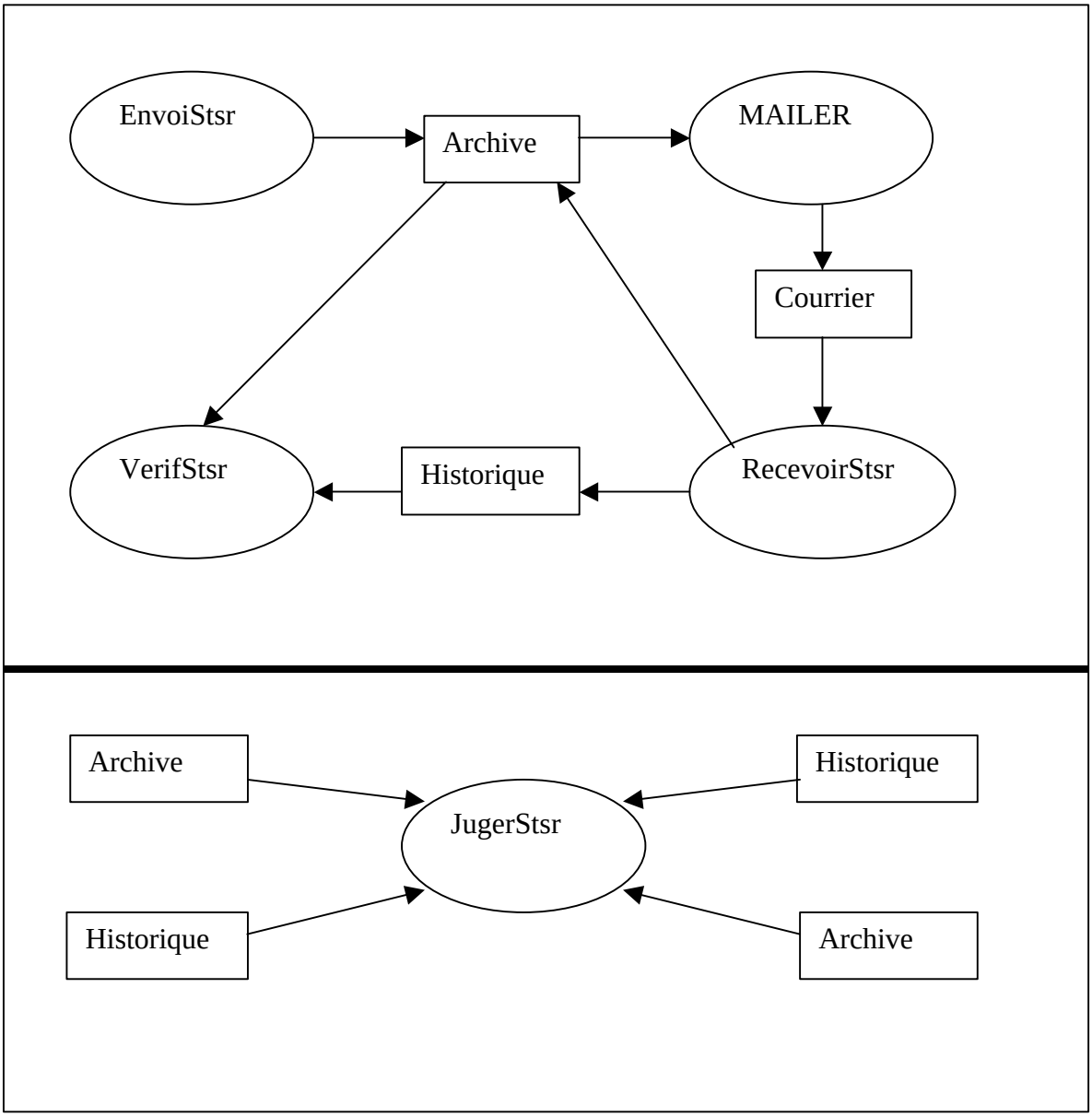

Figure 18 : Interfaces entre les applications

Dans cette illustration, les rectangles symbolisent les interfaces et les ellipses englobent les instances d'un processus d'activité (application). Les interfaces doivent être vues comme des structures de données dans lesquelles les processus peuvent selon le cas déposer de l'information ou bien retirer de l'information.

A ce stade de la spécification du protocole, il est trop tôt pour caractériser précisément les interfaces, le choix d'implantation est encore inconnu. Cependant, le degré de raffinement est suffisant pour nous permettre de différencier les entrées et les sorties. C'est la raison pour laquelle les liaisons entre les activités et les zones de structures de données sont orientées.

2.5.4 Réseau de Pétri de l'application Envoistsr

L'application Envoistsr est utilisée par l'auteur d'un message qui doit préparer la signature de ce message avant d'envoyer le message et la signature à l'acteur avec lequel il communique.

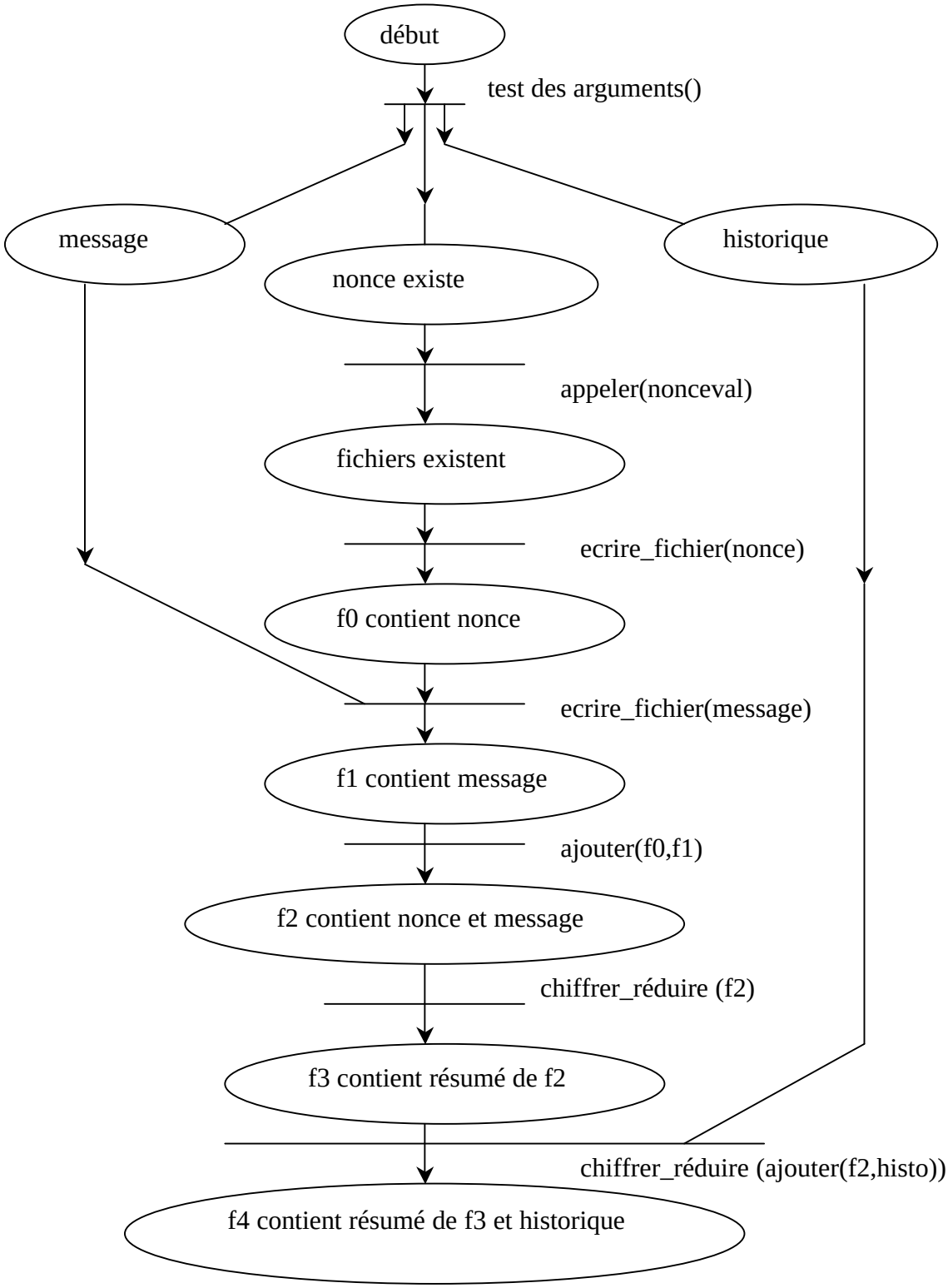

Figure 19 : Réseau de Pétri de l'application EnvoiStsr

2.5.5 Présentation informelle de l’algorithme de l'application EnvoiStsr

Nous présentons ici la liste des différentes opérations réalisées par un site lors du protocole d’envoi de message.

Test de validité des arguments de la ligne de commande
argument0 = le message ~/setsar/courrier/message
argument1 = le fichier historique du tour précédent = ~/setsar/histo/h0
Le nonce est généré une fois par message envoyé
Le nonce sert pour nommer des fichiers
Création des noms des fichiers relatifs pour le répertoire archive
Le fichier f0_nonceval; le nonce
Le fichier f1_nonceval; le message
Le fichier f2_nonceval; le nonce + le message
Le fichier f3_nonceval; empreinte de F2
Le fichier f4_nonceval; empreinte de Historique précédent + f3
Le fichier f5_nonceval; f2 + f3 + f4
Le fichier f7_nonceval; l'historique, le fichier f4_nonceval du tour précédent
Création des fichiers sur le disque
Sauvegarde du nonce dans le fichier f0.nonceval
Sauvegarde du fichier message (arg[0]) dans f1.nonceval
Sauvegarde du fichier historique (arg[1]) dans f7.nonceval
Concaténer f0.nonceval avec f1.nonceval vers f2.nonceval
Chiffrer réduire f2
Lecture du fichier f3.nonceval
Ajouter f3.nonceval avec f7.nonceval
Réduire et chiffrer le tout, le résultat est dans f4.nonceval
Ajouter f3.nonceval avec f7.nonceval résultat dans "tempo7"
Pour embrouiller le pirate éventuel, on écrase le fichier /tmp/temp7 avec un message dédié.

2.5.6 Réseau de Pétri de l'application VerifStsr

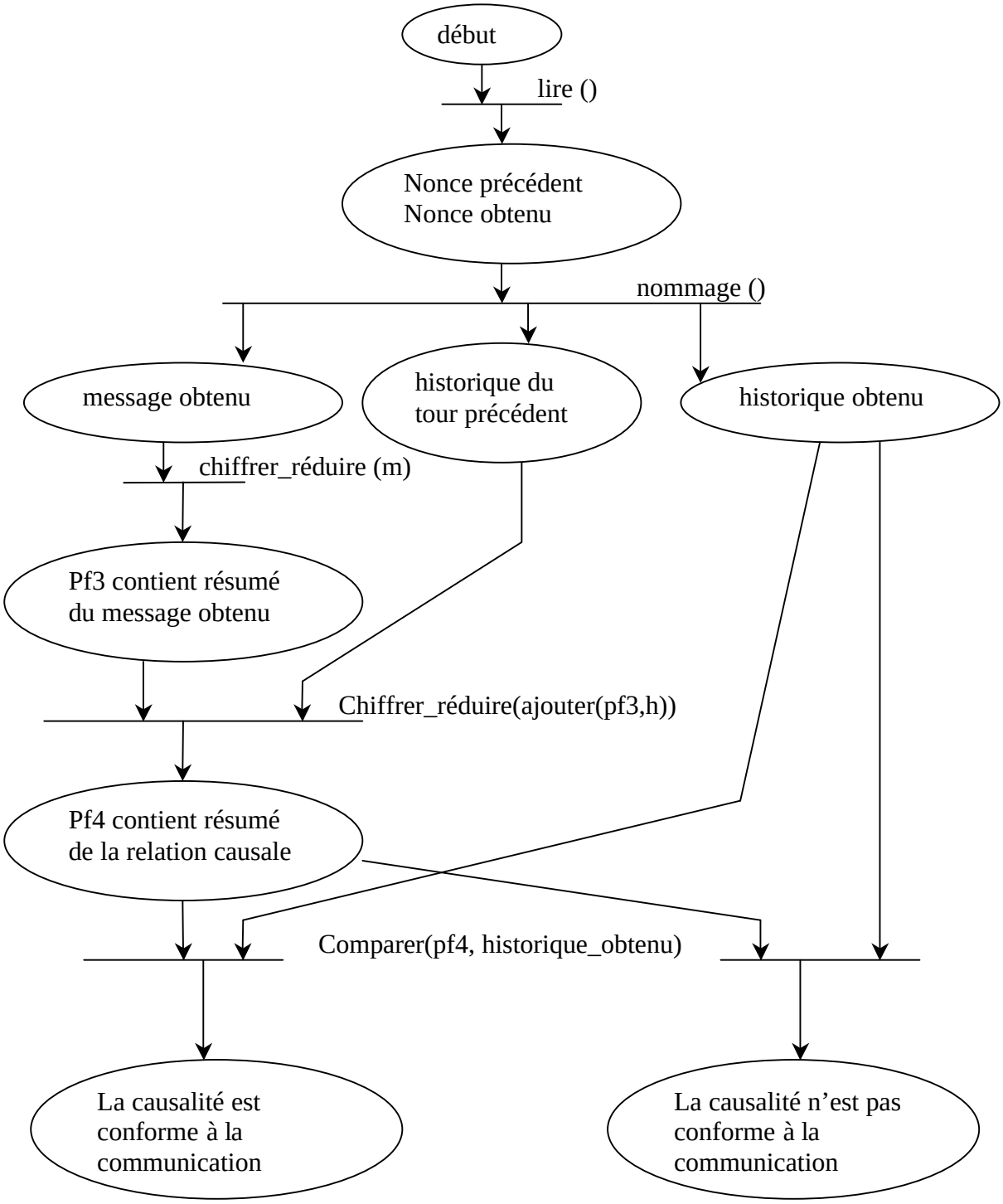

Figure 20 : Réseau de Pétri de l'application VerifStsr

2.5.7 Présentation informelle de l'algorithme de l'application VerifStsr

Nous présentons ici la liste des différentes opérations réalisées par un site lors du protocole de vérification de message.
Test de validité des arguments de la ligne de commande
Construction des arguments nécessaires à partir des données présentes dans les structures h_state_histo et h_state_archive
Le nonce nécessaire est lu à partir de archi.dat. Ainsi, on obtient nonceval correspondant au message reçu.
Un indice est incrémenté, ce qui permet de lire à partir de archi.dat le nonceval correspondant au message envoyé au tour précédent.

Raisonnons en terme d'arguments pour désigner les éléments nécessaires aux calculs.
argument0 = le message avec nonce obtenu ~/setsar/histo/f2.nonceval
argument1 = le fichier historique du tour précédent = ~/setsar/archive/h0
argument2 = le fichier historique obtenu ~/setsar/histo/f4.nonceval

Création des noms des fichiers relatifs pour le répertoire de preuve (~/setsar/verif)
Le fichier pf2_nonceval; le nonce + le message
Le fichier pf3_nonceval; empreinte de pf2
Le fichier pf4_nonceval; empreinte de Historique précédent + pf3
Le fichier pf5_nonceval; pf2 + pf3 + pf4 non utilisé
Le fichier pf7_nonceval; l'historique, le fichier pf4_nonceval du tour précédent

Création des fichiers sur le disque dans le répertoire ~/setsar/verif
sauvegarde du fichier nonce + message (arg[0]) dans pf2
sauvegarde du fichier historique (arg[1]) dans pf7
chiffrer et réduire pf2
Ajouter pf3 avec pf7 résultat dans "tempo7"
réduire et chiffrer le tout, le résultat est dans pf4
Ajouter pf2 avec pf3 et avec pf4
Comparaison de ~/setsar/verif/pf4 avec ~/setsar/courrier/f4.nonceval
    Si la comparaison donne vrai, l'auteur n'a pas construit une fausse causalité
    sinon l'auteur a construit une fausse causalité

Destruction des fichiers de vérification qui ne servent plus à rien.

2.5.8 Preuve de l'application VerifStsr

2.5.8.1 Introduction

Ce chapitre est dédié à la preuve de la sous-application VerifStsr. Un acteur obtient un message de la part d'un auteur. Cet acteur, le destinataire, veut vérifier par lui-même que la relation causale n'a pas été soit détruite, soit fabriquée. A partir des spécifications fonctionnelles rédigées et de l'architecture définie, exprimons des cas d'utilisations de ce protocole et montrons le plus rigoureusement possible que l'algorithme est satisfaisant pour toutes les actions intentionnelles de sabotage envisagées.
Nous proposons de montrer qu'une entité reconnaît un ensemble de messages qui sont apparus dans un ordre précis si bien que la sécurisation de la relation d'ordre causal est une des propriétés de la spécification du protocole SETSAR.

2.5.8.2 Le dictionnaire

Sous la dénomination de dictionnaire, sont rassemblés la syntaxe, les invariants, les variables et les structures de données choisies pour concevoir le protocole dont nous cherchons à prouver le degrés de faisabilité. Les axiomes ne sont pas décrits mais sont admis comme principe de la construction scientifique.

nonce : numéro aléatoire fabriqué pour cet événement unique.
h() : fonction de hachage
0=<i<n : l'indice de l'événement
N(i) : le nonce associé à l'événement indicé par i
EM(i) : le message écrit par l'auteur, ce message contient un nonce
RM(i) : le message obtenu par le destinataire
h(EM(i)) : le résumé du message écrit par l'auteur
/A : structure de données contenant les données préparées pour envoi
/H : structure de données contenant les données obtenues par reception
/A/F2_N(i) : (EM(i) + N(i))
/A/F3_N(i) : h((EM(i) + N(i)))
/A/F4_N(i) : h((/A/F3_N(i) + /H/F4_N(i-1)))
si i=0 alors /A/F4_N(i-1) = /A/init
/H/F2_N(i) : RM(i)
/H/F4_N(i) : h(EM(i))
si i=0 alors /H/F4_N(i-1) = /H/init
/H/bogue_F2_N(i) : message défectueux obtenu par le destinataire
/H/bogue_F4_N(i) : résumé défectueux obtenu par le destinataire

De manière informelle :

/A/init message d'initialisation contient le mot init et seulement ce mot
/H/init message d'initialisation contient le mot init et seulement ce mot
/A/F4_N(i-1) est le résumé créé lors de l'étape précédente
/H/F4_N(i-1) est le résumé obtenu lors de l'étape précédente

Syntaxe de la preuve formelle :

A gauche du séquent de Gentzen figure les prémisses de la règle, c'est à dire les spécifications du protocole, les hypothèses de fonctionnement, les données rémanentes et ce que le destinataire a obtenu, reçu. Tandis qu'à droite du séquent de Gentzen figure les conséquences de la règle. Le protocole a généré des données, une coupure dans le temps a été effectuée, l'enjeu de la preuve et des spécifications est de déterminer si cette coupe est cohérente alors que le protocole a atteint une cible à un instant t.

R(x,m) veut dire que l'utilisateur x reconnaît le message m.
7 R(x,m) veut dire que l'utilisateur x ne reconnaît pas le message m.
Exemple : Si Alice possède dans ses archives le message e, et si Bob lui envoie le message e', alors, le message e précède causalement le message e'.
Γ : la lettre grecque gamma représente les prémisses de la règle ;
⊢ : ce symbole représente la règle d'inférence principale du formalisme des séquents de Gentzen et signifie « si ce qui précède est satisfait, alors » ;
R(a,e) : représente un prédicat qui signifie que Alice reconnaît le message e, créé par Alice;
R(a,e') : représente un prédicat qui signifie que Alice reconnaît le message e', créé par Bob ;
Δ : la lettre grecque delta représente les conséquences de la règle.
alors nous obtenons l'équation de preuve suivante : Γ, R(a,e), R(a,e') ⊢ Δ
CAUS(G) voir paragraphe [1.7.1]
RECEPTION(X,RG) voir paragraphe [1.7.1]

2.5.8.3 Sommaire de la preuve statique de VerifStsr

Il faut scinder les cas possibles d'attaque à l'encontre de SETSAR pouvant être détectés en utilisant la sous-application VerifStsr. Cette liste d'attaque n'est pas complète.
Nous suivrons le plan suivant :
A) Alice est le destinataire
A.1) En cas de répudiation (DENIAL) et Bob a triché
A.1.1) C'est la première fois qu'Alice obtient un message et Bob a triché
A.1.1.1) Bob répudie e, mais construit h(e') à partir de h(e)
A.1.1.2) Bob répudie e, et ne construit pas h(e') à partir de h(e)
A.1.2) C'est au moins la deuxième fois qu'Alice obtient un message et Bob a triché
A.1.2.1) Bob répudie e, et ne construit pas h(e') à partir de h(e)
A.2) En cas de falsification (FORGERY), Bob a triché
A.2.1) C'est la première fois que l'acteur Alice obtient un message
A.2.1.1 Bob a modifié le message avant de l'envoyer
A.2.1.2 Bob a modifié le résumé avant de l'envoyer
A.2.1.3 Le résumé a été construit à partir d'autre chose que init
A.2.1.4 Le résumé semble correct, mais ne correspond pas aux données.
A.2.2) C'est au moins la deuxième fois que l'acteur Alice obtient un message
A.2.2.1.1) Bob a modifié le message avant de l'envoyer
A.2.2.1.2) Bob a modifié le résumé avant de l'envoyer
A.2.2.1.3) Le résumé de la relation a été construit par Bob à partir de données non conformes.
A.2.2.2) Bob répudie un message ancien
A.3) Alice affirme que Bob a triché, mais Bob n'a pas triché
A.3.1) Alice prétend que le résumé qui lui a été envoyé ne correspond pas.
A.3.2) Alice prétend que le message qui lui a été envoyé ne correspond pas.
A.3.3) Alice prétend qu'elle ne reconnaît ni le bon message ni le bon résumé.

B) Bob est le destinataire
B.1) En cas de répudiation (DENIAL), Alice a triché
B.1.1) C'est la première fois que Bob obtient un message
B.1.1.1) Supposons que Alice nie avoir eu connaissance de m1, elle ne reconnaît pas m1
B.1.1.1.1) Quelqu'un a pris l'identité d'Alice
B.1.1.1.2) Alice n'a pas lu ce qu'elle écrivait ou bien elle était hors d'état de comprendre.
B.1.1.2) Alice envoie un message correct et un résumé faux.
B.1.2) C'est au moins la deuxième fois que Bob obtient un message
B.1.2.1) Alice a modifié le message avant de l'envoyer
B.1.2.2) Alice a modifié le résumé avant de l'envoyer, si bien que le résumé ne correspond pas
B.1.2.3) Alice téléphone à Bob et affirme que m1 et m2 sont indépendants
B.2) En cas de falsification (FORGERY), Alice a triché
B.2.1) C'est la première fois que l'acteur Bob obtient un message
B.2.1.1) Alice a modifié le message m1 avant de l'envoyer
B.2.1.2) Alice a modifié le résumé avant de l'envoyer
B.2.1.3) Alice a construit le résumé à partir d'autre chose que init.
B.2.1.4) Alice a construit un résumé qui semble correct
B.2.2) C'est au moins la deuxième fois que l'acteur Bob obtient un message
B.3) Bob affirme qu'Alice a triché, mais Alice n'a pas triché

Nous savons que Alice détient les éléments suivants :

- une structure de donnée /A/archi.dat contenant tous les N(i) classés
- une copie du message qu'elle a envoyé au tour précédent : /A/F2_N(i-1)
- une copie du message qu'elle a obtenu de Bob : /H/F2_N(i)
- le résumé qu'elle a envoyé au tour précédent : /A/F4_N(i-1)
- le résumé qu'elle a obtenu de Bob : /H/F4_N(i)

2.5.8.4 Preuve de VerifStsr si Alice est destinataire

A) En cas de répudiation (DENIAL) et Bob a triché
Dans ce paragraphe, nous montrons que si le destinataire Alice obtient le message e' (m2), et que si le message e (m1) précède e' (m2), alors l'auteur Bob ne peut pas nier que e (m1) est avant e' (m2). Nous montrons de plus que l'auteur Bob ne peut pas prétendre que e (m1) n'est pas avant e' (m2). Toutefois, Bob peut prétendre que e et e' sont indépendants.

A.1.1) C'est la première fois que l'acteur Alice obtient un message

Le message précédent (e) a été envoyé par Alice. Nous supposons que le site de Alice est fiable en terme de sauvegarde des données persistantes. Aucun opposant ne peut altérer les sauvegardes. En tant que destinataire, Alice possède donc des éléments que nous allons exploiter. En ce qui la concerne, Alice a des éléments pour reconstruire l'ordre local et causal conformément à ce qui s'est passé.
Dans ce cas 1.1 S(P) est une sécurisation faible pour CAUS(G)

Le message m1 est antérieur au message m2 et Alice dispose de données non pas pour le prouver car ce n'est pas le but, mais pour contrer un opposant qui voudrait lui prouver le contraire.
De plus si Bob avait pris en considération le message d'Alice et le résumé de la relation causale envoyé par Alice, alors sans aucun doute possible, m1 et m2 seraient dépendants l'un de l'autre. Par conséquent, m1 et m2 ne sont pas indépendants.
A.1.1.1) Supposons que Bob nie avoir eu connaissance de m1 - il ne reconnaît pas m1 - mais il envoie tout de même un résumé qui contient le résumé de m1 qui est F4_N(i-1)
Alice construit PF4 = h((/H/F2_N(i)+(/A/F4_N(i-1))
PF4 est conforme, la relation locale et causale est vérifiée.

Alice peut affirmer que, conformément aux spécifications de la sous-application RecevoirStsr, les deux informations distinctes, message et résumé, sont obtenues de manière atomique. Nécessairement, Bob exploite le résumé du message, cela prouve indubitablement qu'il a reçu le message correspondant. Bob ne peut pas dire qu'il n'a pas pris connaissance du message, il s'est engagé à lire le message correspondant au résumé, il a exploité le résumé et il nie avoir reçu le message. Bob est coresponsable, il doit cesser ce comportement immoral. Personne ne peut alléguer sa propre turpitude. C'est un adage du droit français.

Preuve formelle :

Γ, R(a, h(e)), R(a, e), R(a,e'), R(a, h(e')) $\vdash \Delta$
Dans ce cas, les séquents de Gentzen ne fournissent pas de résultat probant.

Dans ce cas 1.1.1, S(P) est une sécurisation faible pour CAUS(G) et pour RECEPTION(X,RG), donc S(P) est une sécurisation forte de P pour l'assertion CAUS(G).

A.1.1.2) Supposons maintenant que Bob nie avoir obtenu m1, et qu'il envoie un résumé qui ne contient pas le résumé de m1. Tout ce passe comme si Bob envoyait un message pour la première fois à Alice, et de plus, tout se passe comme s'il n'y avait pas d'historique de communication entre Bob et Alice. Le seul historique est init, qui est commun aux deux sites, celui de l'auteur et celui du destinataire. Nous allons nous rendre compte que la preuve ne peut pas être mise en défaut. Dans un premier mouvement, il semble que le protocole ait été mis en défaut, mais il n'en est rien. Cela n'est pas gênant, bien au contraire, nous mettons en évidence la datation de la formalisation d'un contrat synallagmatique. De plus amples explications sont disponibles au chapitre [3]. Ici Bob sort lui-même du protocole.

Preuve formelle :

Γ, R(a, h(e)), R(a, e), R(a, e'), R(a, h(e')) $\vdash \Delta$
Dans ce cas, les séquents de Gentzen ne fournissent pas de résultat probant.

Supposons que Bob a envoyé un résumé bogue_F4_N(i) qu'il a obtenu en utilisant l'invariant présent par défaut sur son site. Nous obtenons l'égalité suivante :
bogue_F4_N(i) = h(/A/F2_N(i) + /H/init)

Notons qu'il aurait dû procéder de la manière suivante :
F4_N(i) = h(/A/F2_N(i) + /H/F4_N(i-1))
En effet sur le site de Bob, à ce stade du programme, il n'existe pas de résumé d'historique autre que celui envoyé par Alice et cet historique est /H/F4_N(i-1).

Alice construit par calcul PF4 = h((/H/F2_N (i) + (/A/F4_N(i-1))))
Nous nous apercevons que PF4 est différent de bogue_F4_N(i) puisque (/A/F4_N(i-1)) est différent de /H/init du site de Bob qui correspond à /A/init sur le site de Alice. Les deux résumés sont différents, une erreur apparaît, une lésion existe dans le déroulement du contrat, il convient d'en assumer l'existence. Techniquement parlant, la propriété déterministe des fonctions de hachage assure que deux sources identiques doivent fournir deux résumés identiques. Si deux résumés différents existent, alors nécessairement les éléments d'origine sont différents.

Ainsi Alice se rend compte que Bob n'a pas envoyé un résumé conforme à l'ordre local et causal. Ainsi Bob doit se rendre compte qu'il n'a pas envoyé un résumé conforme à l'ordre local et causal.

Certes, Bob peut affirmer qu'il communique avec Alice pour la première fois et qu'il n'a rien reçu, mais ce protocole est fait pour que Alice et Bob puissent communiquer, si Alice envoie un premier message m1 à Bob et que Bob nie avoir reçu m1 et le résumé correspondant, alors Bob en envoyant m2 et un résumé qui ne correspond pas à m1 et m2, n'est pas en mesure de dire qu'il communique avec Alice, par contre Alice est en mesure d'affirmer que Bob ne communique pas avec elle conformément aux exigences du protocole.

Nous avons montré dans les cas A.1.1.1 et A.1.1.2 que si PF4 est différent de /H/F4_N(i) alors le résumé envoyé par Bob n'est pas celui qui est garant de l'ordre causal, Bob n'a pas pris en considération le message m1 que dans notre formalisme nous avons appelé /A/F2_N(i-1) sur le site de Alice. Par conséquent, Bob ne peut pas prouver que les messages sont indépendants, et Bob ne peut pas nier que m1 est avant m2 sans qu'Alice s'en aperçoive, et de

plus cela prouve que Bob l'auteur du nouveau message n'a pas créé le résumé F4 à partir des données requises.
Dans ce cas A.1.1 S(P) est une sécurisation forte pour RECEPTION(X,RG)

A.1.2) C'est au moins la deuxième fois que l'acteur obtient un message
Le message précédent a été envoyé par Alice. Le destinataire Alice possède donc les éléments suivants :

- une structure de donnée /A/archi.dat contenant tous les N(i) classés
- une copie du message qu'elle a envoyé au tour précédent : /A/F2_N(i-1)
- le résumé qu'elle a envoyé au tour précédent : /A/F4_N(i-1)
- une copie du message qu'elle a obtenu de Bob : /H/F2_N(i)
- le résumé qu'elle a obtenu de Bob : /H/F4_N(i)

Les éléments sont identiques à ceux disponibles au cas précédent A.1.1. La démonstration est identique, les mêmes faits dans les mêmes conditions procurent les mêmes effets. Dans ce cas A.1.2 S(P) est une sécurisation faible pour RECEPTION(X,RG).
Preuve formelle :
Bob envoie e' et n'envoie pas h(e')
R(a,e)
R(a,e')
7R(a,h(e')
$\Gamma, R(a, e), R(a, h(e)), R(a, e'), 7R(a, h(e')) \vdash \Delta$
$\Gamma, (R(a, e), R(a, e')) \rightarrow R(a, h(e')), 7R(a, h(e')) \vdash \Delta$ $\rightarrow \vdash$
$\Gamma, R(a, e), R(a, h(e')) \vdash R(a, e), R(a, e')), \Delta$ aff $\vdash$
$\Gamma, R(a, e) \vdash R(a, e), R(a, e'), \Delta$ $\vdash$ aff
$\Gamma, R(a, e) \vdash R(a, e), \Delta$ axiome

$\Gamma, R(a, e), R(a,e'), 7R(a, h(e')) \vdash R(a, h(e')), 7R(a, h(e')), \Delta$ aff $\vdash$
$\Gamma\ 7R(a, h(e')) \vdash R(a, h(e')), 7R(a, h(e')), \Delta$ $\vdash$ aff
$\Gamma\ 7R(a, h(e')) \vdash 7R(a, h(e')), \Delta$ axiome

A.2) En cas de falsification (FORGERY) et Bob a triché
Dans ce paragraphe, nous montrons que si le destinataire obtient un message m2 (e'), et que le message m1 (e) est situé avant m2 (e') dans la relation d'ordre causale, alors l'auteur ne peut pas prétendre qu'un message m3 (e") est situé avant m2 (e') et après m1 (e) dans la relation d'ordre causale.

Si l'auteur prépare le résumé de e' à partir de e'' et de e' au lieu de préparer le résumé à partir de e et de e', alors l'auteur n'envoie pas h(e'), l'auteur envoie 7h(e')
Bob envoie e' et 7h(e')
Alice recalcule h(e')

Preuve formelle :
$\Gamma, (R(a, e), R(a, e')) \rightarrow R(a, h(e')), 7R(a, h(e')) \vdash \Delta$ $\rightarrow \vdash$
$\Gamma, R(a, e), R(a, h(e')) \vdash R(a, e), R(a, e')), \Delta$ aff $\vdash$
$\Gamma, R(a, e) \vdash R(a, e), R(a, e'), \Delta$ $\vdash$ aff
$\Gamma, R(a, e) \vdash R(a, e), \Delta$ axiome

$\Gamma, R(a, e), R(a,e'), 7R(a, h(e')) \vdash R(a, h(e')), 7R(a, h(e')), \Delta$ aff $\vdash$

Γ 7R(a, h(e')) ⊢ R(a, h(e')), 7R(a, h(e')), Δ ⊢ aff
Γ 7R(a, h(e')) ⊢ 7R(a, h(e')), Δ axiome

Ainsi, nous avons prouvé formellement en faisant un raisonnement par l'absurde que l'auteur ne peut pas falsifier la relation d'ordre causale qui existe entre les deux partis.

A.2.1) C'est la première fois qu'Alice obtient un message.
Ce message est m2 (e'). Le destinataire Alice possède donc les éléments suivants :

- une structure de donnée /A/archi.dat contenant tous les N(i) classés
- une copie du message qu'elle a envoyé au tour précédent : /A/F2_N(i-1)
- le résumé qu'elle a envoyé au tour précédent : /A/F4_N(i-1)

- une copie du message qu'elle a obtenu de Bob : /H/F2_N(i)
- le résumé qu'elle a envoyé au tour précédent : /A/F4_N(i-1)
- le résumé qu'elle a obtenu de Bob : /H/F4_N(i)

Dans ce cas /A/F4_N(i-1) = init
Appliquons la même démonstration aux trois cas possibles suivants :

A.2.1.1 Bob a modifié le message avant de l'envoyer, si bien que le résumé ne correspond pas au message obtenu par Alice.
A.2.1.2 Bob a modifié le résumé avant de l'envoyer, si bien que le résumé ne correspond pas au message obtenu par Alice.
A.2.1.3 Le résumé a été construit à partir d'autre chose que init, imaginons par exemple que Bob a pris le résumé construit en utilisant un autre message.

Alice construit PF4 = h(h(/H/F2_N(i))+(/A/F4_N(i-1))))
Dans ce cas /A/F4_N(i-1) = init
Alice calcule le résumé qui correspond aux données transmises et aux données qu'elle détient.
Alice compare PF4 avec /H/F4_N(i), d'après la définition des fonctions de hachage, nécessairement à partir de deux entrées différentes le calcul ne fournit pas deux fois le même résumé. Une infime modification du texte d'origine provoque obligatoirement une modification quasi totale du résumé obtenu en appliquant au texte une fonction de hachage.
Alice se rend compte que PF4 est différent de /H/F4_N(i)
Alice considère que manifestement à partir des éléments fournis, Bob a mal agi et qu'il n'est pas en mesure de fournir des éléments prouvant qu'il a bien agi.

A.2.1.4 Le résumé semble correct, mais ne correspond pas aux données.
Considérons maintenant un cas de falsification plus élaboré. Bob a modifié le message, m2 devient m3. De plus, il produit un résumé faux par rapport à m3 et à m2 mais qui lui permet de construire un résumé identique à celui qui résulte de la relation causale qui existe vraiment, à savoir que m1 précède m2. Ainsi, Bob fournirait un message m3 qui ne correspondrait pas à la relation causale, Bob construirait le résumé de la relation causale à partir d'un résumé antérieur qu'il aurait fabriqué, Bob obtiendrait un résumé conforme à la relation causale, ce serait idéal pour lui et catastrophique pour la sécurité du protocole qui serait alors mise en défaut. Bob enregistrerait alors le message fabriqué, le résumé falsifié et le nouveau résumé qui semble conforme aux attentes d'Alice. Bob serait alors capable d'affirmer avec aplomb qu'en ce qui le concerne, son comportement est conforme aux obligations mutuelles qu'il doit respecter. Dans ce type d'attaque, il semble que Bob soit parvenu à faire croire qu'il a bien agi alors qu'il a falsifié des éléments de preuve.

Mais cela n'est pas possible en temps limité à cause de la propriété de non-collision des fonctions de hachage. En effet , reprenons la définition des fonctions de hachage. Soit h la fonction de hachage, D l'ensemble d'origine et R l'ensemble de destination.
h : D -> R et |D| > |R|
De par leur nature, il est certain que plusieurs messages peuvent procurer une valeur de résumé donné. Mais l'objectif poursuivi par les spécifications fonctionnelles des fonctions de hachage est de rendre ce travail infaisable en temps limité. Il est très difficile de trouver deux messages différents qui produisent le même résumé. Dans l'état actuel de développement technique des fonctions de hachage, la version 5 de Message Digest (MD5) assure que cette exigence est remplie.

Bob envoie e' et 7h(e')
Alice recalcule h(e')
Preuve formelle :

$$\frac{\Gamma, R(a,e), R(a,h(e')) \vdash R(a,e), R(a,e'), \Delta \quad \text{aff} \vdash}{\Gamma, R(a,e), R(a,h(e')) \vdash R(a,e), R(a,e'), \Delta \quad \vdash \text{aff}}$$

$$\frac{\Gamma, R(a,e) \vdash R(a,e), R(a,e'), \Delta}{\Gamma, R(a,e) \vdash R(a,e), \Delta \quad \text{axiome}}$$

$$\frac{\Gamma, R(a,e), R(a,e'), 7R(a,h(e')) \vdash R(a,h(e')), 7R(a,h(e')), \Delta \quad \text{aff} \vdash}{\Gamma\ 7R(a,h(e')) \vdash R(a,h(e')), 7R(a,h(e')), \Delta \quad \vdash \text{aff}}$$

$$\Gamma\ 7R(a,h(e')) \vdash 7R(a,h(e')), \Delta \quad \text{axiome}$$

Ainsi, dans les cas 2.1.1, 2.1.2 2.1.3 et 2.1.4 Alice se rend compte que le comportement de Bob est byzantin. Bob ne parvient pas à faire croire qu'il a bien agi. Dans ce cas 2.1 S(P) est une sécurisation faible pour CAUS(G) et S(P) est une sécurisation faible pour RECEPTION(X,RG), donc S(P) est une sécurisation forte de P pour une assertion CAUS(G)
A.2.2) C'est au moins la deuxième fois que l'acteur obtient un message.
Dans ce cas d'utilisation de l'algorithme de vérification, Alice est en mesure de vérifier ce qu'elle vient de recevoir. En effet la sous-application VerifStsr utilise les données indicées par 0 et par 1. L'indice 0 permet de pointer sur la donnée qui vient d'être obtenue. L'indice 1 permet de pointer sur la donnée qu'Alice avait envoyée au tour précédent. Ainsi, il est évident qu'Alice n'est plus du tout en mesure de vérifier une relation causale plus ancienne, il fallait le faire plus tôt dans le déroulement du protocole. Il semble dans une première approche que le comportement byzantin de Bob peut s'exprimer plus facilement parce que /A/F4_N(i-1) n'est pas égal à init. Le cas n'est donc pas figé. Dans ce cas /A/F4_N(i-1) résume l'historique de la relation causale du contrat d'obligations mutuelles qui lie Alice et Bob.

A.2.2.1) C'est donc au moins la deuxième fois qu'Alice obtient un message.
Ce message est mi avec i > 2
Considérons que le message s'appelle m3.
Le destinataire Alice possède donc les éléments suivants :

- une structure de donnée /A/archi.dat contenant tous les N(i) classés
- une copie du message qu'elle a envoyé au tour précédent : /A/F2_N(i-1)
- le résumé qu'elle a envoyé au tour précédent : /A/F4_N(i-1)

- une copie du message qu'elle a obtenu de Bob : /H/F2_N(i)
- le résumé qu'elle a envoyé au tour précédent : /A/F4_N(i-1)

- le résumé qu'elle a obtenu de Bob : /H/F4_N(i)

Appliquons la même démonstration aux trois cas possibles suivants :

A.2.2.1.1) Bob a modifié le message avant de l'envoyer, si bien que le résumé ne correspond pas au message obtenu par Alice.

Bob envoie 7e' et h(e')
Alice recalcule h(e')
Preuve formelle :

$\Gamma, (R(a, e), 7R(a, e')) \rightarrow 7R(a, h(e')), R(a, h(e')) \vdash \Delta$ $\rightarrow \vdash$

$\Gamma, R(a, e), 7R(a, h(e')), R(a, h(e')) \vdash R(a, e), 7R(a, e')), \Delta$ $\vdash$ aff

$\Gamma, 7R(a, h(e')), R(a, h(e')) \vdash \Delta$ $7 \vdash$

$\Gamma, R(a, h(e')) \vdash R(a, h(e')), \Delta$ axiome

$\Gamma, 7R(a, h(e')), R(a, h(e')) \vdash \Delta$ $7 \vdash$

$\Gamma, R(a, h(e')) \vdash R(a, h(e')), \Delta$ axiome

A.2.2.1.2) Bob a modifié le résumé avant de l'envoyer, si bien que le résumé ne correspond pas au message obtenu par Alice. Bob ne fait pas parvenir le bon résumé correspondant au message e', si bien que Alice ne reconnaît pas e'. De plus, Alice calcule un résumé qui sera différent du résumé 7h(e') que lui a envoyé Bob.
Bob envoie e' et 7h(e')
Alice calcule h(e')
Preuve formelle :

$\Gamma, (R(a, e), R(a, e')) \rightarrow R(a, h(e')), 7R(a, h(e')) \vdash \Delta$ $\rightarrow \vdash$

$\Gamma\ R(a, e), R(a, h(e')) \vdash R(a, e), R(a, e')), \Delta$ $\vdash$ aff

$\Gamma\ R(a, e), R(a, h(e')) \vdash R(a, e), \Delta$ aff $\vdash$

$\Gamma, R(a, e) \vdash R(a, e), \Delta$ axiome

$\Gamma, 7R(a, h(e')), R(a, h(e')) \vdash \Delta$ $7 \vdash$

$\Gamma, R(a, h(e')) \vdash R(a, h(e')), \Delta$ axiome

A.2.2.1.3) Le résumé de la relation a été construit par Bob à partir de données non conformes. Bob a utilisé autre chose que le résumé qu'il a obtenu au tour précédent, c'est à dire que le résumé qu'Alice avait construit au tour précédent n'a pas été utilisé par Bob. Imaginons par exemple, que Bob a pris un résumé construit en utilisant un autre message. Bob ne fait pas parvenir le bon message correspondant au résumé h(e), si bien que Alice ne reconnaît pas h(e'). Mais par calcul, Alice va construire un résumé de la relation de causalité et nécessairement, puisque les hypothèses et les règles de calculs sont identiques, le schéma de preuve va montrer que le résumé envoyé par Bob est différent du résumé calculé par Alice.

Bob envoie e' et 7h(e')
Alice calcule h(e')
Preuve formelle :

$\Gamma, (R(a, e), R(a, e')) \rightarrow R(a, h(e')), 7R(a, h(e')) \vdash \Delta$ $\rightarrow \vdash$

$\Gamma\ R(a, e), R(a, h(e')) \vdash R(a, e), R(a, e')), \Delta$ $\vdash$ aff

Γ R(a, e), R(a, h(e')) ⊢ R(a, e) ,Δ aff ⊢
Γ , R(a, e) ⊢ R(a, e) ,Δ axiome

Γ, 7R(a, h(e')) , R(a, h(e')) ⊢ Δ 7 ⊢
Γ, R(a, h(e')) ⊢ R(a, h(e')) , Δ axiome

Par raffinement des données, posons que le résumé qu'Alice a obtenu de Bob s'appelle /H/bogue_F4_N(i)

Alice construit PF4 = h(h(/H/F2_N(i))+(/A/F4_N(i-1))))
Alice calcule le résumé qui correspond aux données transmises et aux données qu'elle détient.
Alice compare PF4 avec /H/bogue_F4_N(i), d'après la définition des fonctions de hachage, nécessairement à partir de deux entrées différentes le calcul ne fournit pas deux fois le même résumé. Une infime modification du texte d'origine provoque obligatoirement une modification quasi totale du résumé obtenu en appliquant au texte une fonction de hachage.

Alice se rend compte que PF4 est différent de /H/F4_N(i)
Alice considère que manifestement à partir des éléments fournis, Bob a mal agi et qu'il n'est pas en mesure de fournir des éléments prouvant qu'il a bien agi.

Dans ce cas 2.2.1, S(P) est une sécurisation faible pour CAUS(G) et S(P) est une sécurisation faible pour RECEPTION(X,RG). Donc S(P) est une sécurisation forte de P pour une assertion CAUS(G)

A.2.2.2) Bob répudie un message ancien
Alice vient de recevoir le message m3 avec le résumé associé qui englobe la relation causale qui lie m1, m2 et m3 dans cet ordre. Bob affirme que le message bogue_m1 précède m2.
Nous savons que le protocole P, ne considère les éléments de la structure de données qui concerne le message en cours et le message précédent. VerifStsr ne permet pas de considérer les messages plus anciens. Par conséquent, il est évident que dans ce cas, Alice n'est pas en mesure de montrer que Bob a un comportement byzantin. Dans ce cas, S(P) n'est pas une sécurisation faible pour CAUS(G). et par conséquent S(P) n'est pas non plus une sécurisation forte pour CAUS(G).

A.3). Alice dit que Bob a triché alors que Bob n'a pas triché

Ce cas d'utilisation du protocole nous rappelle que tous les utilisateurs du protocole peuvent avoir un comportement byzantin. Ici et maintenant, nous avons un cas d'utilisation qui correspond à la propriété de non-répudiation par destination, et de plus, c'est Alice qui présente ce comportement.

A.3.1) Bob fait parvenir le bon résumé correspondant au bon message. Et Alice prétend que le résumé qui lui a été envoyé ne correspond pas au message envoyé.

Bob envoie e' et h(e')
Alice prétend que 7h(e')
Alice calcule h(e')

Preuve formelle :

$\Gamma, (R(a, e), R(a, e')) \rightarrow R(a, h(e')), 7R(a, h(e')) \vdash \Delta$ $\rightarrow \vdash$
$\Gamma\ R(a, e), R(a, h(e')) \vdash R(a, e), R(a, e'))\ ,\Delta$ $\vdash$ aff
$\Gamma\ R(a, e), R(a, h(e')) \vdash R(a, e)\ ,\Delta$ aff $\vdash$
$\Gamma\ , R(a, e) \vdash R(a, e)\ ,\Delta$ axiome

$\Gamma, 7R(a, h(e')) , R(a, h(e')) \vdash \Delta$ $7 \vdash$
$\Gamma, R(a, h(e')) \vdash R(a, h(e')) , \Delta$ axiome

A.3.2) Bob fait parvenir le bon résumé correspondant au bon message. Et Alice prétend que le message qui lui a été envoyé ne correspond pas au résumé envoyé.

Bob envoie e' et h(e')
Alice prétend 7e'
Alice calcule h(e')
Preuve formelle :

$\Gamma, R(a, h(e')) ,(R(a, e), R(a, e')) \rightarrow R(a, h(e')), 7R(a, e') \vdash \Delta$ $\rightarrow \vdash$
$\Gamma\ , R(a, e'), 7R(a, e') \vdash R(a, e), R(a, e'))\ ,\Delta$ $\vdash$ aff
$\Gamma\ , R(a, e'), 7R(a, e') \vdash \Delta$ $7 \vdash$
$\Gamma\ , R(a, e') \vdash R(a, e')\ ,\Delta$ axiome

$\Gamma, 7R(a, h(e')) , R(a, h(e')) \vdash \Delta$ $7 \vdash$
$\Gamma, R(a, h(e')) \vdash R(a, h(e')) , \Delta$ axiome

A.3.3) Bob fait parvenir le bon résumé correspondant au bon message, et Alice prétend qu'elle ne reconnaît ni le bon message ni le bon résumé.

Preuve formelle :

Bob envoie e' et h(e')
Alice calcule h(e')
$\Gamma, R(a, e) ,R(a, e'), R(a, h(e')), 7R(a, e'), 7R(a, h(e')) \vdash \Delta$
$\Gamma, (R(a, e) ,R(a, e')) \rightarrow R(a, h(e')), R(a, h(e')), 7R(a, e'), 7R(a, h(e')) \vdash \Delta$ $\rightarrow \vdash$
$\Gamma, R(a, h(e')), R(a, h(e')), 7R(a, e'), 7R(a, h(e')) \vdash \Delta$ aff $\vdash$
$\Gamma, R(a, h(e')), 7R(a, h(e')) \vdash \Delta$ $7 \vdash$
$\Gamma, R(a, h(e')), \vdash R(a, h(e')), \Delta$ axiome

$\Gamma,\ R(a, h(e')), 7R(a, e'), 7R(a, h(e')) \vdash R(a, h(e')), \Delta$ aff $\vdash$
$\Gamma,\ R(a, h(e')) \vdash R(a, h(e')), \Delta$ axiome

2.5.8.5 Preuve de VerifStsr si Bob est le destinataire

Bob est le destinataire du message que l'auteur Alice lui a envoyé.

B.1) En cas de répudiation (DENIAL) et Alice a triché
En cas de répudiation, Alice essaie de nier être l'auteur d'un événement qui a eu lieu au cours du déroulement du protocole.

B.1.1) C'est la première fois que l'acteur Bob obtient un message.
Ce cas est le plus simple de tous les cas envisageables puisque les données sont minimales. En effet un seul message a circulé entre les deux acteurs communicants et dans ce cas, le résumé de la communication n'est constitué que du simple mot init.
Bob n'a pas envoyé de message précédemment.
Bob détient les éléments suivants :

- le message envoyé par Alice : /H/f2_N(i)
- le résumé minimal : /A/init
- le résumé du message envoyé par Alice : /H/f4_N(i)

En ce qui le concerne, Bob détient des éléments pour reconstruire l'ordre local et causal conformément à ce qui s'est passé. m1 est le premier message et Bob dispose de données non pas pour le prouver, ce n'est pas l'objectif poursuivi par VerifStsr, mais Bob détient des données nécessaires pour contrer un opposant qui voudrait lui faire admettre autre chose.
Il suffit à Bob de construire PF4 tel que
PF4 = h(h(/H/F2_N(i))+(/A/init)))

Alice envoie e et h(e)
Bob calcule h(e)

Preuve formelle :
Γ, R(b, e), R(b, h(e)), 7R(b, e), 7R(b, h(e)) ⊢ Δ aff ⊢
Γ, R(b, e), 7R(b, e) ⊢ Δ 7 ⊢
Γ, R(b, e) ⊢ Δ, R(b, e) axiome

Dans ce cas 1.1 S(P) est une sécurisation faible pour CAUS(G)

B.1.1.1) Supposons que Alice nie avoir eu connaissance de m1 (elle ne reconnaît pas m1), Ce cas d'utilisation semble absurde, puisque Alice envoie un message m1, et en même temps elle nie envoyer un message. Toutefois, nous allons envisager deux cas qui sont englobés par ce cas de figure :
B.1.1.1.1) Quelqu'un a pris l'identité d'Alice et a envoyé un message à l'insu d'Alice en se faisant passer pour Alice. Ce défaut de sécurité est relatif à la propriété d'authentification. Ce problème est sérieux mais sort du cadre de la preuve de ce programme.
B.1.1.1.2) Alice n'a pas lu ce qu'elle écrivait ou bien elle était hors d'état de comprendre ce qu'elle écrivait suite à l'abus d'alcool et de jeux vidéo. Son immoralité est patente. Dans ce cas, selon un adage du droit français, personne ne peut alléguer sa propre turpitude, Alice peut tout au plus bénéficier de circonstances atténuantes.

Alice envoie e et h(e)

Bob calcule h(e)
Preuve formelle :

Γ, R(b, e), R(b, h(e)), 7R(b, e), 7R(b, h(e)) ⊢ Δ aff ⊢
Γ, R(b, e), 7R(b, e) ⊢ Δ 7 ⊢
Γ, R(b, e) ⊢ Δ, R(b, e) axiome

B.1.1.2) Alice envoie un message correct et un résumé faux. Ce cas est moins absurde, puisque Alice envoie un message et en même temps elle envoie un résumé qui ne correspond pas à ce message. Alice envoie bogue_F4_N(i) au lieu d'envoyer F4_N(i) qui correspond à F2_N(i).

Bob détient les éléments suivants :

- le message envoyé par Alice : /H/f2_N(i)
- le résumé minimal : /A/init
- le résumé du message envoyé par Alice : /H/f4_N(i)

Bob construit la preuve PF4 = h(h(/H/F2_N(i))+(/A/init)))
PF4 est conforme à ce qu'il aurait dû obtenir de la part d'Alice.
Mais Alice lui a envoyé bogue_F4_N(i)
Bob compare PF4 avec bogue_F4_N(i)
Bob se rend compte que les données sont différentes.

Alice envoie e et 7h(e)
Bob calcule h(e)
Preuve formelle :

Γ, (R(b, e), R(b, init)) → R(b, h(e)), 7R(b, h(e)) ⊢ Δ → ⊢
Γ, R(b, h(e)), 7 R(b, h(e)) ⊢ Δ 7 ⊢ aff
Γ, R(b, h(e)) ⊢ R(b, h(e)), Δ axiome

Γ, R(b, init) ⊢ R(b, e), R(b, init), Δ ⊢ aff
Γ, R(b, init) ⊢ R(b, init), Δ axiome

Les deux résumés sont différents. Une lésion existe dans le déroulement du protocole qui énonce les obligations mutuelles. Or, nous savons que la propriété fondamentale des fonctions de hachage nous permet d'affirmer que deux résumés différents résultent de calculs effectués sur des données d'origine différentes. Si les données sont identiques, alors les résumés doivent être identiques.
Alice affirme que le résumé bogue_F4_N(i) est conforme mais n'est pas en mesure de le prouver. De plus, Bob possède les éléments qui lui permettent d'affirmer que bogue_F4_N(i) n'est pas identique à PF4.
PF4 étant la structure obtenue par calcul, conformément au programme et aux données.

Dans ce cas 1.1, S(P) est une sécurisation faible pour CAUS(G) et S(P) est une sécurisation faible pour RECEPTION(X,RG). Donc S(P) est une sécurisation forte de P pour une assertion CAUS(G).

B.1.2) C'est au moins la deuxième fois que l'acteur Bob obtient un message de la part de l'acteur Alice Plusieurs cas de figures peuvent se produire, le cas idéal est le cas où aucune malversation n'a été intégrée au déroulement du protocole. Bob détient alors les éléments suivants :

- une structure de donnée /A/archi.dat contenant tous les N(i) classés
- le message envoyé par Alice : /H/F2_N(i)
- le résumé qu'il a construit au tour précédent : /A/F4_N(i-1)
- le résumé du message envoyé par Alice : /H/f4_N(i)

Bob construit PF4 = h(h(/H/F2_N(i))+(/A/F4_N(i-1)))

Bob construit PF4 = h(h(/H/F2_N(i))+(/A/F4_N(i-1)))
Bob compare PF4 avec /H/f4_N(i) et constate que les données sont identiques.
Alice a agi conformément au protocole et Bob aussi.
Dans ce cas 1.2 S(P) est une sécurisation faible pour RECEPTION(X,RG)

B.1.2.1) Alice a modifié le message avant de l'envoyer, mais Alice n'a pas modifié le résumé, si bien que le résumé ne correspond pas au message obtenu par Bob.
Bob détient les éléments suivants :

- une structure de donnée /A/archi.dat contenant tous les N(i) classés
- le message envoyé par Alice : /H/bogue_F2_N(i)
- le résumé qu'il a construit au tour précédent : /A/F4_N(i-1)
- le résumé du message envoyé par Alice : /H/f4_N(i)

Bob construit PF4 = h(h(/H/bogue_F2_N(i))+(/A/F4_N(i-1)))
Bob compare PF4 avec /H/f4_N(i) et constate que les données sont différentes.
Puisque Bob utilise les mêmes données que Alice et les mêmes outils de protocole que Alice, il peut raisonnablement affirmer qu'Alice n'a pas envoyé le résumé qui correspond au message qu'elle envoie. Alice a mal agi et ne peut pas faire croire qu'elle a bien agi.

Dans ce cas 1.2.1 S(P) est une sécurisation forte pour RECEPTION(X,RG)

B.1.2.2) Alice a modifié le résumé avant de l'envoyer, si bien que le résumé ne correspond pas au message obtenu par Bob.
Bob détient les éléments suivants :

- une structure de donnée /A/archi.dat contenant tous les N(i) classés
- le message envoyé par Alice : /H/F2_N(i)
- le résumé qu'il a construit au tour précédent : /A/F4_N(i-1)
- le résumé du message envoyé par Alice : /H/bogue_F4_N(i-1)

Bob construit PF4 = h(h(/H/F2_N(i))+(/A/F4_N(i-1)))
Bob compare PF4 avec /H/bogue_F4_N(i-1) et constate que les données sont différentes.
Bob utilise les mêmes données que Alice. Bob utilise la même fonction de hachage que celle qui a servi à créer le résumé envoyé par Alice. La différence qui provoque la lésion ne provient ni des outils, ni du résumé précédent (/A/F4_N(i-1), par conséquent la différence est causée par les données qui ont été envoyées par Alice. Alice a mal agi et n'est pas en mesure de faire croire qu'elle a agi conformément au protocole.

Alice envoie e’ et 7h(e’)
Bob obtient par calcul le résumé h(e)

Preuve formelle :
Γ, (R(b, e), R(b, e')) → R(b, h(e')), 7R(b, h(e')) ⊢ Δ → ⊢
Γ, R(b, h(e')), 7 R(b, h(e')) ⊢ Δ 7 ⊢ aff
Γ, R(b, h(e')) ⊢ R(b, h(e')), Δ axiome

Γ, R(b, e), 7R(b, h(e')) ⊢ R(b, e), Δ aff ⊢
Γ, R(b, e) ⊢ R(b, e), Δ axiome

Dans ce cas 1.2.2 S(P) est une sécurisation forte pour RECEPTION(X,RG)

B.1.2.3) Alice téléphone à Bob et affirme que m1 et m2 sont indépendants
m1 est le message que Bob lui a envoyé.
m2 est le message qu'elle a envoyé.
Alice n'a pas modifié le message m2 et n'a pas modifié le résumé correspondant.

Bob détient les éléments suivants :
- une structure de donnée /A/archi.dat contenant tous les N(i) classés
- le message envoyé par Alice : /H/F2_N(i)
- le résumé qu'il a construit au tour précédent : /A/F4_N(i-1)
- le résumé du message envoyé par Alice : /H/F4_N(i-1)

Bob construit PF4 = h(h(/H/F2_N(i))+(/A/F4_N(i-1))))
Bob compare PF4 avec /H/F4_N(i-1) et constate que les données sont identiques
Bob affirme que m1 précède causalement m2.
Bob infirme le fait que m1 ne précède pas m2.
Si comme le prétendait Alice m1 ne précédait pas m2,
alors PF4 serait différent de /H/F4_N(i-1).
Alice affirme une assertion qu'elle n'est pas en mesure prouver par un fait tangible, et même le fait qu'elle procure va à l'encontre de ce qu'elle affirme au téléphone.
Alice envoie e' et h(e') et affirme que 7h(e')
Bob calcule h(e')
Preuve formelle :
Γ, (R(b, e), R(b, e')) → R(b, h(e')), 7R(b, h(e')) ⊢ Δ → ⊢
Γ, R(b, h(e')), 7 R(b, h(e')) ⊢ Δ 7 ⊢ aff
Γ, R(b, h(e')) ⊢ R(b, h(e')), Δ axiome

Γ, R(b, e), 7R(b, h(e')) ⊢ R(b, e), Δ aff ⊢
Γ, R(b, e) ⊢ R(b, e), Δ axiome

Dans ce cas B.1.2.3 S(P) est une sécurisation forte pour RECEPTION(X,RG)

B.2) En cas de falsification (FORGERY) et Alice a triché
En cas de falsification Alice a créé une relation causale différente de la relation causale qui a eu lieu entre les événements successifs.
B.2.1) C'est la première fois que l'acteur Bob obtient un message.
Bob obtient un message et un résumé, Alice en est l'auteur, et c'est le premier message qui circule entre les deux acteurs du protocole de communication. Ce cas de falsification semble proche du cas de falsification que nous avons exploré précédemment lorsque Alice parvient à vérifier les données transmises par Bob. Toutefois, les deux cas doivent être dissociés pour

deux raisons. D'abord, le comportement byzantin d'Alice doit être mis en évidence de la même manière que le comportement Bob. Ensuite, le fait qu'un seul message ait circulé entre les deux entités change les données de la preuve, en effet, les données sont réduites à leur plus simple expression.
Plusieurs possibilités apparaissent, détaillons les.
B.2.1.1) Alice a modifié le message m1 avant de l'envoyer
Alice a envoyé bogue_F2_N(i) et F4_N(i)
Le destinataire Bob possède donc les éléments suivants :

- une structure de donnée /A/archi.dat contenant tous les N(i) classés

Dans ce cas la pile FILO contient init et N(i) qui est le nonce
qu'Alice a calculé sur le site d'Alice. Rappelons que init
est situé au sommet de la pile.

- une copie du message qu'il a obtenu d'Alice : /H/bogue_F2_N(i)
- le résumé minimal /A/init
- le résumé qu'il a obtenu d'Alice : /H/F4_N(i)

Bob construit PF4 = h(h(/H/bogue_F2_N(i) + /A/init))
en utilisant le protocole P.
Maintenant Bob compare PF4 et /H/F4_N(i), les deux résumés sont différents. Alice a mal agi et n'est pas en mesure de prouver qu'elle a bien agi.
Dans ce cas de base 2.1.1, S(P) est une preuve faible pour RECEPTION(X,RG)
Si Alice avait bien agi, S(P) serait une preuve faible pour CAUS(G)
Par conséquent, S(P) est une sécurisation forte de P pour CAUS(G)
B.2.1.2) Alice a modifié le résumé avant de l'envoyer
Alice a envoyé bogue_F4_N(i) et F2_N(i)
Le destinataire Bob possède donc les éléments suivants :

- une structure de donnée /A/archi.dat contenant tous les N(i) classés
- une copie du message qu'il a obtenu d'Alice : /H/ F2_N(i)
- le résumé minimal /A/init
- le résumé qu'il a obtenu d'Alice : /H/bogue_F4_N(i)

Bob construit PF4 = h(h(/H/F2_N(i) + /H/bogue_F4_N(i))) en utilisant le protocole P.
Maintenant Bob compare PF4 et /H/F4_N(i), les deux résumés sont différents. Alice a mal agi et n'est pas en mesure de prouver qu'elle a bien agi.
Alice envoie e et 7h(e)
Bob calcule h(e)
Preuve formelle :

$\Gamma, (R(b, e), R(b, init)) \rightarrow R(b, h(e)), 7R(b, h(e)) \vdash \Delta \quad \rightarrow \vdash$
$\Gamma, R(b, h(e)), 7\,R(b, h(e)) \vdash \Delta \quad 7 \vdash$
$\Gamma, R(b, h(e)) \vdash R(b, h(e)), \Delta$ axiome

$\Gamma, R(b, init), 7R(b, h(e)) \vdash R(b, e), R(b, init), \Delta \quad aff \vdash$
$\Gamma, R(b, init) \vdash R(b, e), R(b, init), \Delta \quad \vdash aff$
$\Gamma, R(b, init) \vdash R(b, init), \Delta$ axiome

Dans ce cas de base, S(P) est une preuve faible pour RECEPTION(X,RG)
Si Alice avait bien agi, S(P) serait une preuve faible pour CAUS(G)
Par conséquent, S(P) est une sécurisation forte de P pour CAUS(G)
B.2.1.3) Alice a construit le résumé à partir d'autre chose que init.
Alice a envoyé bogue_F4_N(i) et F2_N(i)

Ce cas 2.1.3 présente les mêmes données que le cas précédent, alors que la méthode suivie par Alice est différente. Le protocole S(P) fournit les mêmes résultats.
Par conséquent, S(P) est une sécurisation forte de P pour CAUS(G)

B.2.1.4) Alice a construit un résumé qui semble correct
Alice a envoyé F4_N(i) et bogue_F2_N(i)
Cela voudrait dire qu'Alice aurait réussi à résoudre les deux équations suivantes :
F4_N(i) = h (h(/H/bogue_F2_N(i) + /A/bogue_init))
F4_N(i) = h (h(/H/F2_N(i) + /A/init))
Mais nous savons, d'aprés la définition de non-collision des fonctions de hachage, nécessairement à partir de deux entrées différentes le calcul ne fournit pas deux fois le même résumé. Une infime modification du texte d'origine provoque obligatoirement une modification quasi totale du résumé obtenu en appliquant au texte une fonction de hachage.
Alice ne peut pas réaliser ce tour de force, ce cas ne tient pas. Nous réfutons donc cette éventualité de fonctionnement de S(P) Par conséquent, S(P) est une sécurisation forte de P pour CAUS(G)

B.2.2) C'est au moins la deuxième fois que l'acteur Bob obtient un message
Ce cas doit être rapproché de celui où Alice obtient un message pour la deuxième fois ou plus. Les hypothèses de fonctionnement sont les mêmes à ceci près qu'Alice devient Bob et que Bob devient Alice.
Tout fait a une cause, et les mêmes faits dans les mêmes conditions procurent les mêmes effets. Bob est en mesure de vérifier ce qu'il vient d'obtenir. Si Alice a mal agi, elle n'est pas en mesure de fournir des éléments qui tendent à montrer qu'elle a bien agi.
Dans ce cas S(P) est une sécurisation faible pour CAUS(G) et S(P) est une sécurisation faible pour RECEPTION(X,RG). Donc S(P) est une sécurisation forte de P pour une assertion CAUS(G)
Par contre si Alice répudie un message plus ancien, Bob n'est pas en mesure d'exhiber une preuve qui permettrait de montrer qu'Alice a mal agi. dans ce cas S(P) est une sécurisation faible pour CAUS(G) et S(P) n'est pas une sécurisation pour RECEPTION(X,RG)

2.5.8.6 Conclusion

Nous avons atteint ici la fin de la démonstration concernant la preuve de la sous-application VerifStsr, en conclusion, nous pouvons estimer que le protocole est bien formé et que les cas envisagés n'ont pas permis de mettre en défaut l'assertion suivante :
S(P) est une sécurisation faible pour CAUS(G) et S(P) est une sécurisation faible pour RECEPTION(X,RG). Donc S(P) est une sécurisation forte de P pour une assertion CAUS(G)
Cependant, dans le cas où le comportement byzantin de l'un des partis conduirait à vérifier une relation causale antérieure, la nécessité de prouver le protocole de Jugement serait avérée.

2.5.9 Organigramme de l'application JugementStsr

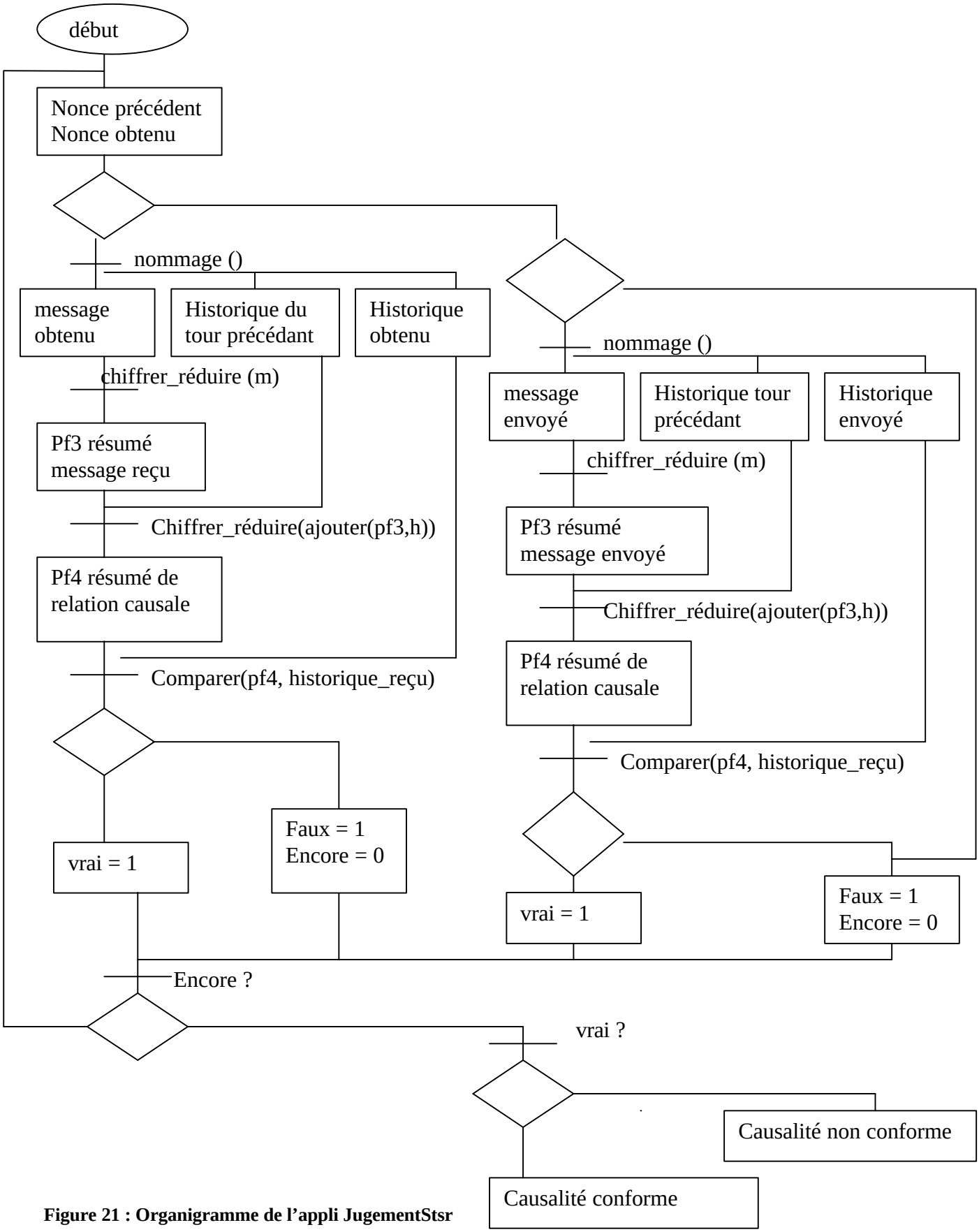

Figure 21 : Organigramme de l'appli JugementStsr

2.5.10 Présentation informelle de l'algorithme de l'application JugementStsr

Nous présentons ici la liste des différentes opérations réalisées par un site lors du protocole JugementStsr.

2.5.10.1 Objectifs de l'algorithme

On veut construire la pile qui contient tous les nonces depuis le début.
Rappel : le fichier archi.dat est garant de l'ordre local.
On veut vérifier que la relation causale n'a pas été brisée.
On veut déterminer à partir de deux messages lequel est le premier des deux.
Si 1 argument est présenté sur la ligne de commande
argument0 = c'est l'utilisateur que le juge doit juger
on fait la partie A et la partie B de l' algorithme
Si 3 arguments sont présentés sur la ligne de commande
argument0 = c'est l'utilisateur que le juge doit juger
argument1 = le premier nonce N1
argument2 = le deuxième nonce N2
on fait la partie A, la partie B et la partie C de l'algorithme

2.5.10.2 Partie A de l'algorithme : la relation locale

L'objectif de la partie A de l'algorithme de la sous-application JugementStsr est de construire une relation locale, d'ordonner tous les nonces conformément au fichier archi.dat.

Initialisation de la pile P1 qui contient tous les nonces du fichier archi.dat
Initialisation de l'index de la pile
Initialisation de l'index du fichier
Tant que (index du fichier pointe sur une entrée valide du fichier)
faire
    lire une ligne
    la valeur du nonce est enregistrée dans la pile
    incrémenter l'index de la pile
    incrémenter l'index du fichier
fait

2.5.10.3 Partie B de l'algorithme : la relation causale

L'objectif de la partie B de l'algorithme de la sous-application JugementSTSR est de vérifier que la relation causale n'a pas été brisée

Initialisation des fichiers de vérification
indice_ancien = taille_de_la_pile pour pointer sur élément initial de la pile FIFO
indice_nouveau = (taille_de_la_pile - 1) pour pointer sur élément 0 de la pile FIFO
indice_de_boucle = 0
encore = vrai
relation_locale = 0
relation_causale = 0
garde = 0
taille-archi = taille_de_la_pile

```
Tant que (encore = vrai)
faire
  nonceval1 = P1.lire(indice_ancien)
  nonceval2 = P1.lire(indice_nouveau)
  Dans ce cas le juge reconstruit les noms des fichiers tels que :
  Le message et le nonce ont été générés par l'autre acteur,
  Le message et le nonce se trouvent dans le répertoire histo
  Le résumé de l'itération précédente se trouve dans archive
  construire nom du fichier histo/nouveau_f2_nonceval2
  construire nom du fichier archive/ancien_f4_nonceval1
  construire nom du fichier histo/nouveau_f4_nonceval2
  si histo_f2_nonceval n'existe pas
        Dans ce cas le juge reconstruit les noms des fichiers tels que :
        Le message et le nonce ont été générés par l'auteur que le juge vérifie
        Le message et le nonce se trouvent dans le répertoire archive
        Le résumé de l'itération précédente se trouve dans le répertoire histo
        construire nom du fichier archive/nouveau_f2_(nonceval2)
        construire nom du fichier histo/ancien_f4(nonceval1)
        construire nom du fichier archive/nouveau_f4_(nonceval2)

        si fichier histo/ancien_f2_(nonceval2) n'existe pas
              la relation causale est brisée
              garde = 1
        fin si
  fin si
  générer pf3 = résumé de nouveau_f2_nonceval
  générer pf4 = résumé de (pf3 + ancien_f4_nonceval)
  si (pf4 != nouveau_f4_nonceval)
        causalité = faux
        encore = faux
        garde = 1
  fin si
  indice_ancien --
  indice_nouveau --
  indice_de_boucle ++
  si (indice_ancien =0 )
        toute la pile FIFO a été explorée
        encore = faux
  fin si
fait
si (taille_archi > 1)
        relation_locale = 1
si (taille_archi < indice_de_la_boucle)
        relation_causale = 1
```

2.5.10.4 Partie C de l'algorithme : l'oracle

L'objectif de la partie C de l'algorithme de la sous-application JugementSTSR est de donner une réponse simple à l'interrogation posée quant à l'ordre d'apparition des événements constitutifs de la relation d'ordre causal.
On a une liste de nonce dans une pile FIFO
On a 2 numéros de nonce N1 et N2
On veut savoir lequel des deux nonces connus est le premier, et lequel est le dernier des deux.

```
 premier = inconnu
 dernier = inconnu
tant que (index > -1)
faire
        lire la valeur du nonce dans la pile
        si valeur du nonce = N1
                si premier = inconnu
                alors premier = N1
                sinon dernier = N1
                fin si
        fin si
        si valeur du nonce = N2
                si dernier = inconnu
                alors premier = N2
                sinon dernier = N2
                fin si
        fin si
        décrémenter l'indice de la pile
fait
```

2.5.10.5 Partie D de l'algorithme : la conclusion

L'objectif de la partie D de l'algorithme de la sous-application JugementSTSR est d'apporter une conclusion en rassemblant les résultats intermédiaires.
Si taille_de_la_pile = indice_de_boucle
alors relation_causale = 1
fin si
Si taille_de_la_pile > 1 et relation_causale = 1
alors relation_locale = 1
fin si
Si relation_locale = 1 et relation_causale = 1 et garde = 0
alors la relation locale et la relation causale sont satisfaites
fin si
Si la partie B et la partie A se sont bien déroulées, alors la relation d'ordre causale et la relation d'ordre locale sont satisfaites.
Si la partie A s'est bien déroulée, mais que la partie B s'est mal déroulée, alors aucune conclusion n'est possible parce que la relation d'ordre locale aurait pu être fabriquée par l'utilisateur.
Si la partie A, la partie B se sont bien déroulées, si les deux nonces ont été trouvés, alors l'oracle peut se prononcer et fournir une réponse simple à l'ordre de précédence qui existe entre les deux événements distincts qui sont intervenus lors du déroulement du protocole. Il convient de noter que l'oracle ne peut se prononcer sur la précédence que dans la mesure où la relation d'ordre causale a validé la relation d'ordre locale.

2.5.11 Preuve de l'application JugementStsr

2.5.11.1.1 Introduction

Un litige est apparu au cours du protocole qui lie Alice et Bob. Le juge détient maintenant les archives de Bob et celles d'Alice. Nous supposons que les falsifications ou les destructions éventuelles résultent exclusivement des comportements byzantins d'Alice et de Bob.
Nous allons utiliser un schéma de preuve dynamique qui considère l'ensemble des solutions possibles en construisant des arbres de preuve dont les noeuds symbolisent des questions et les feuilles les réponses. Dans un premier temps, nous allons nous intéresser à l'arbre de preuve dynamique qui englobe l'ensemble des solutions relatives à la notion de tricherie. Nous choisissons des branches successives de l'arbre avant de nous prononcer sur la culpabilité éventuelle d'Alice ou de Bob. Dans un deuxième temps nous considèreront la preuve relative à l'oracle.

2.5.11.1.2 Tricherie éventuelle

Dans un premier temps nous allons envisager un arbre décisionnel qui concerne une tricherie éventuelle et l'identité du tricheur. L'application JugementStsr peut-elle mettre en évidence l'existence d'une éventuelle tricherie et dans ce cas désigner avec certitude le tricheur ?

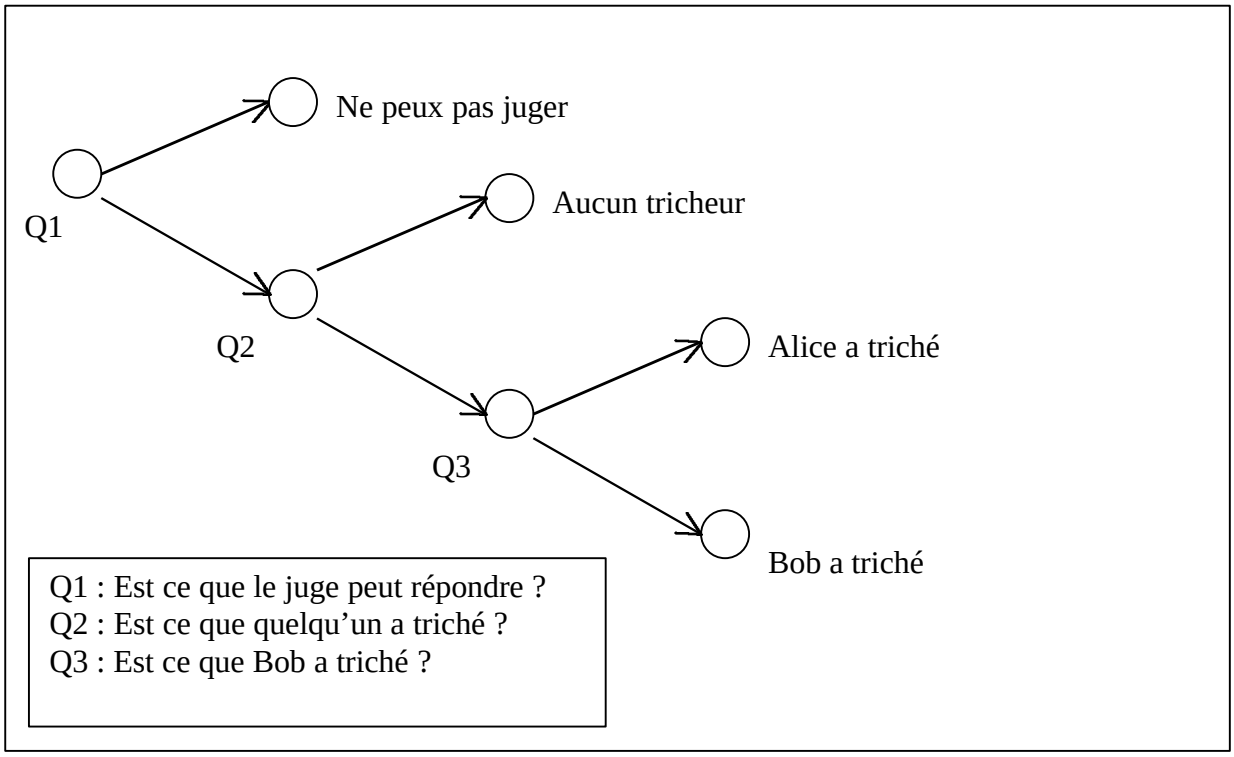

Figure 22 : Arbre décisionnel de la tricherie

Question 1 de l'arbre décisionnel : Est que le juge peut répondre ?
Pour répondre à cette question, nous devons considérer un ensemble de conditions liées et préliminaires. Tout d'abord, les deux acteurs principaux du protocole, Alice et Bob, doivent avoir fourni au juge les éléments de preuve dont il a besoin. Celui-ci va vouloir déterminer si les contractants ont signé le protocole d'accord par lequel ils s'engagent à suivre les spécifications du protocole. Le juge doit posséder aussi la spécification du protocole, les cas d'utilisation et la volonté de ne pas effectuer un deni de justice. Dans la mesure où tous ces éléments sont réunis, le protocole va commencer à examiner un début de preuve et tenter de

répondre à la question suivante : Existe-t-il une relation d'ordre locale qui débute par le mot magique init ?

Soit M l'ensemble des messages
Soit P l'ensemble des nonces regroupés dans la pile
Soit f la relation qui fait correspondre les éléments de M aux éléments de P
Montrons que f est une bijection.
F est-elle une fonction ? Pouvons nous affirmer l'assertion suivante ?
$\forall$ x E P, $\exists$ y E M : y = f(x)
L'existence de l'application EnvoisStsr nous permet d'affirmer cela. En effet, chaque message qu'un auteur veut envoyer fait l'objet d'un nonce qui lui est associé.
Par conséquent, la relation f est une fonction.
Par définition , un nonce n'est généré qu'à une occasion et une seule.
donc : $\forall$ x E P, $\forall$ y E M, f(x) = f(x') => x=x'
Par conséquent, la relation f est une injection
Les éléments de P sont générés à chaque fois qu'un message est créé, il n'existe pas d'élément créé par avance, si bien qu'il n'existe pas d'élément qui ne soit pas l'image d'un message.
donc : $\forall$ y E M, $\exists$ x E P : y = f(x)
Par conséquent, la relation f est une surjection
Nous avons démontré que f est une fonction. Nous savons aussi que f est une injection et une surjection, par conséquent, nous pouvons affirmer que f est une bijection. C'est une condition suffisante pour affirmer qu'une relation d'ordre locale existe sur le site du participant du protocole. Mais ce n'est pas une condition suffisante, il faut aussi démontrer que si un événement e précède un événement e', alors le nonce associé à e précède le nonce associé à e' dans l'ordre d'enregistrement de la pile associée à l'ensemble des nonces.
Procédons à un raisonnement par récurrence.
La pile contient le nonce init lors de l'initialisation. Le premier nonce succède à init.
Donc sucR(N0) = N1
N1 est associé à M_1.
Le nonce n est associé au message M_n et le nonce n+1 est associé au message M_{n+1}
Lors de l'envoi du message Mn+1, quand l'application EnvoiStsr va s'initialiser, le nonce associé au message Mn figure déjà dans la pile associée au nonces. Quand l'application EnvoiStsr va préparer un nonce pour le message N+1, ce nonce va être enregistrer à la suite du nonce associé au message Mn.
Par conséquent, la relation d'ordre est avérée pour le premier message. De plus si un événement Mn précède un événement Mn+1 alors le nonce associé Nn précède le nonce Nn+1. Nous pouvons ainsi affirmer que la relation d'ordre est avérée quel que soit le message considéré.

Soit M ensemble des messages
Soit P la pile des nonces
Γ, M, P |- ordre local , Δ

Par conséquent si l'application jugementStsr dispose d'une pile de nonce associée à une suite de messages, alors il devient possible de mettre en évidence une relation d'ordre local qui permet de juger et d'envisager la suite de la preuve dynamique. Il devient possible d'explorer les autres branches de l'arbre décisionnel de la tricherie.

Question 2 de l'arbre décisionnel : Est ce que quelqu'un a triché ?

S'il existe une démonstration par récurrence qui permet d'affirmer que tous les événements sont ordonnés selon une relation d'ordre causale conformément aux spécifications du protocole alors il n'y a pas de tricherie et la relation d'ordre local est aussi une relation d'ordre causale et totale.
Si la trace de A est correcte et qu'en plus la trace de B est correcte alors il n'y a pas eu de tricherie, et la relation d'ordre causale est totalement ordonnée.

Preuve :
Ce protocole gère des structures de données d'archivage.
Ce protocole gère des structures de données d'historique.
L'historique est une structure de données de type h_state [KOPETZ97]. L'historique subit l'évolution des modifications provoquées par le calcul. Chaque itération du système, correspond à une modification de l'historique. Dans notre exemple de système qui ajoute un historique à des échanges de courrier électronique, chaque message envoyé contient un historique différent. Si on considère qu'il existe n messages, et que l'historique vérifie la relation d'ordre causal à l'ordre n. Si on montre que le message existe aussi à l'ordre n+1 et qu'il est aussi conforme à la relation d'ordre causal à l'ordre n+1, alors l'historique satisfait aux propriétés des fonctions de récurrence conformément aux axiomes de Peano. Cependant, il ne faut surtout pas oublier de poser le cas de base de la récurrence. Quand n=0, il faut prévoir un historique pour fixer les bases du raisonnement, c'est la raison pour laquelle la structure de données d'archivage contient un fichier qui s'appelle init et qui contient le mot magique init. De plus, la structure de données d'historique contient aussi un fichier qui s'appelle init et qui contient le mot magique init.
Si le programme explore les données présentées par Alice en piochant alternativement dans les deux structures, et qu'il ne détecte aucune erreur, alors le juge peut estimer qu'il n'y a pas eu de tricherie, ni d'un coté ni de l'autre.
Si le programme explore la trace fournie par Bob en piochant alternativement dans les deux structures, et qu'il ne détecte aucune erreur, alors le juge peut estimer qu'il n'y a pas eu de tricherie, ni d'un coté ni de l'autre.
Si le programme explore séquentiellement les deux traces fournies par Alice et par Bob. Il commence par la trace fournie par un des acteurs, puis il termine en explorant la trace fournie par l'autre acteur. A chaque fois, il doit agir en piochant alternativement dans les deux structures (d'archive et d'historique). S'il ne détecte aucune erreur, alors le juge peut estimer qu'il n'y a pas eu de tricherie, ni d'un coté ni de l'autre.
Preuve formelle :
La preuve formelle est apportée par un raisonnement par récurrence sur l'algorithme de la sous application JugementStsr.
1.) le premier message, le cas de base de la récurrence
x=0 => e0 = init
x=1 => e = m1
1.1.) Le juge explore les données fournies par Alice. Le juge vérifie le premier message qu'Alice a envoyé à Bob.
/archive/f4 = h(/histo/init) + h(/archive/e)
/archive/f4 = h(0) + h(1)
/archive/f4 = F(h(0))
h(1) = F(h(0))
h(S(0)) = F(h(0))
Ici h(S(x)=F(h(x))

1.2.) Le juge explore les données fournies par Bob. Le juge vérifie le premier message que Bob a envoyé à Alice.
/archive/f4 = h(/histo/init) + h(/archive/e)
/archive/f4 = h(0) + h(1)
/archive/f4 = F(h(0))
h(1) = F(h(0))
h(S(0)) = F(h(0))
1.3.) Le juge explore les données fournies par Alice. Le juge vérifie le premier message qu'Alice a obtenu de Bob.
/histo/f4 = h(/archive/init) + h(/histo/e)
/histo/f4 = h(0) + h(1)
h(1) = F(h(0))
h(S(0)) = F(h(0))
1.4.) Le juge explore les données fournies par Bob. Le juge vérifie le premier message que Bob a obtenu d'Alice.
/histo/f4 = h(/archive/init) + h(/histo/e)
/histo/f4 = h(0) + h(1)
h(1) = F(h(0))
h(S(0)) = F(h(0))
Nous avons donc démontré que quelque soit le cas considéré, le cas de base de la récurrence est avéré. La fonction de hachage appliquée au successeur de l'événement initial est exprimable à l'aide de cette fonction de hachage, d'une fonction que nous avons noté F et de l'événement initial. Si x=0 alors h(S(0)) = F(h(0))

2.) Démonstration par récurrence concernant le nième message.
x=n-1 => d=Mn-1
x=n => e = Mn
x=n+1 => e'=Mn+1
2.1.) Le juge explore les données fournies par Alice. Le juge vérifie le nième message qu'Alice a envoyé à Bob.
/archive/f4 = h(/histo/Mn-1) + h(/archive/Mn)
/archive/f4 = h(n-1) + h(n)
h(S(n-1)) = h(n-1) + h(n)
h(S(n-1)) = F(h(n-1))
La propriété est vraie à l'ordre n-1, vérifions la à l'ordre n.
2.2.) Le juge explore les données fournies par Alice. Le juge vérifie le message suivant. C'est un message qu'Alice a obtenu de Bob
/histo/f4 = h(/archive/Mn-1) + h(/histo/Mn)
/histo/f4 = h(n-1) + h(n)
h(S(n)) = F(h(n))
2.3.) Le juge explore les données fournies par Bob. Le juge vérifie le nième message que Bob a envoyé à Alice.
/archive/f4 = h(/histo/Mn-1) + h(/archive/Mn)
/archive/f4 = h(n-1) + h(n)
h(S(n-1)) = h(n-1) + h(n)
h(S(n-1)) = F(h(n-1))
La propriété est vraie à l'ordre n-1, vérifions la à l'ordre n
2.4.) Le juge explore les données fournies par Bob. Le juge vérifie le nième message que Bob a obtenu de Alice
/histo/f4 = h(/archive/Mn-1) + h(/histo/Mn)

/histo/f4 = h(n-1) + h(n)
h(S(n)) = F(h(n))

Terminons la démonstration par récurrence. En ce qui concerne le paragraphe dédié aux messages indicés par N. Quelque soit le message n, quelque soit l'entité du protocole, si x=n alors h(S(n)) = F(h(n))
La fonction bijective F est donc avérée si x=0. De plus non avons démontré que si nous supposons qu'elle est vraie pour l'itération x = n, alors la fonction est vraie pour x = n+1.
Par conséquent, nous pouvons conclure en affirmant que la fonction est vraie quelque soit le message.
$\forall x$ (x E N $\rightarrow$ h(S(x)) = F(h(x))
h(S(0)) = F(h(0))
Nous avons donc démontré qu'il existe une fonction bijective qui à chacun des événements associe une structure d'historique et une seule (soit archive, soit histo). Si la structure est conforme aux spécifications du protocole, personne n'a triché, sinon quelqu'un a triché. Nous pouvons continuer à explorer l'arbre de preuve dynamique et nous intéresser à la question 3 de l'arbre décisionnel.

Question 3 de l'arbre décisionnel : Est ce que Bob a triché ?

3.1 Si le juge utilise la trace de Bob,
3.1.1 Si une erreur est détectée dans les calculs effectués suite au choix relatif au message obtenu, la structure de données h_histo contient une erreur, alors Bob n'a pas triché, et Alice a triché lors de l'envoi du message.
3.1.2 Si une erreur est détectée dans les calculs effectués suite au choix relatif au message envoyé, alors la structure de données h_archive contient une erreur, par conséquent, Bob a triché lors de l'envoi du message, et Alice n'a pas triché
3.2 Si le juge utilise la trace d'Alice
3.2.1 Si erreur détecté dans choix relatif à message obtenu, la structure de données h_histo contient une erreur, alors Bob a triché, et Alice n'a pas triché.
3.2.2 Si erreur détectée dans choix relatif à message envoyé la structure de données h_archive contient une erreur, alors Bob n'a pas triché, et Alice a triché

Alice et Bob ont tous les deux un comportement byzantin. La règle des 30 % de byzantin est considérée comme seuil justifiant de la caractérisation d'un groupe de participants byzantin dans son ensemble. Nous avons démontré par syllogisme la preuve formelle suivante :
$\forall$x: (U(x) -> B(x)) et U(Bob) |- B(Bob)
Si tous les utilisateurs de mail sont byzantins et si Bob utilise un mail, alors Bob est byzantin.
Une personne dont le comportement fut byzantin une fois, est considérée comme faisant parti du groupe des personnes à caractère byzantin pour toujours.
Ainsi, il est tout à fait envisageable de confronter une attaque du protocole de sécurisation du protocole de communication qui existe par ailleurs. Et cette attaque interviendrait de l'intérieur du système car elle serait le fait de l'une des entités. Mais les preuves formelles que nous venons de considérer, nous confortent dans l'opinion que la sous-application JugementStsr permet de diagnostiquer une erreur qui surviendrait dans le déroulement du protocole.
La preuve forte est assurée.

2.5.11.1.3 L'oracle

Considérons maintenant la deuxième preuve dynamique. Dans un deuxième temps, nous allons nous intéresser à l'arbre de preuve dynamique qui englobe l'ensemble des solutions relatives au principe de l'oracle. Etant donné deux événements, l'oracle est-il en mesure de les classer ?

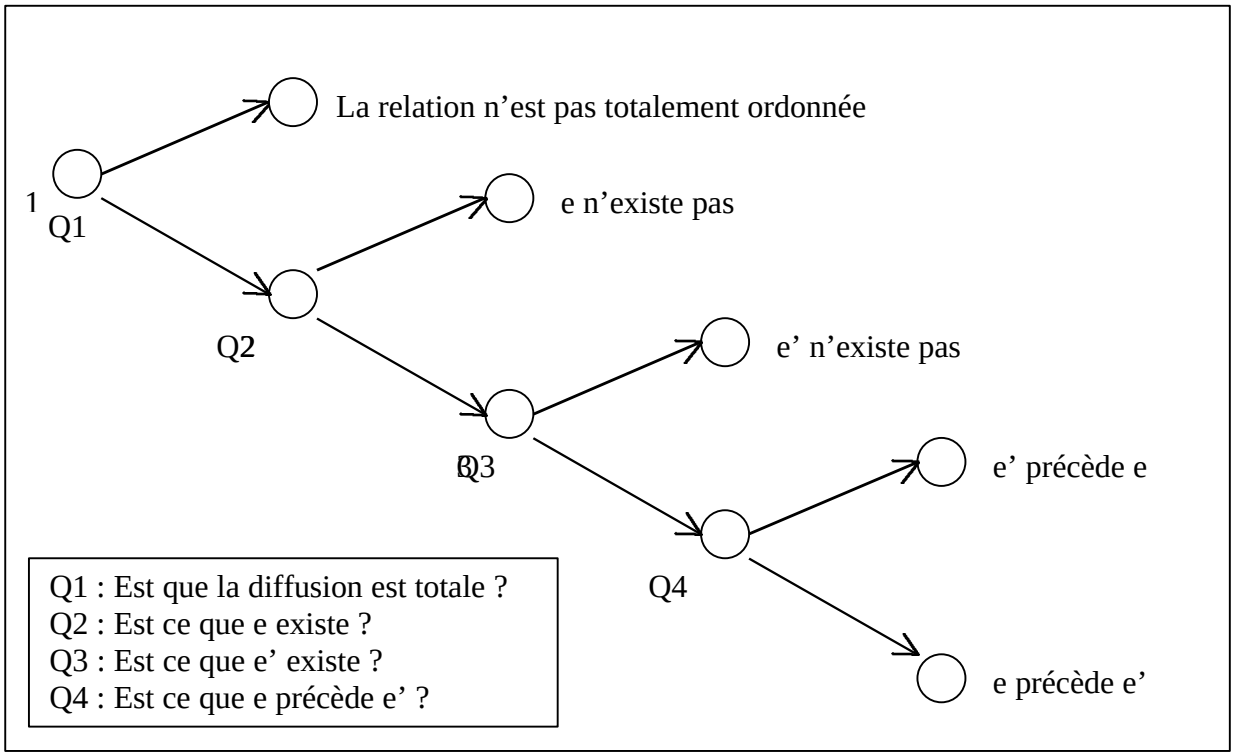

Figure 23 : Arbre décisionnel du principe de l'oracle

Considérons l'arbre décisionnel hiérarchique constructible à partir des questions suivantes :

1. Est que la diffusion est totale ?
2. Est ce que e existe ?
3. Est ce que e' existe ?
4. Est ce que e précède e' ?

Question 1 de l'arbre de l'oracle : Est que la diffusion est totale?

Considérons la question 1. et envisageons plusieurs cas :

1.1 La relation d'ordre qui classe les événements est une relation d'ordre totale dans la mesure où la pile des nonces est unique et évolue en même temps que la création d'un nouveau message. Nous avons vu que la relation qui à chaque élément de l'ensemble des nonces est une fonction bijective.

1.2 Si Oscar supprime un nonce dans la pile, alors cette relation sera toujours une fonction, puisque chaque élément de l'ensemble de départ aura une image dans l'ensemble d'arrivée. Nonobstant, l'action d'Oscar nous permet d'affirmer que dans l'ensemble des messages, un message ne sera pas l'image d'un nonce, cette condition est suffisante pour affirmer que la fonction n'est pas une surjection. La fonction n'est pas une bijection et la

relation de causalité n'est pas une relation d'ordre locale et donc f n'est pas une relation de diffusion totale.

Si le cas 1.1 est retenu, nous pouvons envisager dynamiquement de continuer d'explorer l'arbre décisionnel dédié aux tricheurs.

Question 2 de l'arbre de l'oracle : Est que e existe ?

Considérons la question 2 puisque nous savons que la relation d'ordre qui relie les événements entre eux est une relation d'ordre locale et totale.

Nous savons qu'il existe une fonction mathématique bijective qui relie l'ensemble des messages et l'ensemble des nonces. Le protocole JugementStsr peut déterminer l'existence éventuelle d'un message, il suffit pour cela d'indiquer le numéro de nonce correspondant au message. Si le nonce apparaît dans la pile dédiée à l'enregistrement des nonces, alors le protocole indiquera que le nonce a été trouvé. Ou bien si le nonce est inconnu, le protocole indiquera que le nonce n'a pas été trouvé.

Question 3 de l'arbre de l'oracle : Est que e' existe ?

La réponse est triviale pour qui a lu le paragraphe de la question 2.

Question 4 de l'arbre de l'oracle : Est que e est avant e'?

Si e existe et si e' existe, alors il est facile de déterminer lequel apparaît le premier dans la pile. Le plus proche de init est avant l'autre.

2.5.11.1.4 Conclusion

En conclusion de ce chapitre, nous pouvons affirmer que l'arbre de preuve dynamique de la tricherie éventuelle nous permet de trancher et d'assurer que la relation d'ordre totale étant établie et vérifiée dans son ensemble, il est tout à fait envisageable et possible d'identifier le tricheur éventuel. Celui-ci n'est pas en mesure de prouver qu'il n'a pas triché.

L'arbre de preuve dynamique de l'oracle nous permet d'affirmer que la relation d'ordre totale étant établie et vérifiée dans son ensemble, il est tout à fait envisageable et possible de classer deux événements distincts dans la mesure où ces événements appartiennent à l'ensemble des événements identifiés.

2.6 Spécification détaillée du logiciel

2.6.1 Langage

Le choix du langage de développement qui s'est imposé est celui de la machine virtuelle java en version 1.3. Les critères de ce choix seront discutés au paragraphe [3.6].

2.6.2 Interfaces

Le choix des interfaces qui s'est imposé est celui des fichiers plats. Chaque utilisateur possède dans son compte informatique les répertoires : courrier, archive et histo.

2.6.2.1 ~/setsar/courrier

Le répertoire ~/setsar/courrier est une interface temporaire.
Ce répertoire contient le fichier que l'auteur écrit avant de le confier à l'application EnvoiStsr.
Ce répertoire est utilisé par l'application RecevoirStsr pour obtenir les fichiers afin de les soumettre à un traitement.
Ce répertoire contient le fichier que le destinataire obtient de la part du logiciel de mailing, concrètement, le destinataire sauvegarde dans cette interface temporaire les fichiers que le logiciel de mailing lui a fait parvenir.

2.6.2.2 ~/setsar/archive

Le répertoire ~/setsar/archive contient les fichiers générés par l'application de EnvoiStsr. Ces fichiers sont ensuite envoyés par l'auteur à l'attention du destinataire.
Ce répertoire contient le fichier archi.dat dans lequel est enregistré le nonce généré par l'application EnvoiStsr et aussi par l'application RecevoirStsr.
La suite des nombres de type nonce est le garant de l'ordre local.
L'auteur possède un répertoire ~/setsar/archive qui contient les messages au moment actuel, quand il les génère.
Le destinataire possède un répertoire ~/setsar/archive qui contient les messages qu'il a générés lui-même au tour précédent.

2.6.2.3 ~/setsar/histo

Le répertoire ~/setsar/histo contient les fichiers obtenus par le destinataire. C'est la sous-application RecevoirSTSR qui fournit les fichiers à ce répertoire.

L'auteur possède un répertoire ~/histo qui contient les messages que le destinataire a générés précédemment au moyen de la sous-application EnvoiSTSR.
Le destinataire possède un répertoire ~/histo qui contient les messages que l'auteur a généré précédemment au moyen de la sous-application EnvoiSTSR.

2.6.2.4 Le pivot de l'application SETSAR

Chaque acteur, auteur ou destinataire possède un fichier ~/archive/archive.dat qui est organisé selon le modèle d'une pile LIFO, Last In First Out, Premier Entré Dernier Sorti. Cet objet persistant contient la liste des nonces, numéros d'identifiant.

Cet objet est mis à jour par la sous-application EnvoiSTSR
Cet objet est mis à jour par la sous-application RecevoirSTSR
Cet objet contient init au démarrage de l'application.

2.6.3 Diagramme des classes de la sous-application EnvoiStsr

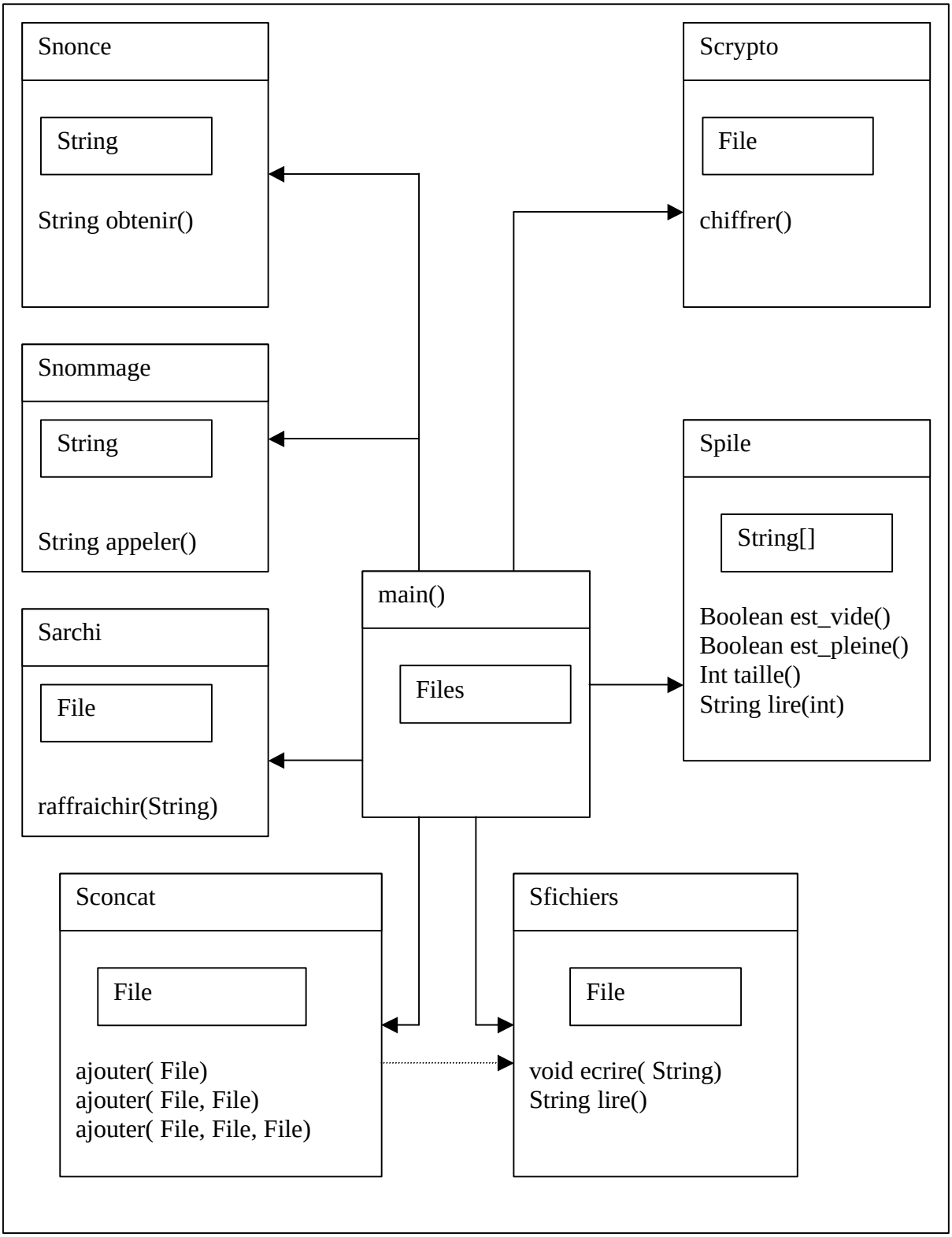

Figure 24 : Diagramme des classes de la sous-application EnvoiStsr

2.6.4 Composants de la sous-application EnvoiStsr

L'application ENVOISTSR est divisée en plusieurs composants qui sont les suivants : Sarchi, Sconcat, Sfichiers, Skrypto, Snommage, Spile.

2.6.4.1 Son composant Sarchi

Son composant Sarchi ajoute le nonce à la suite de la structure archi.dat

2.6.4.1.1 Composant applicatif Sarchi : le modèle structurel

Les attributs identifiés sont :
une chaîne de caractère qui identifie de manière unique un fichier.

2.6.4.1.2 Composant applicatif Sarchi : signature du type abstrait de données

type : Sarchi
utilise : String, StringBuffer, File
opérations :
construire : String -> Sarchi
construire : String * String -> Sarchi
construire : File * String Sarchi
rafraichir : String -> Sarchi

2.6.4.1.3 Composant applicatif Sarchi : le modèle dynamique

nom relatif d'instance -> nom de fichier unique

2.6.4.2 Son composant Sconcat

Son composant Sconcat sert à concaténer deux fichiers et mettre le résultat dans le fichier qui est l'attribut de la classe Sconcat.
note : Sconcat hérite de la classe FILE, par conséquent, tous les composants de la classe FILE existent et sont disponibles pour toutes les instances de la classe Sconcat. Les composants du modèle structurel, du type abstrait de données et du modèle dynamique de la classe File appartiennent automatiquement à la classe Sconcat.

2.6.4.2.1 Composant applicatif Sconcat : le modèle structurel

Les attributs identifiés sont :
une chaîne de caractère qui identifie de manière unique un fichier.

2.6.4.2.2 Composant applicatif Sconcat : signature du type abstrait de données

type : Sconcat
utilise : String, StringBuffer, String, File
opérations :
construire : -> Sconcat
construire : String -> Sconcat
construire : String * String -> Sconcat
ajouter : File -> Sconcat
ajouter : File * File -> Sconcat
ajouter : File * File * File -> Sconcat

2.6.4.2.3 Composant applicatif Sconcat : le modèle dynamique

nom relatif d'instance -> nom de fichier unique

2.6.4.3 Son composant Sfichiers

note : Sfichiers hérite de la classe FILE, par conséquent, tous les composants de la classe FILE existent et sont disponibles pour toutes les instances de la classe Sfichiers. Les composants du modèle structurel, du type abstrait de données et du modèle dynamique de la classe File appartiennent automatiquement à la classe Sfichiers.
Son composant Sfichiers sert à lire et à écrire dans un fichier.

2.6.4.3.1 Composant applicatif Sfichiers : le modèle structurel

Les attributs identifiés sont :
une chaîne de caractère qui identifie de manière unique un fichier..

2.6.4.3.2 Composant applicatif Sfichiers : signature du type abstrait de données

type : Sfichiers
utilise : String, StringBuffer, String, File
opérations :
construire : String -> Sfichiers
construire : String * String -> Sfichiers
construire : File * String -> Sfichiers
lire_fichier : Sfichiers -> String
ecrire_fichier : String -> Sfichiers

2.6.4.3.3 Composant applicatif Sfichiers : le modèle dynamique

nom relatif d'instance -> nom de fichier unique

2.6.4.4 Son composant Skrypto

Son composant Skrypto sert à chiffrer le contenu d’un fichier et à mettre le résultat dans le fichier qui est l’attribut de la classe Skrypto.
note : Skrypto hérite de la classe FILE, par conséquent, tous les composants de la classe FILE existent et sont disponibles pour toutes les instances de la classe Skrypto. Les composants du modèle structurel, du type abstrait de données et du modèle dynamique de la classe File appartiennent automatiquement à la classe Skrypto.

2.6.4.4.1 Composant applicatif Skrypto : le modèle structurel

Les attributs identifiés sont :

2.6.4.4.2 Composant applicatif Skrypto : signature du type abstrait de données

type : Skrypto
utilise : String, File, StringBuffer
opérations :
construire : String -> Skrypto
construire : String * String -> Skrypto
construire : File * String -> Skrypto
chiffrer_reduire : File -> Skrypto

2.6.4.4.3 Composant applicatif Skrypto : le modèle dynamique

nom relatif d'instance -> nom de fichier unique

2.6.4.5 Son composant Snommage

Son composant Snommage permet de créer le nom d'un fichier à partir de paramètres utiles pour l'application : le type, le nonce.

2.6.4.5.1 Composant applicatif Snommage : le modèle structurel

Les attributs identifiés sont :

- le nom du fichier d'instance;
- le nom du fichier temporaire.

2.6.4.5.2 Composant applicatif Snommage : signature du type abstrait de données

type : Snommage
utilise : StringBuffer, String
opérations :
construire : -> Snommage
appeler : Snommage -> String

2.6.4.5.3 Composant applicatif Snommage : le modèle dynamique

nom relatif d'instance -> nom de fichier unique

2.6.4.6 Son composant Spile

2.6.4.6.1 Composant applicatif Spile : le modèle structurel

Les attributs identifiés sont :
un tableau d'entiers qui regroupe tous les nonces.

2.6.4.6.2 Composant applicatif Spile : signature du type abstrait de données

type : Spile
utilise :
Tableau d'entiers, Entier, Fichier, String, StringBuffer, Boolean
opérations :
construire : -> Spile
construire : Entier * Fichier -> Spile
estVide : Spile -> Boolean
estPleine : Spile -> Boolean
taille : Spile -> Entier
lire : Entier -> String

2.6.5 Diagramme des classes de la sous-application RecevoirStsr

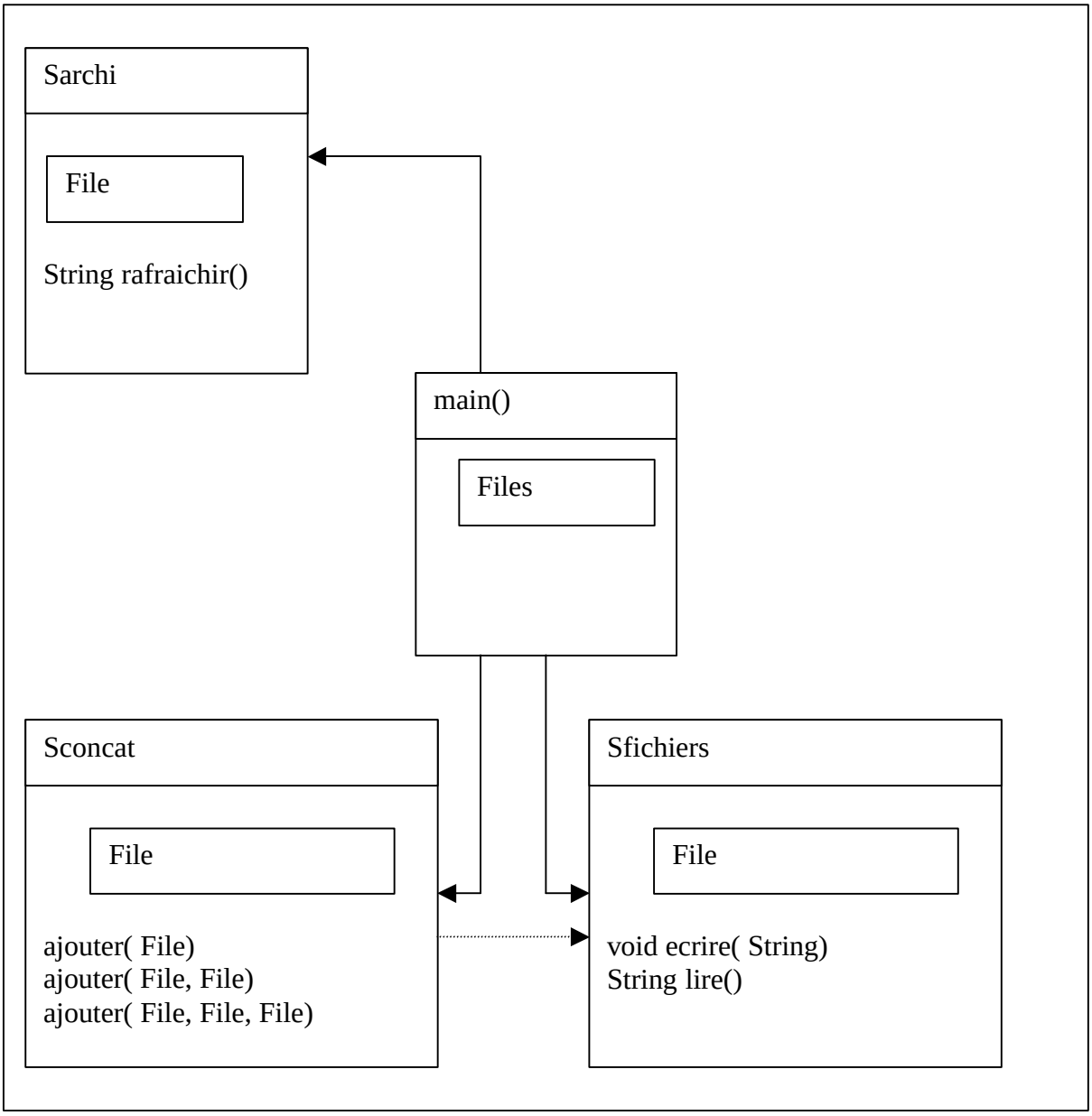

Figure 25 : Diagramme des classes de la sous-application RecevoirStsr

2.6.6 Composants de la sous-application RecevoirStsr

Les composants de la sous-application RecevoirStsr sont :

Sarchi, Sconcat, Sfichiers

2.6.6.1 Son composant Sarchi

Son composant Sarchi ajoute le nonce à la suite de la structure archi.dat

2.6.6.1.1 Composant applicatif Sarchi : le modèle structurel

Les attributs identifiés sont :

une chaîne de caractère qui identifie de manière unique un fichier.

2.6.6.1.2 Composant applicatif Sarchi : signature du type abstrait de données

type : Sarchi
utilise : String, StringBuffer, File
opérations :
construire : String -> Sarchi
construire : String * String -> Sarchi
construire : File * String -> Sarchi
rafraichir : String -> Sarchi

2.6.6.1.3 Composant applicatif Sarchi : le modèle dynamique

nom relatif d'instance -> nom de fichier unique

2.6.6.2 Son composant Sconcat

Son composant Sconcat sert à concaténer deux fichiers et mettre le résultat dans le fichier qui est l'attribut de la classe Sconcat.

2.6.6.2.1 Composant applicatif Sconcat : le modèle structurel

Les attributs identifiés sont :
une chaîne de caractère qui identifie de manière unique un fichier.

2.6.6.2.2 Composant applicatif Sconcat : signature du type abstrait de données

type : Sconcat
utilise : String, StringBuffer, String, File
opérations :
construire : -> Sconcat
construire : String -> Sconcat
construire : String * String -> Sconcat
ajouter : File -> Sconcat
ajouter : File * File -> Sconcat
ajouter : File * File * File -> Sconcat

2.6.6.2.3 Composant applicatif Sconcat : le modèle dynamique

nom relatif d'instance -> nom de fichier unique

2.6.6.3 Son composant Sfichiers

Son composant Sfichiers sert à lire et à écrire dans un fichier.

2.6.6.3.1 Composant applicatif Sfichiers : le modèle structurel

Les attributs identifiés sont :
une chaîne de caractère qui identifie de manière unique un fichier.

2.6.6.3.2 Composant applicatif Sfichiers : signature du type abstrait de données

type : Sfichiers
utilise : String, StringBuffer, String, File
opérations :
construire : String -> Sfichiers
construire : String * String -> Sfichiers
construire : File * String -> Sfichiers
lire_fichier : Sfichiers -> String
ecrire_fichier : String -> Sfichiers

2.6.7 Diagramme des classes de la sous-application VerifStsr

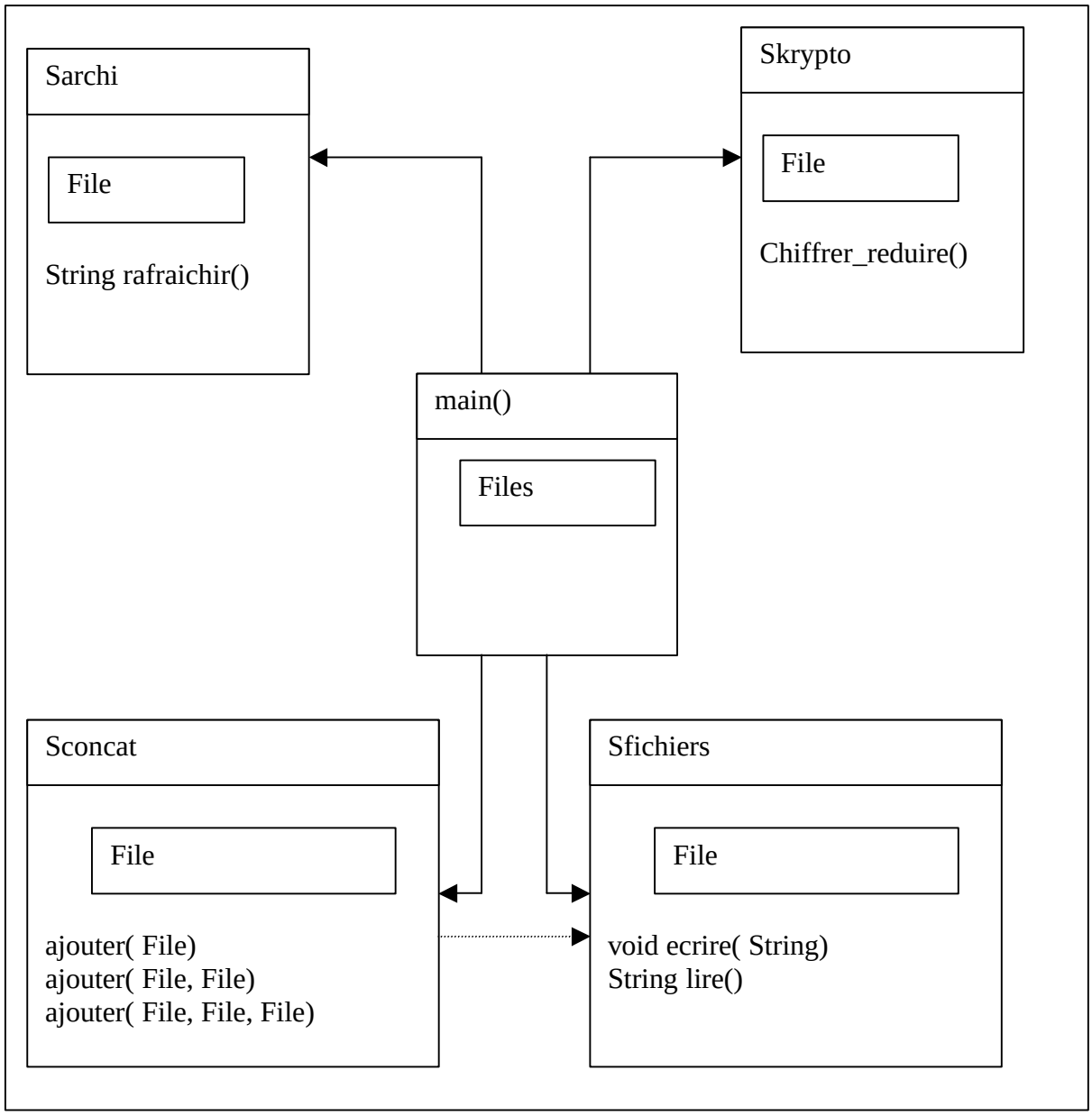

Figure 26 : Diagramme des classes de la sous-application VerifStsr

2.6.8 Composants de la sous-application VerifStsr

Les composants de la sous-application VerifStsr sont :
Scompar, Sconcat, Sfichiers et Skrypto.

2.6.8.1 Son composant Scompar

Son composant Scompar sert à comparer deux fichiers.

2.6.8.1.1 Composant applicatif Scompar : le modèle structurel

Les attributs identifiés sont :

une chaîne de caractère qui identifie de manière unique un fichier.

2.6.8.1.2 Composant applicatif Scompar : signature du type abstrait de données

type : Scompar
utilise : String, StringBuffer, File, Boolean
opérations :

construire : String -> Scompar
construire : String * String -> Scompar
construire : File * String -> Scompar
comparer : File -> Boolean

2.6.8.1.3 Composant applicatif Scompar : le modèle dynamique

nom relatif d'instance -> nom de fichier unique

2.6.8.2 Son composant Sconcat

Son composant Sconcat sert à concaténer deux fichiers et mettre le résultat dans le fichier qui est l'attribut de la classe Sconcat.

2.6.8.2.1 Composant applicatif Sconcat : le modèle structurel

Les attributs identifiés sont :

une chaîne de caractère qui identifie de manière unique un fichier.

2.6.8.2.2 Composant applicatif Sconcat : signature du type abstrait de données

type : Sconcat
utilise : String, StringBuffer, String, File
opérations :

construire : -> Sconcat
construire : String -> Sconcat
construire : String * String -> Sconcat
ajouter : File -> Sconcat
ajouter : File * File -> Sconcat
ajouter : File * File * File -> Sconcat

2.6.8.2.3 Composant applicatif Sconcat : le modèle dynamique

nom relatif d'instance -> nom de fichier unique

2.6.8.3 Son composant Sfichiers

Son composant Sfichiers sert à lire et à écrire dans un fichier.

2.6.8.3.1 Composant applicatif Sfichiers : le modèle structurel

Les attributs identifiés sont :

une chaîne de caractère qui identifie de manière unique un fichier.

2.6.8.3.2 Composant applicatif Sfichiers : signature du type abstrait de données

type : Sfichiers
utilise : String, StringBuffer, String, File
opérations :

construire : String -> Sfichiers
construire : String * String -> Sfichiers
construire : File * String -> Sfichiers
lire_fichier : Sfichiers -> String
ecrire_fichier : String -> Sfichiers

2.6.8.3.3 Composant applicatif Sfichiers : le modèle dynamique

nom relatif d'instance -> nom de fichier unique

2.6.8.4 Son composant Skrypto

Son composant Skrypto sert à chiffrer le contenu d'un fichier et à mettre le résultat dans le fichier qui est l'attribut de la classe Skrypto.
note : Skrypto hérite de la classe FILE, par conséquent, tous les composants de la classe FILE existent et sont disponibles pour toutes les instances de la classe Skrypto. Les composants du modèle structurel, du type abstrait de données et du modèle dynamique de la classe File appartiennent automatiquement à la classe Skrypto.

2.6.8.4.1 Composant applicatif Skrypto : le modèle structurel

Les attributs identifiés sont :

une chaîne de caractère qui identifie de manière unique un fichier..

2.6.8.4.2 Composant applicatif Skrypto : signature du type abstrait de données

type : Skrypto
utilise : String, File, StringBuffer
opérations :

construire : String -> Skrypto
construire : String * String -> Skrypto
construire : File * String -> Skrypto
chiffrer_reduire : File -> Skrypto

2.6.8.4.3 Composant applicatif Skrypto : le modèle dynamique

nom relatif d'instance -> nom de fichier unique

2.6.9 Diagramme des classes de la sous-application JugementStsr

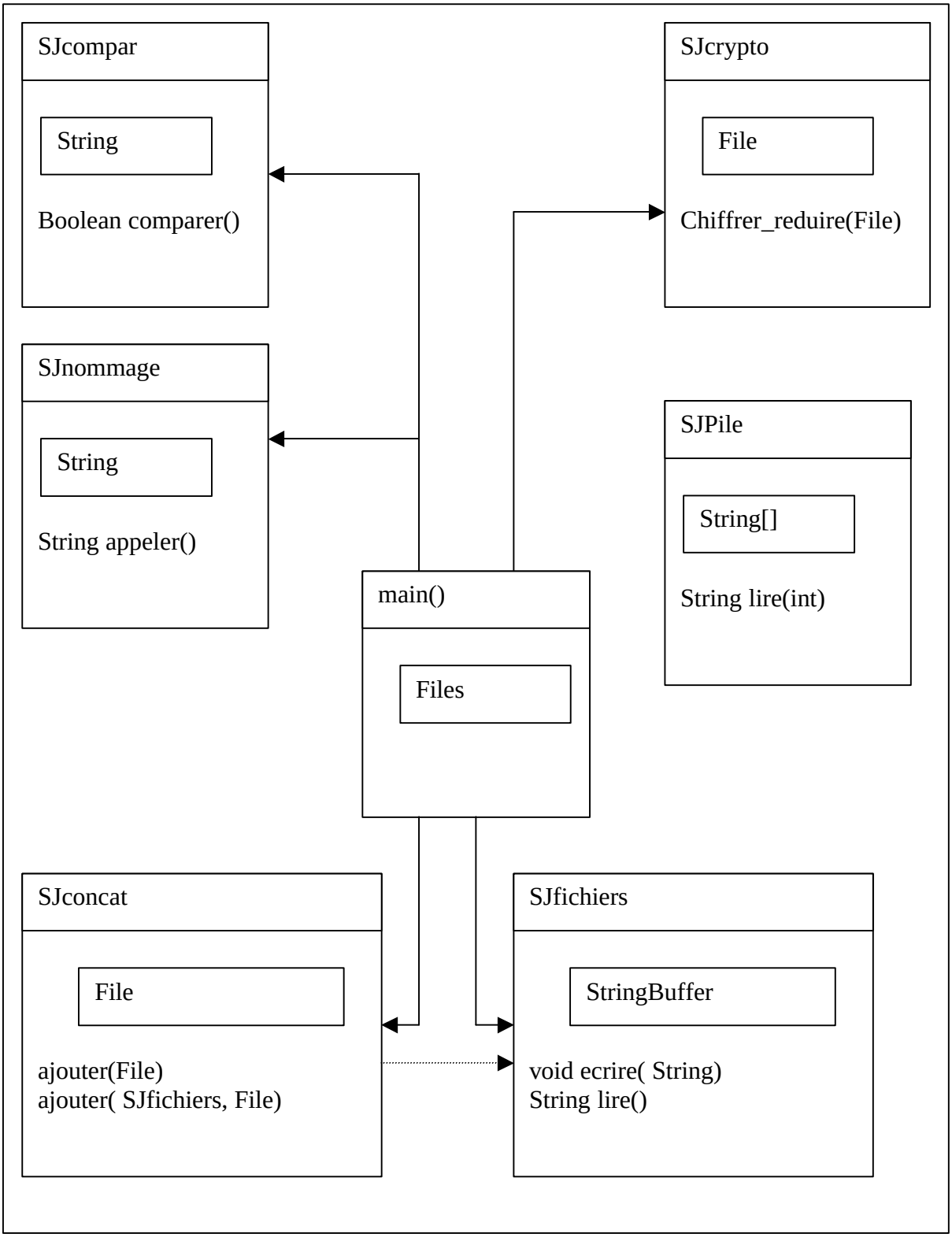

Figure 27 : Diagramme des classes de la sous-application JugementStsr

2.6.10 Composants de la sous-application JugementStsr

Cette sous-application est complètement distincte des trois autres. Les classes utilisées par JugementSTSR sont différentes des classes utilisées par EnvoiSTSR, RecevoirSTSR et VerifSTSR. Les noms sont différents. Le code est différent. L'approche de désignation des fichiers en utilisant les noms absolus (à partir de la racine du file system) a été préféré à la désignation des fichiers par nom relatif utilisée pour les sous-applications utilisées par les acteurs principaux de l'application SETSAR.

2.6.10.1 Son composant SJcompar

Son composant SJcompar sert à comparer deux fichiers.

2.6.10.1.1 Composant applicatif SJcompar : le modèle structurel

Les attributs identifiés sont :

Une chaîne de caractère qui identifie de manière unique un fichier..

2.6.10.1.2 Composant applicatif SJcompar : signature du type abstrait de données

type : SJcompar
utilise : String, StringBuffer, File, Boolean
opérations :
construire : String -> SJcompar
construire : String * String -> SJcompar
construire : File * String -> SJcompar
comparer : File -> Boolean

2.6.10.1.3 Composant applicatif SJcompar : le modèle dynamique

nom relatif d'instance -> nom de fichier unique

2.6.10.2 Son composant SJconcat

Son composant SJconcat sert à concaténer deux fichiers et mettre le résultat dans le fichier qui est l'attribut de la classe SJconcat.

2.6.10.2.1 Composant applicatif SJconcat : le modèle structurel

Les attributs identifiés sont :

Une chaîne de caractère qui identifie de manière unique un fichier..

2.6.10.2.2 Composant applicatif SJconcat : signature du type abstrait de données

type : SJconcat
utilise : String, StringBuffer, File
opérations :
construire : String -> SJconcat
construire : String * String -> SJconcat
construire : File * String -> SJconcat
ajouter : File -> SJconcat
ajouter : File * File -> SJconcat
ajouter : File * File * File -> SJconcat

2.6.10.2.3 Composant applicatif SJconcat : le modèle dynamique

nom relatif d'instance -> nom de fichier unique

2.6.10.3 Son composant SJfichiers

Son composant SJfichiers sert à lire et à écrire dans un fichier.
note : SJfichiers hérite de la classe FILE, par conséquent, tous les composants de la classe FILE existent et sont disponibles pour toutes les instances de la classe SJfichiers. Les composants du modèle structurel, du type abstrait de données et du modèle dynamique de la classe File appartiennent automatiquement à la classe SJfichiers.

2.6.10.3.1 Composant applicatif SJfichiers : le modèle structurel

Les attributs identifiés sont :

Une chaîne de caractère qui identifie de manière unique un fichier..

2.6.10.3.2 Composant applicatif SJfichiers : signature du type abstrait de données

type : SJfichiers
utilise : String, StringBuffer File
opérations :
construire : String -> SJfichiers
construire : String * String -> SJfichiers
construire : File * String -> SJfichiers
lire_fichier : SJfichiers -> String
ecrire_fichier String -> SJfichiers

2.6.10.3.3 Composant applicatif SJfichiers : le modèle dynamique

nom relatif d'instance -> nom de fichier unique

2.6.10.4 Son composant SJkrypto

Son composant SJkrypto sert à chiffrer le contenu d'un fichier et à mettre le résultat dans le fichier qui est l'attribut de la classe SJkrypto.
note : SJkrypto hérite de la classe FILE, par conséquent, tous les composants de la classe FILE existent et sont disponibles pour toutes les instances de la classe SJkrypto. Les composants du modèle structurel, du type abstrait de données et du modèle dynamique de la classe File appartiennent automatiquement à la classe SJkrypto.

2.6.10.4.1 Composant applicatif SJkrypto : le modèle structurel

Les attributs identifiés sont : String, File, StringBuffer

2.6.10.4.2 Composant applicatif SJkrypto : signature du type abstrait de données

type : SJkrypto
utilise : String, File, StringBuffer
opérations :
construire : String -> SJkrypto
construire : String * String -> SJkrypto
construire : File * String -> SJkrypto
chiffrer_reduire : File -> SJkrypto

2.6.10.4.3 Composant applicatif SJkrypto : le modèle dynamique

nom relatif d'instance -> nom de fichier unique

2.6.10.5 Son composant SJnommage

2.6.10.5.1 Composant applicatif SJnommage : le modèle structurel

Les attributs identifiés sont :

Une chaîne de caractère qui identifie de manière unique un fichier..

2.6.10.5.2 Composant applicatif SJnommage : signature du type abstrait de données

type : SJnommage
utilise : StringBuffer, String
opérations :
construire : -> SJnommage
construire : Entier * String * String * String * String * String -> SJnommage
obtenir_nom : SJnommage -> String

2.6.10.5.3 Composant applicatif SJnommage : le modèle dynamique

nom relatif d'instance -> nom de fichier unique

2.6.11 Manuel de l'installateur

Ce paragraphe fournit les information nécessaires pour une installation pertinente des différentes applications qui composent l'application setsar. L'aspect de la gestion de configuration est abordé afin de disposer des éléments nécessaires pour déployer l'application SETSAR sur les postes de travail.

2.6.11.1 Utilisateur : Ingénieur système

L'ingénieur système identifie le système d'exploitation, crée les répertoires, installe les fichiers exécutables et modifie les fichiers de configuration.
L'installation de l'application SETSAR requière des compétences d'ingénieur système pour que l'application SETSAR soit installée, configurée et optimisée quel que soit le système utilisé et quel que soit l'utilisateur de l'application.

2.6.11.1.1 Arborescence initiale de l'application

L'application s'appelle setsar ce qui veut dire sécurité et temps dans les systèmes et applications répartis. Le répertoire setsar doit se trouver dans le répertoire d'accueil de l'utilisateur. Dans notre exemple, le répertoire est /home/alicenav et le nom absolu de l'application est /home/alicenav/setsar
Le répertoire setsar doit lui-même contenir quatre répertoires archive, histo, verif et courrier.
Les noms absolus des répertoires sont donc les suivants :

- /home/alicenav/setsar/archive
- /home/alicenav/setsar/histo
- /home/alicenav/setsar/verif
- /home/alicenav/setsar/courrier

2.6.11.1.2 Fichier de configuration initiale

Chaque utilisateur doit posséder un répertoire ~bin et dans ce répertoire, doit se trouver un fichier initstsr. Ce fichier doit être lancé avant de démarrer le déroulement du protocole. Il doit être lancé une seule fois. Dans notre exemple, le fichier /home/alicenav/bin/initstsr permet d'exécuter les tâches suivantes :

- Ce fichier initialise le cas de base de la récurrence.
- Ce fichier initialise le fichier archi.dat
- Ce fichier détruit les anciens fichiers inutiles

Trace LINUX :

```
alicenav@petitfour:~/bin> cat initstsr
#!/bin/sh
# ce script initialise l'appli setsar
touch /home/alicenav/setsar/archive/archi.dat
/bin/cat /dev/null > /home/alicenav/setsar/archive/archi.dat
echo "init" > /home/alicenav/setsar/archive/archi.dat
echo "init" > /home/alicenav/setsar/archive/init
/bin/rm /home/alicenav/setsar/archive/f*
/bin/rm /home/alicenav/setsar/histo/f*
```

2.6.12 Manuel de l'utilisateur

Ce paragraphe fournit les information nécessaires pour une utilisation pertinente des différentes applications qui composent l'application setsar. Du point de vue de l'utilisateur, l'application setsar se décompose en 4 sous-applications : EnvoiStsr, RecevoirStsr, VerifStsr et Jugement. Nous allons les détailler successivement.

2.6.12.1 Utilisateur 1 : l'Auteur

L'auteur utilise la sous-application EnvoiStsr
L'auteur écrit un message et le dépose dans le répertoire courrier, ensuite, il utilise la commande suivante :
java EnvoiStsr fichier_absolu_message nom_du_destinataire

Cette commande génère des fichiers qui sont enregistrés automatiquement dans le répertoire d'archive. Ces fichiers sont de la forme f2_nonceval et f4_nonceval. Le mot nonceval indique une valeur de nonce qui est générée automatiquement par le système. La valeur de nonce qui est considérée est celle qui apparaît en dernier à la fin du fichier archi.dat.

Ensuite l'auteur fait un mail en attachant les fichiers f2_nonceval et f4_nonceval qui sont dans le répertoire d'archive.
Le nonceval a changé, par conséquent le fichier archi.dat aussi. Le nonce qui vient d'être généré figure à la dernière ligne du fichier ~setsar/archive/archi.dat
Note : l'utilisateur ne devrait en aucun cas accéder directement au composant archi.dat

2.6.12.2 Utilisateur 2 : Le destinataire

Le destinataire utilise la sous-application RecevoirStsr.

Le destinataire lit son mail. Il a des messages en fichiers attachés. Il s'agit de fichier de la forme f2_nonceval et f4_nonceval. Un nombre à cinq chiffres correspond à _nonceval. Il enregistre ces fichiers attachés dans le répertoire courrier.

Il lance obligatoirement la commande suivante :
java RecevoirStsr fichier_absolu_nonce+message

Cette appli RecevoirStsr permet de mettre à jour le fichier archi.dat. En effet, le nonce qui figure en haut à gauche du fichier qu'il vient de recevoir est lu par la sous-application RecevoirStsr. Cette valeur de nonce est enregistrée à la suite du fichier ~setsar/archive/archi.dat
Cette appli RecevoirStsr permet aussi de transférer les deux fichiers obtenus par Email depuis le répertoire courrier vers le répertoire ~setsar/histo

2.6.12.3 Utilisateur 3 : Le vérificateur

Le vérificateur peut utiliser la sous-application VerifStsr
Cette appli permet de vérifier que le message reçu et son résumé associé reçu en même temps, sont conformes et correspondent à l'archivage du message f4 du tour précédent.
La syntaxe de la commande associée est la suivante :
java VerifStsr auto alice
java VerifStsr auto bob

2.6.12.4 Utilisateur4 : Le juge

Le juge utilise la sous-application Jugement
Cette sous-application permet au juge de vérifier la relation locale, la relation causale et de trancher un litige d'antériorité entre les messages.
Un certain nombre de conditions doivent être remplies et des données sont nécessaires.
Le répertoire d'accueil du Juge (sa home directory), doit contenir un répertoire pour chaque acteur qui doit être jugé. Dans notre exemple, si le répertoire du juge est /mnt/home/navarr_s et que les utilisateurs sont alicenav et bobnav, les répertoires utilisés par le juge sont les suivants :
/mnt/home/navarr_s/alicenav
/mnt/home/navarr_s/bobnav

Chaque répertoire d'utilisateur doit contenir les répertoires histo et archive. Par exemple, pour les utilisateurs alicenav et bobnav, nous obtenons les répertoires suivants :
/mnt/home/navarr_s/alicenav/archive
/mnt/home/navarr_s/alicenav/histo
/mnt/home/navarr_s/bobnav/archive
/mnt/home/navarr_s/bobnav/histo

Pour une utilisation convenable de la sous-application Jugement, il faut disposer d'outils d'enregistrement et de transfert de fichiers, les commandes tar, dump, rcp, cp par exemple.

Deux utilisations sont possibles :

- Le juge veut vérifier l'ordre local et l'ordre causal, la commande est alors la suivante : java Jugement nom_utilisateur
- Le juge veut savoir quel message a été envoyé en premier. le message est répertorié au moyen du numéro qui figure en haut à gauche de chaque message en clair reçu. Ainsi valeur1 et valeur2 sont des numéros valides. La commande devient alors la suivante : java Jugement nom_utilisateur valeur1 valeur2

2.6.13 Les versions successives du code de l'application SETSAR

SETSAR01 Version 0.1 : 2 janvier 2002

SETSAR10 Version 1.0 : 20 janvier 2001
L'appli EnvoiSTSR permet à un auteur de générer des fichiers, utilisation de MD5.
L'appli VerifSTSR permet à un destinataire de vérifier des fichiers, la propriété de non-répudiation est assurée. Les classes associées sont Scompar, Sconcat, Sfichiers, Skrypto, Snommage et Snonce.

alicenav@freesurf.fr et bobnav@freesurf.fr communiquent par mail de netscape 4.76, mais un problème de compatibilité existe entre ISO-8859 qui est le format de départ et Non-ISO extended-ASCII text qui est le format d'arrivée.
fin le 31 janvier 2001

SETSAR11 Version 1.1 : 31 janvier 2001
archi.dat est un fichier qui se trouve dans ~setsar/archive
L'appli EnvoiSTSR met à jour archi.dat, les fichiers créés sont stockés dans archive.
L'appli RecevoirSTSR met à jour archi.dat, les fichiers reçus sont stockés dans histo.
L'appli JugementSTSR scrute la totalité de la pile archi.dat, la vérification est effectuée en prenant les fichiers alternativement dans archive et dans histo; la relation causale vérifiée, valide la relation locale.
L'appli JugementSTSR fournit un Oracle, sous réserve que la relation locale soit validée, deux nonces passés en paramètre sont comparés. Les classes associées sont Sjcompar, Sjconcat, Sjfichiers, SJkrypto et Sjnommage.
Le constructeur de la classe SJnommage construit un nom absolu de fichier, 6 choix sont possibles.

alicenav@freesurf.fr et bobnav@freesurf.fr ne peuvent pas communiquer par PINE car PINE nécessite un fournisseur qui implante un protocole de type IMAP
alicenav@aol.com et bobnavar@aol.com peuvent envoyer des messages par PINE depuis petitfour mais ne peuvent pas les recevoir par PINE sur la machine petitfour.
fin le 14 février 2002

SETSAR12 Version 1.2 : 14 février 2002
Objectif : le RSA
Cette version n'est pas terminée parce que les fichiers de certification nécessaires n'ont pas été obtenus. Les fichiers obtenus ayant une durée de validité limitée à un mois, la version de SETSAR n'a pas été achevée.

SETSAR13 Version 1.3 : 15 mai 2002
La version V1.3 est une évolution de la version V1.1
La sous-application Jugement a été réécrite pour corriger un bogue mis en évidence au cours de la rédaction de la preuve dynamique. Dans cette version, le fichier qui regroupe l'ensemble des nombres de type nonce est parcouru en commençant par le nonce initial, puis de proche en proche, tant qu'il existe encore un nombre dans la structure d'archivage. De cette manière, il devient possible de mettre en évidence de façon indubitable le comportement byzantin de l'une des entités du protocole.

3 Discussion

3.1 Introduction

Le protocole de ce mémoire devrait permettre à plusieurs participants situés dans un environnement qualifié de système réparti, de coopérer en étant liés par des contraintes réciproques. L'environnement est-il vraiment coopératif ? Le protocole de ce mémoire met-il en place un mécanisme qui interdise à une des entités de pouvoir répudier des messages aussi bien par origine que par destination ? Est-il envisageable d'atteindre l'objectif recherché ? Avons nous apporté une modeste pierre à l'édifice garant d'une nouvelle et perpétuelle mémoire ?
Envisageons une critique constructive sur les différents points constitutifs du cycle de vie du logiciel SETSAR.

3.2 Réflexion sur les travaux de Reiter

Les travaux de Reiter et Li Gong [REITER99] ont été décrits au paragraphe [1.7]. La distinction entre denial et forgery, entre destruction et fabrication gagnerait en clarté si elle était expliquée par rapport à une échelle temporelle. La destruction de la causalité intervient après la construction de la causalité. La fabrication de causalité semble caractériser la préméditation dans la mesure où elle peut intervenir avant que les entités usuelles du protocole se livre à leur commerce.
Au paragraphe [1.7.2.1] , la prophylaxie par serveur de causalité ne permet pas selon nous d'opposer un schéma de preuve forte dans le cas où l'une des entités du protocole nierait avoir reçu le message que le serveur de causalité lui avoir envoyé. Selon ce schéma, l'existence d'une preuve faible est envisageable, mais l'existence de la preuve forte ne l'est pas.

3.3 Réflexion sur la conception préliminaire

Lors de la conception préliminaire, nous avons choisi l'identification par numéro aléatoire pour une occasion unique, par le nonce. Cet artifice permet de nommer et d'identifier chaque message de manière unique, le nonce est imprédictible. Ce choix permet de se prémunir de l'attaque par re-jeu pour la simple raison que l'identification des messages est imprévisible pour un attaquant. Estelle peut espionner la suite des messages, elle n'est pas en mesure de prévoir la valeur du nonce qui identifiera le prochain message. De son coté, Maurice n'est pas non plus en mesure de fabriquer un message avec un nonce identique au nonce généré sur le site de l'un des auteurs.
Le fait que les fichiers ne sont pas écrasés entre deux itérations de l'application est un point primordial qui assure la disponibilité de l'application. Les messages sont identifiés de manière unique. Le nom F2_N1 est différent du nom F2_N2. Ce point est apparu de manière essentielle lors des démonstrations relatives à la preuve.
Cependant, il pourrait arriver que le générateur de nombre aléatoire fournisse un numéro de nonce préalablement utilisé. Cette éventualité rare nous amène à proposer l'ajout d'un deuxième identifiant de message tiré cette fois-ci de manière incrémentale et donc prédictible.
La cohabitation des deux services d'identification apporte de la redondance. Les qualités et les défauts des deux services distincts se complètent.
Par conséquent, SETSAR propose donc un mécanisme qui prémunit contre les attaques de type re-jeu, mais ce mécanisme est perfectible.

3.4 Réflexion sur la preuve

Au paragraphe [2.5.1] nous avons présenté des éléments dédiés à la preuve. Ensuite, nous avons démontré un schéma de preuve statique au paragraphe[2.5.8] pour la sous-application VerifStsr. En ce qui concerne la preuve dynamique, le paragraphe [2.5.11] est dédié à la sous-application JugementStsr. Dans presque tous les cas considérés, nous avons démontré que le protocole SETSAR procure à la fois un schéma de preuve faible, et un schéma de preuve forte. Considérons de plus près le seul cas d'utilisation qui semble mettre en défaut le schéma de preuve forte. Pour cela, le point de vue du juriste va nous être très salutaire.
Si l'entité qui initie la transaction et le contrat n'est pas suivi d'effet, si le co-contractant pressenti ne répond pas au message, alors il ne peut s'agir que d'un contrat unilatéral. Certes, la volonté solitaire est efficace dans le but de forger un schéma de preuve faible. Mais, il ne s'agit que d'une pollicitation. La non-acceptation du contrat est toujours possible de la part du destinataire puisque toute offre comporte un délai raisonnable, ce laps de temps pouvant être mis à profit par le destinataire pour examiner les tenants et les aboutissants afin de déterminer le bien fondé de poursuivre la communication. Mais, pour construire un contrat d'obligations mutuelles, un accusé de réception est nécessaire, c'est la raison pour laquelle le destinataire doit envoyer ce qui pour lui est un premier message, mais qui est le deuxième message du protocole. De plus dès lors que ce deuxième message est envoyé, il existe une preuve de la volonté du co-contractant d'accepter les exigences et les spécifications liées au contrat d'obligations mutuelles. L'heure effective de l'accord contractuel est fixée par l'émission de cet accusé de réception. La jurisprudence française va dans ce sens. Certes, comme toujours, il existe des réticences qui confirment la règle. Ne serait-il pas sage de s'en tenir, une fois pour toutes, à la thèse de Pothier admise en 1932 par la chambre des requêtes : le contrat est formé par l'acceptation définitive au lieu où elle est donnée. [ROLAND99]

Notre protocole ne déroge pas à ces règles contractuelles. Le schéma de preuve formalisée en utilisant les séquents de Gentzen a mis en évidence l'existence d'une preuve faible lors de l'initialisation du protocole et d'une preuve forte dans tous les autres cas. En effet, nous avons démontré que si Alice a envoyé un message e à Bob et que si Bob répond en envoyant un message e' et un résumé h(e') qui ne tient pas compte du résumé de e, alors Bob n'est pas encore lié avec Alice par contrat.
Par conséquent, tant que le destinataire n'a pas répondu à la sollicitation de l'initiateur de la communication, le protocole n'apporte qu'un mécanisme de preuve qualifiée de preuve faible. Mais dans tous les autres cas explorés, le comportement byzantin des entités du protocole, quelque soit sa force, est inefficace pour s'opposer à la politique de sécurité mise en place. Le protocole apporte un mécanisme capable de résister et met en évidence un schéma de preuve forte.

Si le destinataire ne retire pas son message, avec le résume correspondant, le contrat synallagmatique entre lui et l'auteur est rompu. Le destinataire a mal agi mais il n'est pas en mesure de prouver avoir bien agi. Il n'est pas en mesure de rédiger un message à la place de l'auteur car le nonce est imprédictible et de plus la fonction de hachage MD5 est à sens unique et sans collision.

La sous-application JugementStsr constitue une preuve de bon fonctionnement du protocole coopératif SETSAR. La preuve dynamique de JugementStsr étant satisfaite, nous pouvons affirmer que nous avons prouvé une preuve !

3.5 Réflexion sur l'architecture

Au lieu de recevoir de manière atomique les messages f2 et f4, il semble pertinent d'envisager une solution de type suivant :
a.) l'auteur dépose f2 et f4 dans la zone temporaire de livraison
b.) le destinataire obtient f4
c.) le destinataire acquitte f4
d.) La zone franche fournit à la fois le message et l'acquittement au destinataire de manière atomique. Mais alors, la zone franche n'est plus simplement une zone de stockage temporaire où les objets peuvent être lus par tout le monde mais modifiés seulement par leur propriétaire. Le répertoire commun /tmp/LIVRAISON devient une zone sous la responsabilité d'une autorité de confiance qui transmet des acquittements et des messages à qui de droit. Mais cela ne correspond pas aux spécifications fonctionnelles de SETSAR, dans lesquelles la notion de contrat mutuel entre les acteurs est indépendante de la présence d'une autorité de confiance, hormis l'existence d'un litige contestable. D'après les spécifications fonctionnelles, le programme doit être commun aux deux acteurs principaux A et B, et ne pas nécessiter d'autre programme, sauf l'application de jugement invoquée en cas de rupture de contrat synallagmatique entre les deux acteurs principaux.
L'architecture déployée tient compte de l'énoncé des besoins et de la politique de sécurité requise. Les quatre sous-applications sont suffisamment cloisonnées, les interfaces nous semblent correctement dimensionnées et adressées pour que les entités puissent communiquer efficacement en tenant compte des obligations réciproques et mutuelles.

3.6 Réflexion sur le codage

Le langage de programmation java a été utilisé pour coder. Le nombre de lignes de code ne peut pas être comparé au nombre de lignes de code qui aurait été écrit en utilisant un autre langage de programmation parce que java étant un langage orienté objet, plusieurs classes sont appelées à partir des applications ce type de programmation minimise le nombre de lignes de code nécessaires. Suite à compilation, le bytecode généré est portable sur des systèmes d'exploitation différents de celui qui a servi lors du développement des classes. Le langage orienté objet apporte un confort de développement indéniable. Le polymorphisme, la modularité, le ramasse-miette performant et les milliers de classes prédéfinies procurent un confort au développeur. Si bien que les éditeurs de logiciel qui d'appuient sur d'autres langages (ada ou c) ne sont pas en mesure de rivaliser et de relever les défis que java est capable d'assumer. Enfin, la puissance des machines croît désormais à un rythme tel que les critiques habituelles concernant la lenteur de java seront très bientôt désuètes et hors de propos. Java n'est pas un effet de mode, Java représente une avancée technologique significative.
Les fichiers représentent une structure de données qui convient tout à fait au style de programmation orienté objet. Chaque classe possède des attributs pour identifier précisément un fichier, les méthodes fournissant des moyens fiables pour lire ou modifier le contenu du fichier. Les nombreuses classes disponibles dans la version 1.3.1 du Kit de Développement Java (JDK 1.3.1) n'ont pas toutes été utilisées, loin s'en faut. Mais, pendant la phase de développement, des classes adéquates ont permis de relever les défis imposés. Certaines classes appartenant plus particulièrement au domaine de la sécurité méritent une attention plus prononcée. MD5 est l'acronyme de Message Digest version 5. De plus ample informations sur cet algorithme à fonction de hachage sont disponibles au paragraphe [1.5.4]. L'implantation disponible dans le JDK a donné satisfaction. Il apparaît difficile d'obtenir des résumés identiques à partir de messages différents. Toutefois, si on ajoute des espaces dans un message, bien que le nouveau message soit différent du message précédent, les deux

empreintes sont identiques. Heureusement, les spécifications fonctionnelles et la politique de sécurité choisie avait fourni un artifice permettant de contrer cette imperfection.
Le nonce, généré aléatoirement à partir du paquetage mathématique du JDK est un bon moyen permettant de différencier les messages.

3.7 Réflexion sur l'heure des messages

Avant de quitter définitivement Alice et Bob, considérons une dernière fois une de leurs querelles due à leur comportement byzantin. Alice envoie un message à midi, Bob l'obtient à 13h, Alice affirme l'avoir envoyé à 11h. Effectivement, il n'y a pas de mécanisme prévu dans SETSAR pour se prémunir contre ce type de fraude, parce que rien n'est prévu pour faire de la datation en fonction du temps universel. Cette problématique de l'horodatage servant à dater les événements ne correspond pas à la problématique posée à l'issu du chapitre 2 dédié aux moyens et méthodes. Les spécifications de SETSAR ont été écrites de manière à respecter l'ordre causal qui existe entre les événements du système.

3.8 Réflexion sur le temps

Le temps englobe tout, dater les messages en fonction de l'heure universelle nous a paru difficile à sécuriser. La datation effectuée en créant une empreinte de la relation d'ordre causal nous a permis de spécifier et d'implanter un protocole tel que la propriété d'ordre ne soit répudiable ni par origine, ni par destination.

4 Conclusion

La problématique posée et les techniques cryptographiques retenues devaient permettre de construire un protocole qui assure à long terme la propriété de non répudiation des événements et des documents électroniques. Ce protocole devant être utilisable dans le contexte de la jurisprudence relative à la signature électronique et permettre la formalisation de contrats synallagmatiques sans intervention d'un tiers de confiance sauf en cas de litige.
Faisant suite à l'exploration de l'énoncé de la problématique et de l'analyse fonctionnelle. Nous avons proposé une architecture et des spécifications préliminaires. Le découpage des deux fonctions de bases suffisantes pour que les entités principales puissent communiquer et la spécification des interfaces ont procuré un socle sain et fiable constitutif d'une première couche protocolaire permettant de tracer une relation sécurisée. Ensuite, nous avons ajouté une deuxième couche protocolaire dédiée à la vérification, voire au jugement. Les points soulevés lors de la discussion étaient relatifs à la conception préliminaire, à la preuve, à l'architecture et au code java. Tous ces points ont été traités sans que des anomalies graves n'aient pu être mises en évidence. De plus, SETSAR est conforme au principe de Kerkhoff qui présume que l'adversaire connaît le message chiffré et tous les détails du protocole de communication mais qu'il ne connaît pas la clef.

En conclusion, la sécurité du protocole possède donc une propriété de non-répudiation par origine satisfaisante. Par conséquent, la répudiation par origine de la part d'une entité est non avérée. De même, la sécurité de SETSAR possède aussi une propriété de non-répudiation par destination satisfaisante. Par conséquent, la répudiation par destination de la part d'une entité est non avérée.

Le protocole SETSAR est utile, dans la mesure où il peut se greffer à de nombreux protocoles de type coopératif. Le protocole SETSAR peut servir à sécuriser un protocole de communication qui existe par ailleurs et qui nécessite la volonté de personnes distantes obligées de s'associer, de partager des ressources et de s'engager sur des délais. Ces entités doivent suivre des contraintes d'obligations mutuelles.

Souhaitons qu'il contribue à susciter des vocations et à ce qu'elles se réalisent.

5 Bibliographie

[BOURROUILH01] A. BOURROUILH, *Spécification et réalisation d'un protocole de serveur de temps pour les applications de sécurité*. Mémoire d'ingénieur IIE. 2001.

[CARREZ96] C. CARREZ, G. FLORIN, C. TOINARD. *Résultats pour la construction efficace de protocoles à diffusion qui garantissent une qualité d'ordre maximale*. 1996.

[CARREZ98] C. CARREZ, G.FLORIN, C. TOINARD. *Causally and totally ordered multicast protocols*. 1998.

[FAUSSE01] A. FAUSSE. *La signature électronique*, Dunod, 2001.

[GINGUAY81] M. GINGUAY. *Dictionnaire de l'informatique*, Masson, 1981.

[GOSCINSKI91] A. GOSCINSKI. *Distributed operating systems, Addison-Wesley Publishing company*, 1991.

[KOPETZ97] H. KOPETZ. *Real time systems*, Kluwer academic publishers, 1997.

[MENEZES96] A. MENEZES. *Handbook of applied cryptography*, CRC Press, 1996.

[MERCADAL93] B. MERCADAL. *Droit des affaires*, Lefebvre Francis, 1993.

[MULLER00] P. MULLER. *Modélisation objet avec UML*, Eyrolles, 2000.

[NATKIN02] S. NATKIN. *Les protocoles de sécurité de l'Internet*, Dunod, 2002.

[PIETTE01] T. PIETTE-COUDOL. *La signature électronique*. Litec, 2001.

[REITER99] M. REITER. *Securing causal relationship in distributed operating systems*, 1999.

[ROLAND99] H. ROLAND, L. BOYER. *Adages du droit français*, Litec, 1999.

[STERN98] J. STERN. *La science du secret*, Odile Jacob, 1998.

[STINSON96] D. STINSON. *Cryptographie*, Prentice Hall, 1996.

[TANENBAUM95] A. TANENBAUM. *Distributed operating systems*, Prentice-Hall International, 1995.

[TANENBAUM98] A. TANENBAUM. *Réseaux*, Interéditions, 1998.

6 Annexe 1 : Test fonctionnel 1

Le testeur peut générer des données en prenant alternativement les identités de alicenav, de bobnav. Les sous-applications EnvoiStsr, RecevoirStsr et VerifStsr sont testées.

INITIALISATION

1. Initialisation des application SETSAR
1.1 alicenav initialise son compte
    /home/alicenav/bin/initstsr
1.2 bobnav initialise son compte
    /home/bobnav/bin/initstsr

DEBUT DE TEST

2. alicenav veut envoyer un message à bobnav
2.1 alicenav dépose un message1 dans
    /home/alicenav/setsar/courrier/message1
2.2 alicenav envoie la commande suivante
    java EnvoiStsr /home/alicenav/setsar/courrier/message1 bob
2.3 alicenav copie les fichiers dans /tmp/livraison
    cp /home/alicenav/setsar/archive/f2_nonceval /tmp/LIVRAISON
    cp /home/alicenav/setsar/archive/f4_nonceval /tmp/LIVRAISON
  ou bien alicenav envoie les deux fichiers par mail
3. bobnav traite les messages
3.1 bobnav récupère les fichiers qui sont dans /tmp/LIVRAISON
    cd /home/bobnav/courrier; cp /tmp/LIVRAISON/* .
3.2 bobnav traite le courrier
    java RecevoirStsr /home/bobnav/setsar/courrier/f2_nonceval
3.2.1 si bobnav le souhaite, il vérifie
    java VerifStsr auto alicenav
4. alicenav vide /tmp/LIVRAISON des fichiers qui lui appartiennent
5. bobnav veut envoyer un message à alicenav
5.1 bobnav dépose un message2 dans
    /home/bobnav/setsar/courrier/message2
5.2 bobnav envoie la commande suivante
    java EnvoiStsr /home/bobnav/setsar/courrier/message2 alice
5.3 bobnav copie les fichiers dans /tmp/livraison
    cp /home/bobnav/setsar/courrier/f2_nonceval /tmp/LIVRAISON
    cp /home/bobnav/setsar/courrier/f4_nonceval /tmp/LIVRAISON
  ou bien bobnav envoie les deux fichiers par mail
6. alicenav traite les messages
6.1 alicenav récupère les fichiers qui sont dans /tmp/LIVRAISON
    cd /home/alicenav/courrier; cp /tmp/LIVRAISON/* .
6.2 alicenav traite le courrier
    java RecevoirStsr /home/alicenav/setsar/courrier/f2_nonceval
6.2.1 si alicenav le souhaite, elle vérifie
    java VerifStsr auto bobnav
7. bobnav vide /tmp/LIVRAISON des fichiers qui lui appartiennent
Tant que le testeur veut continuer à générer des données sur les sites de alicenav et de bobnav faire
    retour au 2.
fait

7 Annexe 2 : Test fonctionnel 2

Si le testeur veut tester la sous-application JugementStsr

1. alicenav sauvegarde les données de alicenav
    tar cvf alicenav_date.tar setsar11
    cp alicenav_date.tar /tmp

2. bobnav sauvegarde les données de bobnav
    tar cvf bobnav_date.tar setsar11
    cp bobnav_date.tar /tmp

3. navarr_s exploite les données
    cd
    mkdir alicenav
    mkdir bobnav
    cd JEUX_DE_TEST
    cp /tmp/alicenav_date.tar .
    cp /tmp/bobnav_date.tar .

    tar xvf alicenav_date.tar
    mv setsar11 setsar
    mv setsar ../alicenav

    tar xvf bobnav_date.tar
    mv setsar11 setsar
    mv setsar ../bobnav

    cd; cd setsar/java;

Pour tester la validité de la relation causale dans les données fournies par Alice :
    java JugementStsr auto alicenav
Pour tester la validité de la relation causale dans les données fournies par Bob :
    java JugementStsr auto bobnav
Pour consulter l'oracle, il faut disposer de deux numéros de message.
    java JugementStsr auto alicenav N1 N2
    java JugementStsr auto bobnav N1 N2

8 Annexe 3 : Le code en java