UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

LA CORRECTION DES PROTOCOLES DE NON-RÉPUDIATION

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN INFORMATIQUE

PAR

LUKE SULLIVAN

MAI 2006

UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS
Département d'informatique et d'ingénierie

Ce mémoire intitulé :

LA CORRECTION DES PROTOCOLES DE NON-RÉPUDIATION

présenté par
Luke Sullivan

pour l'obtention du grade de maître ès sciences (M.Sc.)

a été évalué par un jury composé des personnes suivantes :

Kamel Adi.....directeur de recherche

Jurek Czyzowicz.....président du jury

Mohamed Mejri.....membre du jury

Mémoire accepté le : 5 mai 2006

Je dédie ce travail à mon fils Jean-Sébastien et à ma fille Katherine qui sont des êtres d'un courage exceptionnel et qui m'inspirent à tous les instants.

Remerciements

J'aimerai prendre cette opportunité pour remercier de tout mon coeur mon superviseur Professeur Kamel Adi, pour son temps, sa patience, sa passion et ses connaissances, et sans qui ce mémoire n'aurait pu voir le jour. Je remercie aussi tous les membres du groupe de recherche LRSI pour les discussions fructueuses que nous avons eues ensemble. En particulier je remercie L. Pene, qui a démontré une immense générosité et a contribué de façon importante à ce travail. J'aimerai aussi remercier Professeur Mohamed Mejri (Université Laval) et Professeur Jurek Czyzowicz (UQO) pour le temps investi dans l'évaluation de ce document et les précieux conseils qu'ils ont fourni pour l'amélioration de celui-ci.

Résumé

Les services de non-répudiation préviennent les pertes de tout ordre qui peuvent survenir quand un des participants à un échange électronique, nie en avoir fait partie. Ces services sont essentiels pour plusieurs types d'applications de commerce électronique comme les systèmes d'encans, la vente et l'achat de biens par voie électronique, les systèmes de vote électronique, etc. Un protocole de non-répudiation offre les services de non-répudiation de réception ou d'origine en générant des preuves pour l'un ou l'autre des participants. Ces preuves qui doivent satisfaire des propriétés bien précises, peuvent alors être utilisées devant un arbitre ou juge en cas de dispute entre les protagonistes. Cependant, la complexité et la subtilité des services de non-répudiation rendent difficile la tâche de concevoir des protocoles de non-répudiation corrects.

Nous présentons une nouvelle approche qui vise la conception de protocoles de non-répudiation corrects. En particulier pour les protocoles de non-répudiation avec et sans troisième agent de confiance, nous visons dans une première phase, l'élaboration d'un ensemble de conditions syntaxiques pour caractériser une classe de protocoles de non-répudiation. Dans un second temps, nous démontrerons formellement que ces conditions sont suffisantes pour garantir la correction de cette classe de protocoles par rapport à la propriété d'équité (la plus forte propriété de non-répudiation) et la propriété de timeliness. Dans un troisième temps, nous démontrerons en utilisant une théorie de jeux que dans un protocole de non-répudiation sans troisième agent de confiance, la meilleure équité que nous pouvons espérer est une équité probabiliste.

Abstract

Non-repudiation services in an electronic network setting are designed to prevent losses that can be incurred when a party to an exchange denies having been involved in the given exchange. Such services are essential for several e-commerce applications such as online bidding, online purchasing, online voting etc. If a customer for example denies the fact that he placed an electronic bid for an article, it must be possible for the auction house to have proof that the bid was actually placed.

A non-repudiation protocol provides non-repudiation services by generating proofs of each of the party's participation in the exchange. These proofs can then be used by an impartial judge to settle any disputes. The protocol must satisfy very specific properties to generate correct proofs. The difficulty that designers of non-repudiation protocols face is the subtlety of the properties that a protocol must satisfy in order to always generate correct proofs that guarantee the non-repudiation services.

The aim of this thesis is to design correctness conditions that can be used as simple guides for the design and implementation of non-repudiation protocols. This is especially useful when a developer has other services in addition to non-repudiation to offer. There are two general cases to investigate : non-repudiation protocols with and without a Trusted Third Party. For the case with a Trusted Third Party, we develop syntactic conditions and show that they are sufficient for guaranteeing fairness and timeliness for a very specific type of Trusted Third Party. For the case without, syntactic conditions are developed under which probabilistic fairness and timeliness can be guaranteed. Finally, by modeling non-repudiation protocols as games, we use game theory results to show that the best level of fairness such protocols can achieve is probabilistic fairness.

Table des matières

Liste des tableaux

Chapitre 1

Introduction

Un protocole de sécurité a pour objectif d'offrir un service de sécurité à une communication électronique sur un réseau distribué ou Internet. Ces services sont exigés pour protéger les échanges contre les actions d'agents hostiles internes ou externes à la communication. Les services de sécurité les mieux connus et les plus étudiés sont ceux de l'authentification et de la confidentialité.

Les services de non-répudiation sont des services de sécurité moins connus mais néanmoins importants et critiques pour un grand nombre d'échanges sur un Internet et surtout pour le commerce électronique.

Une étape de communication dans un protocole d'échange électronique consiste en un échange de message entre un initiateur A et un receveur B. Cependant, si après la communication l'un des agents A ou B nie avoir participé à l'échange, l'autre agent peut subir une perte. Par exemple le message échangé peut être une signature électronique de contrat et dans ce cas la répudiation de A ou de B peut remettre en cause la validité du contrat avec toutes les conséquences négatives que cela peut engendrer. Dans ce contexte, étant donné que les agents A et B ne se font pas mutuellement confiance, A exigera que B lui envoie une preuve de réception qui prouve qu'il a bel et bien reçu le message, tandis que B exigera une preuve d'origine prouvant que le message a été envoyé par A.

Tandis qu'un protocole d'authentification protège les agents honnêtes participants aux protocoles contre un attaquant externe, les protocoles de non-répudiation protègent contre les actions malhonnêtes d'un agent participant au protocole.

Considérons par exemple un système d'élection par vote électronique, qui permet la transmission du vote d'un électeur directement au bureau de scrutin à travers l'Internet. Il y a des risques inhérents dans un tel système si l'électeur ou le bureau de scrutin répudie

la transmission du vote. En effet, les droits de l'électeur seront compromis si le bureau de vote nie avoir reçu son vote. De même, l'élection sera compromise si un électeur peut nier avoir envoyé son vote et par conséquence peut voter plusieurs fois. Pour se protéger contre de telles éventualités, l'électeur exigera un service de non-répudiation de réception, tandis que le bureau de scrutin exigera un service de non-répudiation d'origine.

Les protocoles de non-répudiation tentent de garantir les services de non-répudiation en générant des preuves. Chaque agent veut obtenir une preuve de la participation de l'autre. Dans l'exemple du vote électronique, l'électeur devra exiger une preuve que son vote a été reçu tandis que le bureau de scrutin exigera une preuve que le vote provient vraiment de l'électeur inscrit sur le bulletin. Les preuves générées doivent être d'une forme très spécifique : une preuve de non-répudiation doit permettre à un juge indépendant de décider si le récepteur a réellement reçu le message et si l'émetteur l'a réellement envoyé. Ces preuves prennent le plus souvent la forme de signatures digitales. Nous distinguons donc deux types de preuves de non-répudiation : d'origine et de réception. La première protège le receveur et la dernière l'initiateur.

Qu'un protocole satisfasse les propriétés de non-répudiation d'origine et de réception n'est pas suffisant pour garantir que les preuves seront reçues face à un agent malhonnête ou un canal de communication non fiable. Imaginons un protocole de non-répudiation conçu pour un système d'élection électronique, qui spécifie que si un électeur envoie son vote avec une preuve d'origine, alors le bureau doit lui renvoyer une preuve de réception. Un bureau de scrutin malhonnête peut tricher facilement en retenant la transmission de la preuve de réception. Il détient alors le vote et une preuve de son origine, mais l'électeur lui est perdant, puisqu'il n'a aucune preuve de la réception du vote par le bureau de scrutin.

Pour être le moindrement utile, un protocole de non-répudiation doit aussi garantir que tout agent participant au protocole ne peut tricher. Un tel protocole est dit *équitable*(fair). La propriété d'équité est essentielle pour protéger contre les actions des agents malhonnêtes, cependant, il existe d'autres propriétés qui peuvent aussi être exigées comme la propriété de *timeliness* qui exige un délais de réponse fixé ou la propriété de viabilité qui garantie aux participants du protocoles la réception des preuves. Par exemple, si un bureau de scrutin doit attendre des heures ou même des jours pour recevoir la preuve d'origine du vote, l'élection sera retardée d'autant et le système sera vu comme non pratique.

Pour assurer que les propriétés discutés précédemment soient respectées, le protocole fait souvent usage d'un troisième agent de confiance (Trusted Third Party) et noté *TTP* dans le reste du document. Le TTP ne peut favoriser un agent au dépends d'un autre et doit respecter les consignes du protocole. Il sert le plus souvent, comme base de données ou passerelle sécuritaire et toute preuve qu'il génère sera automatiquement acceptée par le juge.

Dans le cas de notre système de vote, un TTP pourrait servir de passerelle entre l'électeur et le bureau de scrutin. Ainsi, si le TTP reçoit le vote, une preuve d'origine sera générée pour le bureau de scrutin. Le TTP l'acheminera au bureau de scrutin seulement s'il reçoit une preuve que le bureau s'est engagé à l'échange.

Parce que le TTP peut être une cause importante de ralentissement, surtout s'il intervient dans chaque échange, sa participation doit préférablement être limitée. Dans la littérature, nous retrouvons une variété de type de protocoles de non-répudiation qui permettent la participation du TTP à des degrés différents.

Garantir les propriétés de non-répudiation dans un protocole n'est pas une tâche simple car il est nécessaire de prouver que les propriétés sont préservées pour toute exécution du protocole, face à un agent malhonnête par exemple ou des canaux de communications peu fiables.

Depuis les années 70, les chercheurs ont utilisé les méthodes formelles pour analyser une variété de protocoles de sécurités [22]. Ceux-ci eurent un certain succès, surtout pour la découverte de failles dans des protocoles qui ont été longtemps considérés comme corrects. Mais ces méthodes ont leurs limites. La vérification par model-checker par exemple peut mener à une simplification trop importante du protocole analysé et la complexité et la subtilité des propriétés à vérifier sont souvent difficiles à spécifier correctement. Finalement, la vérification se fait une fois le protocole conçu et le plus souvent après qu'il a été implémenté et utilisé.

Nous présentons une nouvelle approche pour garantir la correction. En s'inspirant du travail de [13], nous planifions élaborer, pour deux propriétés de non-répudiation, un nombre restreint de conditions qui garantiront qu'un protocole de non-répudiation est correct pour une propriété de non-répudiation donnée.

Nous établissons des conditions syntaxiques qu'un concepteur de protocole pourra utiliser comme guide de conception. Par exemple, une condition assurant l'équité pourra exiger qu'à des étapes précises du protocole de non-répudiation, l'initiateur doit arrêter le protocole, s'il n'a pas reçu des preuves de réception pour les messages déjà envoyés.

Nos contributions à la recherche sur les protocoles de non-répudiation incluent

- Formalisation des concepts reliés aux protocoles de non-répudiation.
- Élaboration d'un ensemble de conditions suffisantes qui garantissent les propriétés d'équité et de timeliness pour les protocoles de non-répudiation avec TTP.

- Élaboration d'un ensemble de conditions suffisantes qui garantissent les propriétés d'équité probabiliste et de timeliness pour les protocoles de non-répudiation sans TTP.
- Utilisation des concepts de la théorie de jeux pour démontrer qu'un protocole sans TTP satisfaisant nos conditions peut garantir au plus une équité probabiliste.

Le reste de ce document est organisé comme suit : dans un premier temps, nous allons introduire les algorithmes cryptographiques qui sont nécessaires pour assurer la non-répudiation. Cela sera suivi par une présentation détaillée des protocoles de non-répudiation, en soulignant la grande variété de types de protocoles qui existent et en introduisant une formalisation des concepts. Ensuite, nous allons étudier les méthodes de vérification formelle qui ont été appliquées pour vérifier leurs corrections. Finalement, nous présenterons les conditions de correction syntaxiques pour les propriétés de l'équité et le timeliness pour des protocoles avec et sans TTP. Cela sera suivi par une proposition basée sur la théorie des jeux qui démontre que le meilleur résultat possible d'un protocole de non-répudiation sans TTP est l'équité probabiliste.

Chapitre 2

État de l'art

2.1 Algorithmes cryptographiques

L'encryptage d'un message M par une clé d'encryptage k, rend M illisible, sauf pour ce qui possède la clé de décryptage k'. On parle généralement de deux types d'algorithmes d'encryptage : *à clé symétrique* et *à clé publique.*

Un algorithme d'encryptage à clé symétrique est caractérisé par le fait que $k = k'$. Le message M est encrypté par une clé k, action dénotée par $C = \{M\}_k$ et C peut être décrypté seulement par la même clé k. Un algorithme d'encryptage à clé publique quant à lui comporte deux clés. On associe une paire de clés (SK_x, PK_x) à chaque agent X, où PK_x est la *clé publique* de X et SK_x sa *clé privée.* Ainsi, si B veut encrypter un message M pour l'agent A, il appliquera la clé publique de A à M, que l'on dénote par $C = \{M\}_{PK_a}$. A déchiffrera M en appliquant sa clé privée à C, ainsi $M = \{\{M\}_{PK_a}\}_{SK_a}$.

Une *signature* digitale d'un message est une preuve de l'identité de la source d'un message. On emploie généralement les algorithmes à clé publique pour générer les signatures. Ainsi, pour générer une signature pour le message M, l'agent A appliquera sa clé privée à M, action que nous dénotons par $\{M\}_{SK_a}$. Cette signature est jointe au message M. Un agent B receveur, peut vérifier que la signature est celle de A, en y appliquant la clé publique PK_a et en comparant le résultat avec le message originale. Donc, il devra s'assurer que

$$M = \{\{M\}_{SK_a}\}_{PK_a}$$

En plus d'assurer l'identité de la source d'un message, la signature digitale assure l'intégrité du contenu du message. En effet, si un agent malhonnête modifie le message, la signature digitale ne sera plus correcte.

Il est rare que l'on signe le message M lui-même. En effet, les algorithmes d'encryptage publique sont lents, et on signe plutôt le *hash* du message, dénoté par $H(M)$. La fonction $H()$ réduit le message à une valeur de taille fixe et généralement beaucoup plus petite que M. Si la fonction $H()$ est bien choisie, il sera impossible de trouver un M' tel que $H(M) = H(M')$, ni de déduire M de son hash $H(M)$.

Pour tout algorithme à clé publique, la clé privée doit demeurer secrète, tandis que la clé publique doit être distribuée aux agents avec qui A s'attend communiquer. On remarque que la distribution des clés publiques est problématique : un agent doit avoir une garantie de la provenance de chaque clé publique dont il se servira. Pour ce faire, les clés publiques sont endossées par un *Certificate Authority* ou CA dans lequel chaque agent a confiance.

Les protocoles de non-répudiation doivent leur efficacité aux développements de la cryptographie à clé publique. En effet, les preuves générées par les protocoles de non-répudiation sont souvent basées sur des signatures digitales. Ceci permet à implanter des protocoles efficaces et simples, mais malheureusement non sans risques. Il est essentiel de comprendre les risques encourus par l'utilisation de signatures digitales comme preuves de non-répudiation. Nous rappelons certains de ces risques ci-dessous.

Supposons que la signature digitale $\{M\}_{SK_a}$ est offerte par l'agent A à un agent B, comme preuve de non-répudiation de la transmission du message M. Si B possède la clé publique PK_a, il peut vérifier la preuve en vérifiant que $M = \{\{M\}_{SK_a}\}_{PK_a}$.

Même si la condition tient, elle n'est pas suffisante pour avoir confiance en la preuve. En effet, B doit pouvoir vérifier au moins deux éléments additionnels :

- que la clé publique appartient vraiment à A, et
- que la clé privée de A n'a pas été compromise.

Les *certificats* ont été introduits pour satisfaire le premier point. Un certificat $Cert_A$ est un message contenant au minimum la clé publique de A, son nom, un identificateur unique de A, la date de création du certificat, une date d'expiration et une signature digitale du message au complet créée par un CA. Donc si B a confiance en le CA, il pourra alors avoir confiance que la clé publique appartient réellement à A.

Pour le second point, un CA peut *révoquer* un certificat, si celui-ci pense que la clé privée associée a été compromise. Le système de révocation est habituellement assez simple, souvent une liste d'identificateur de certificat signée par le CA, et rendue publique sur un serveur et continuellement mise à jours.

Notons qu'une clé privée peut être compromise de plusieurs façons. Elle peut par exemple être volée ou avoir sa valeur devinée. On ne s'attardera pas sur le vol, mais nous notons simplement qu'il existe des techniques, comme par exemple des jetons matériels sur lesquels un agent peut conserver sa clé de façon sécuritaire. Ayant accès à une signature digitale, un attaquant peut réussir à deviner la valeur de la clé privée qui l'a produite, si celle-ci est faible. Une clé est jugée faible si elle est de trop petite taille. La taille d'une clé cryptographique est mesurée en *b*its. Une clé peut être aussi jugée faible si elle a été mal générée. Une clé privée est générée à partir d'une source aléatoire, et donc meilleure est la source aléatoires, plus difficile sera la tâche de deviner la clé. Malheureusement, il est souvent difficile de concevoir des sources aléatoires assez riches en entropies.

Finalement, les fonctions de hachage H() peuvent être aussi une source de problèmes. Comme mentionnée ci haut, puisque la fonction qui produit une signature digitale est lente, il est souvent le cas que c'est $H(M)$ et non le message M lui même qui est signée. Les caractéristiques d'une bonne fonction de hachage est qu'elle soit non-reversible, qu'elle ne produit aucune collision, c'est-à-dire que deux messages M et M' ne produisent pas la même valeur de hachage et qu'elle soit rapide. Malheureusement, des chercheurs ont découvert très récemment des collisions pour plusieurs fonctions de hachage incluant MD5, SHA0 et SHA1 ([32]). Ainsi, si l'on utilise une de ces fonctions pour produire la signature digitale, le receveur qui fait la vérification ne peut plus être assuré que le message signée soit réellement M. Petite anecdote pour souligner l'importance des ces découvertes : un homme en Australie a contesté avec succès une amende pour excès de vitesse dont la preuve a été produite par une camera dont la photo digitale avait été hachée par la fonction MD5 et ensuite signée. La décision de la cour reposait alors sur ces questions techniques.

Ces propos nous rappellent que si nous utilisons des signatures digitales pour générer les preuves de non-répudiation, il y a certaine précaution à prendre pour garantir que ces preuves aient une valeur réelle.

2.2 Protocoles de non-répudiation

Nous présentons dans cette section des exemples précis de protocoles de non-répudiation sans et avec TTP. Malheureusement, les définitions des concepts de base de la non-répudiation varient d'un chercheur à l'autre. Pour remédier à ce problème et aider à la compréhension, nous formalisons dans ce qui suit, les concepts tels les preuves et les propriétés de non-répudiation avant de présenter les protocoles.

2.2.1 Formalisation des concepts de la non-répudiation

Introduisons tout d'abord une notation de base pour décrire les protocoles de non-répudiation. Les agents du protocole sont identifiés par A, B et TTP, A jouant le rôle d'*initiateur* du protocole et source du message M, B le rôle du *receveur* et TTP le rôle du *TTP*. Les protocoles de sécurité utilisent généralement un agent I, qui joue le rôle d'intrus. On ne se servira pas d'intrus dans les protocoles de non-répudiation puisque la menace vient plutôt des actions d'un des agents qui agit malhonnêtement. Nous dénotons par $\mathcal{A}$ l'ensemble de tous les agents participants au protocole.

Un protocole de non-répudiation consiste en une séquence de transmissions de messages appelées étapes. Nous formalisons cette description dans la définition qui suit.

Définition 2.2.1 (Protocole de non-répudiation) *Un protocole $P^n_{\mathcal{A}}(M)$ de non-répudiation avec TTP où $\mathcal{A} = \{A, B, TTP\}$, ou sans TTP où $\mathcal{A} = \{A, B\}$, consiste en n etapes de communications notées $a_1, \ldots, a_n$ de la forme :*

$$a_i.Agent_j \rightarrow Agent_k : Message \tag{2.1}$$

où $Agent_j, Agent_k \in \mathcal{A}$

Souvent, il est nécessaire d'indiquer une exécution donnée du protocole. Pour le reste de ce document nous dénotons une exécution de $P^n_{\mathcal{A}}(M)$, par T.

L'ensemble de tous les messages possible m est défini par le grammaire BNF suivant :

$$\begin{array}{rcll} m & ::= & A & \text{identificateur d'agent} \\ & | & k & \text{clé cryptographique} \\ & | & \{m\}_k & \text{message encrypté ou signé par la clé } k \\ & | & X & \text{variable message} \\ & | & \{PK_x, SK_x\} & \text{les clés publique et privée de l'agent } X \\ & | & m.m' & \text{enchaînement de messages} \end{array} \tag{2.2}$$

On utilise une annotation de la forme m_x pour associer un message m à l'agent initiateur X. Les messages qui ne peuvent être divisés sont dites messages *atomiques*.

Les *connaissances* d'un agent acquises pendant l'exécution du protocole représentent l'ensemble des messages que chaque agent possède avant l'exécution du protocole et les messages qu'il reçoit pendant l'exécution du protocole. L'ensemble des connaissances d'un agent X après l'exécution T du protocole $P^n_{\mathcal{A}}(M)$ est dénoté par $\mathcal{K}_X(\mathsf{T})$.

Propriétés de la non-répudiation

Les preuves échangées sont à la base de la non-répudiation. Nous présentons ci-dessous les définitions des deux preuves qui peuvent être générées pendant l'exécution d'un protocole $P_{\mathcal{A}}^{n}(M)$.

Définition 2.2.2 (Preuves d'origine et de réception) *Une preuve de non-répudiation de réception ou NRR doit permettre à un juge indépendant de décider si le receveur B a réellement reçu le message M de l'initiateur A au cours d'une exécution T du protocole $P_{\mathcal{A}}^{n}(M)$. La preuve de non-répudiation d'origine ou NRO doit permettre à un juge indépendant de décider, si l'initiateur A a réellement transmis le message M à B pendant une exécution T du protocole $P_{\mathcal{A}}^{n}(M)$.*

Pour offrir un service de non-répudiation de réception ou d'origine, un protocole $P_{\mathcal{A}}^{n}(M)$ doit générer le NRR pour A ou le NRO pour B respectivement. Les items requis par B sont donc le NRO et le message M, tandis que A requiert le NRR.

Un protocole de non-répudiation, qui génère le NRR ou le NRO, satisfait respectivement aux propriétés de non-répudiation d'origine et de réception que nous définissons ci-bas, en terme de la possession de ces preuves.

Définition 2.2.3 (Non-répudiation d'origine et de réception) *Un protocole $P_{\mathcal{A}}^{n}(M)$ satisfait la propriété de non-répudiation d'origine si à la fin d'une exécution T du protocole :*

$$NRO \in \mathcal{K}_B(\mathsf{T}) \tag{2.3}$$

et un protocole $P_{\mathcal{A}}^{n}(M)$ satisfait la propriété de non-répudiation de réception si à la fin d'une exécution T du protocole

$$NRR \in \mathcal{K}_A(\mathsf{T}) \tag{2.4}$$

Dans l'équation 2.5, nous présentons un exemple d'un protocole de non-répudiation entre deux agents sans TTP qui satisfait les propriétés de non-répudiation d'origine et de réception.

$$\begin{array}{lll} 1: & A \rightarrow B: & M \\ 2: & B \rightarrow A: & \text{NRR} \\ 3: & A \rightarrow B: & \text{NRO} \end{array} \tag{2.5}$$

Dans ce protocole, l'initiateur A initie l'échange en envoyant le message M. Si B répond avec le NRR, A enverra le NRO.

Agents malhonnêtes et canaux de communication peu fiables

Il y a deux facteurs principaux qui peuvent empêcher qu'un protocole ne puisse offrir un service de non-répudiation d'origine ou de réception : un agent malhonnête ou un canal de communication peu fiable. Dans le protocole de l'équation 2.5 ci-haut, il est clair que si A agit de façon malhonnête, et n'envoie pas son dernier message ou que le message est perdu, alors B n'obtiendra pas tous ses items requis.

Pour une exécution donnée du protocole, on dit qu'un agent malhonnête a triché, si par ses actions, il a reçu au moins un de ses items requis, mais que l'autre ne reçoit pas au moins un des siens. Donc A a triché si, à la fin d'une exécution T d'un protocole, il a obtenu le NRR mais B n'a pas reçu M ou le NRO. De la même façon, B triche s'il détient M ou le NRO, mais que A n'ait pas obtenu son NRR. Nous formalisons cette idée par la définition suivante :

Définition 2.2.4 (Tricher) *L'initiateur A d'un protocole de non-répudiation $P_{\mathcal{A}}^{n}(M)$ a* triché *si à la fin d'une exécution T du protocole $P_{\mathcal{A}}^{n}(M)$ soit que*

$$NRR \in \mathcal{K}_A(\mathsf{T}) \quad et \quad \begin{cases} NRO \notin \mathcal{K}_B(\mathsf{T}) \ ou \\ M \notin \mathcal{K}_B(\mathsf{T}) \end{cases} \tag{2.6}$$

De la même façon, un agent B a triché si à la fin de l'exécution T de $P_{\mathcal{A}}^{n}(M)$,

$$NRR \notin \mathcal{K}_A(\mathsf{T}) \quad et \quad \begin{cases} NRO \in \mathcal{K}_B(\mathsf{T}) \ ou \\ M \in \mathcal{K}_B(\mathsf{T}) \end{cases} \tag{2.7}$$

La fiabilité des réseaux de communication est l'autre facteur qui peut empêcher qu'un protocole $P_{\mathcal{A}}^{n}(M)$ offre les services de non-répudiation. Dans la littérature, on retrouve trois types de canaux de communication : *non-fiable*, *résilient* et *opérationnel*. Dans un canal non-fiable, un message peut toujours être perdu. Dans un canal résilient, un message ne peut être perdu, mais le temps de sa transmission est fini mais inconnu. Finalement, dans un canal opérationnel, un message n'est jamais perdu et le temps de transmission est constant et connu.

Il est clair que face à un agent malhonnête ou un canal non-fiable, les propriétés de non-répudiation d'origine et de réception ne sont pas suffisantes pour garantir que le protocole offrira un service de non-répudiation d'origine ou de réception en toutes circonstances. Dans le protocole de l'équation 2.5, le service de non-répudiation d'origine ne sera pas offert, si, par exemple, A n'envoie pas son dernier message ou qu'il est perdu par le canal de communication.

De telles garanties requierent des propriétés additionnelles. La première et la plus importante étant l'*équité*, qui spécifie que soit que les agents reçoivent tous leurs items requis ou soit qu'ils n'en reçoivent aucun et ce pour chaque exécution possible du protocole.

Nous présentons ci-dessous, une définition formelle de la propriété d'équité en se basant sur la définition de [21]).

Définition 2.2.5 (La propriété d'équité) *Un protocole $P_{\mathcal{A}}^{n}(M)$ est dit équitable, si à la fin de chaque session T du protocole, soit que*

$$\begin{cases} NRR \in \mathcal{K}_A(\mathsf{T}) \ \wedge NRO \in \mathcal{K}_B(\mathsf{T}) \ \wedge M \in \mathcal{K}_B(\mathsf{T}) \ ou \\ NRR \notin \mathcal{K}_A(\mathsf{T}) \ \wedge NRO \notin \mathcal{K}_B(\mathsf{T}) \ \wedge M \notin \mathcal{K}_B(\mathsf{T}) \end{cases} \tag{2.8}$$

Dans l'exemple du protocole l'équation 2.5, il est facile de trouver au moins une exécution qui ne satisfait pas à la définition. En effet, si A ne transmet pas son dernier message, l'équité ne peut être satisfaite. De la même manière, si B n'envoie pas le deuxième message, l'équité ne sera encore pas satisfaite.

En plus de l'équité, il peut aussi être important d'exiger que le temps d'exécution d'un protocole ne peut être manipulé pour obtenir un gain quelconque. La propriété de *timeliness* spécifie qu'un agent malhonnête ne peut manipuler le temps pour tricher.

Définition 2.2.6 (La propriété de timeliness) *Un protocole de non-répudiation $P_{\mathcal{A}}^{n}(M)$ satisfait la propriété de timeliness, si et seulement si aucun agent ne peut tricher en manipulant le temps d'exécution du protocole.*

Le protocole de l'équation 2.5, pourrait ne pas respecter le timeliness, si un agent malhonnête A retient son deuxième message pour un temps qui dépasse le temps d'attente maximum de B.

Une dernière propriété qui peut être exigée est la *viabilité*. Celle-ci spécifie que si les agents sont honnêtes, ils recevront leurs items requis. On donne la définition suivante de la viabilité (en se basant encore sur la définition donnée dans [21]) :

Définition 2.2.7 (La propriété de viabilité) *Un protocole $P_{\mathcal{A}}^{n}(M)$ respecte la propriété de viabilité, si pour chaque exécution T dans laquelle les agents agissent honnêtement, on a que*

$$NRR \in \mathcal{K}_A(\mathsf{T}) \ \wedge NRO \in \mathcal{K}_B(\mathsf{T}) \ \wedge M \in \mathcal{K}_B(\mathsf{T}) \tag{2.9}$$

Avant de continuer, faisons certaines observations concernant la propriété d'équité.

Observations sur la propriété de l'équité

L'équité garantit qu'un agent malhonnête ne puissent obtenir un avantage quelconque (en terme des items requis) au dépend de l'autre. L'action de tricher a été bien définie dans la définition 2.2.4. Dans ce qui suit, nous analyserons en détail les cas où l'équité ne tient pas.

De la définition 2.2.5, l'équité est satisfaite, si à la fin de chaque session T de $P^n_{A,B}(M)$ on a :

$$\begin{cases} NRR \in \mathcal{K}_A(\mathsf{T}) \ \wedge \ NRO \in \mathcal{K}_B(\mathsf{T}) \ \wedge \ M \in \mathcal{K}_B(\mathsf{T}) \text{ ou} \\ NRR \notin \mathcal{K}_A(\mathsf{T}) \ \wedge \ NRO \notin \mathcal{K}_B(\mathsf{T}) \ \wedge \ M \notin \mathcal{K}_B(\mathsf{T}) \end{cases} \tag{2.10}$$

Examinons en détail les 6 cas pour lesquels l'équité ne tient pas. Cette étude permettra de montrer que chacun des cas (possibles) représente un danger réel pour l'un ou l'autre des agents. D'après l'équation 2.10, les 6 cas sont :

$$\begin{array}{l} 1.NRR \in \mathcal{K}_A(\mathsf{T}) \ \wedge \ NRO \in \mathcal{K}_B(\mathsf{T}) \ \wedge \ M \notin \mathcal{K}_B(\mathsf{T}) \\ 2.NRR \in \mathcal{K}_A(\mathsf{T}) \ \wedge \ NRO \notin \mathcal{K}_B(\mathsf{T}) \ \wedge \ M \notin \mathcal{K}_B(\mathsf{T}) \\ 3.NRR \in \mathcal{K}_A(\mathsf{T}) \ \wedge \ NRO \notin \mathcal{K}_B(\mathsf{T}) \ \wedge \ M \in \mathcal{K}_B(\mathsf{T}) \\ 4.NRR \notin \mathcal{K}_A(\mathsf{T}) \ \wedge \ NRO \in \mathcal{K}_B(\mathsf{T}) \ \wedge \ M \in \mathcal{K}_B(\mathsf{T}) \\ 5.NRR \notin \mathcal{K}_A(\mathsf{T}) \ \wedge \ NRO \in \mathcal{K}_B(\mathsf{T}) \ \wedge \ M \notin \mathcal{K}_B(\mathsf{T}) \\ 6.NRR \notin \mathcal{K}_A(\mathsf{T}) \ \wedge \ NRO \notin \mathcal{K}_B(\mathsf{T}) \ \wedge \ M \in \mathcal{K}_B(\mathsf{T}) \end{array} \tag{2.11}$$

Remarquons tout de suite que les cas 1 et 2 sont logiquement impossibles. Le fait que A possède NRR implique que B l'a généré, ce qui implique nécessairement que l'agent B possède le message M selon la définition 2.2.2 du NRR. Il nous reste alors seulement quatre cas à examiner.

Pour chacun de ces 4 cas, on peut déterminer qui, entre les deux agents, sera avantagé. On remarque que l'avantage que peut tirer un agent d'un de ces cas, dépend de deux facteurs : le contenu du message M, et si les agents connaissent le contexte de l'échange. Prenons l'exemple où l'agent B ne reçoit pas la preuve d'origine NRO pour le message M. Si M est une marchandise défectueuse, les conséquences pour B de ne pas avoir reçu la preuve d'origine, pourraient être complètement différentes si M était non défectueux. En plus, le fait que B sache qu'il attend par exemple une facture, donc qu'il connaît le contexte de l'échange, pourrait aussi avoir un impact majeur dans le cas où il ne reçoit jamais le message.

Dans notre analyse, nous tiendrons compte alors du contenu du message, et nous supposerons que les agents connaissent toujours le contexte de chaque échange (ce qui dans le plus part des cas ne serait pas une supposition irréaliste). Nous identifions trois types de

Cas	M/P non-défectueuse	M/P défectueuse	Facture
3	$A\|\|$	$A\uparrow$	$A\|\|$
4	$A\downarrow$	$A\downarrow$	$A\downarrow$
5	$A\uparrow$	$A\uparrow$	$A\downarrow$
6	$A\|\|$	$A\uparrow$	$A\downarrow$

TAB. 2.1 – Avantages de l'initiateur

messages dans cette analyse : une *marchandise/paiement non défectueux*, une *marchandise/paiement défectueux*, et une *facture*. Connaissant le contexte, nous supposons que B saura si le message qu'il va recevoir est une marchandise/paiement ou une facture. Il ne pourra pas par contre s'apercevoir qu'une marchandise/paiement est défectueuse sans l'avoir reçue.

Nous allons maintenant examiner les cas 3, 4, 5 et 6 séparément. Dans le cas 3, B reçoit seulement M, tandis que A reçoit son NRR. Donc A a le pouvoir de répudier l'envoi du message M à B. Si M représente une marchandise/paiement défectueux, A peut toujours le répudier et B n'aura aucun recours. Par contre, A ne retire aucun avantage de répudier les autres types de message.

Pour les cas 4, 5 et 6, A ne reçoit pas son NRO, donc B peut toujours répudier la réception de M. Dans le cas 4, B reçoit tous ses items, donc, si le message est une marchandise/paiement non défectueux ou une facture, A sera perdant si B répudie la réception. Pour une marchandise/paiement défectueux, B aura tous les recours nécessaires pour poursuivre A. Dans le cas 5, B ne reçoit pas le message, mais seulement la preuve d'origine. Sans le message, B ne peut prouver ce que A lui a envoyé. Donc pour les deux types de marchandise/paiement B est perdant. Si c'est une facture, connaissant le contexte de la communication, B peut la répudier et obtenir ainsi un avantage. Finalement, pour le cas 6, B reçoit seulement le message, et n'ayant aucune preuve d'origine, il ne peut poursuivre A si c'est une marchandise/paiement défectueuse. Si c'est une facture, B peut l'ignorer sans risque et A sera perdant.

Nous résumons notre courte analyse par le tableau 2.1, dans lequel nous présentons les 4 cas possibles, en notant les avantages en termes de l'agent A et ce pour les 3 types de messages. Nous dénotons un avantage pour A par $A\uparrow$, un désavantage par $A\downarrow$ et une situation pour lequel il ne retire aucun avantage ni désavantage par $A||$.

Nous remarquons que chaque cas représente un risque pour au moins un des agents. On remarque aussi qu'en tout, le nombre de situations désavantageuses pour A est équivalent au nombre qui lui sont avantageuses (4). Cette symetrie n'est pas présente pour tous les

types de messages. Pour certains, A ne pourra jamais en tirer avantage (les factures), tandis que pour d'autres, il ne pourra perdre (marchandise/paiement défectueuses).

Équité et atomicité

La propriété de l'équité ressemble en certains points à la propriété d'atomicité. Celle-ci est utilisée pour prévenir les pertes dans les transactions de commerce électronique. De la définition de [31], l'atomicité relie deux ou plusieurs opérations d'une transaction de commerce électronique, de façon à ce que :

- toutes les opérations de la transaction sont exécutées ou
- aucunes ne sont exécutées.

Par exemple, supposons que A est un client et B un marchand, et que A achète pour $\$x$ de B. Il y a exactement deux opérations qui doivent s'effectuer après l'achat : le compte de A doit indiquer un retrait de $\$x$ tandis que le compte du marchand doit augmenter de $\$x$. Il est clair que si la transaction finit après la première opération, le marchand sera perdant. Si par contre, seulement la deuxième opération est exécutée alors le client est perdant. Donc pour le client et le marchand, les seules exécutions convenables de la transaction sont que les deux opérations se fassent ou qu'elles ne se fassent pas du tout.

Les propriétés de l'équité et de l'atomicité partagent toutes deux cette notion du tout ou rien. Par contre, tandis que l'atomicité relie les opérations d'une transaction, l'équité relie les preuves de réception et d'origine à chacune des opérations. Comme souligné dans [31], l'équité et l'atomicité doivent opérer de concert pour assurer l'intégrité et la fiabilité des transactions de commerce électronique.

Composantes d'un message et leurs preuves

Souvent, pour satisfaire aux propriétés de non-répudiation (surtout l'équité), un protocole divisera le message M en n composantes m_i. Dans le cas où posséder M est équivalent à posséder les n composantes $\{m_1, ..., m_n\}$, nous introduisons la notation suivante :

Définition 2.2.8 (Composantes de M) *Nous dénotons le fait que posséder M est équivalent à posséder n messages $\{m_1, ..., m_n\}$ par*

$$M \asymp \{m_1, ..., m_n\}$$

Un protocole qui utilise des composantes doit aussi générer des preuves d'origine et de réception pour chacune de ces composantes. Dans ce cas, pour obtenir le NRO, B doit obtenir les preuves d'origine pour chacune des composantes de M. De même pour A et le NRR. Nous présentons cette idée plus formellement ci bas.

Définition 2.2.9 (Preuves partielles) *Supposons que* $M \asymp \{m_1, ..., m_n\}$. *Si* $EOO(m_i)$ *est la preuve d'origine de la composante* m_i *générée par* A *durant une exécution* T *de* $P^n_{\mathcal{A}}(M)$, *alors on obtient*

$$\forall i = 1, .., n,\ EOO(m_i) \in \mathcal{K}_B(\mathsf{T}) \Leftrightarrow NRO \in \mathcal{K}_B(\mathsf{T}) \tag{2.12}$$

et si $EOR(m_i)$ *est la preuve de réception de la composante* m_i *générée par* B *durant l'exécution* T *de* $P^n_{\mathcal{A}}(M)$, *alors on obtient*

$$\forall i = 1, .., n,\ EOR(m_i) \in \mathcal{K}_A(\mathsf{T}) \Leftrightarrow NRR \in \mathcal{K}_A(\mathsf{T}) \tag{2.13}$$

Considérons l'exemple d'un protocole de non-répudiation sans TTP de l'équation 2.14. Dans ce protocole, A n'envoie pas M mais les deux composantes : $C = \{M\}_k$ et k. B et A s'échangent respectivement les preuves de réception et d'origine pour chacune de ces composantes.

$$\begin{array}{lll} 1: & A \rightarrow B: & C, EOO(C) \\ 2: & B \rightarrow A: & EOR(C) \\ 3: & A \rightarrow B: & k, EOO(k) \\ 4: & B \rightarrow A: & EOR(k) \end{array} \tag{2.14}$$

Ce protocole satisfait les propriétés de non-répudiation d'origine et de réception, mais n'est pas équitable. En effet B peut tricher facilement : s'il arrête le protocole après avoir reçu le dernier message de A, il possédera M et le NRO mais A n'ayant pas reçu la dernière preuve partielle, n'aura pas son NRR.

Etiquettes et identificateurs d'exécution

Chaque message d'un protocole $P^n_{\mathcal{A}}(M)$ a une fonction très précise. Pour éviter qu'un agent malhonnête se serve d'un message pour une raison autre que sa fonction initiale. Ainsi une étiquette est ajouter à chaque message pour identifier sa fonction. La notation la plus répandue d'une étiquette est f_x, qui dénote que la fonction du message est x.

Aussi, pour éviter qu'un agent malhonnête réutilise des messages d'une exécution précédente, on associe, à chaque étape d'une exécution, la même valeur aléatoire unique. On la dénote souvent par l.

Dans l'exemple du protocole de l'équation 2.14, l'exécution et les messages peuvent être identifiés de la façon suivante :

$$\begin{array}{ll} 1: & A \rightarrow B: \ f_{EOO(C)}, l, B, C, EOO(C) \\ 2: & B \rightarrow A: \ f_{EOR(C)}, l, A, EOR(C) \\ 3: & A \rightarrow B: \ f_{EOO(k)}, l, B, k, EOO(k) \\ 4: & B \rightarrow A: \ f_{EOR(k)}, l, A, EOR(k) \end{array} \tag{2.15}$$

On remarque ici que l'on a aussi identifié la destination de chaque message.

Pour que ces identificateurs ne soient modifiés à leur tour, ils sont protégés par la signature digitale utilisée pour générer les preuves. Ainsi, dans le protocole 2.15, si SK_a et SK_b sont les clés privées de A et B respectivement, on définit les preuves partielles :

$$\begin{array}{l} EOO(C) = \{f_{EOO(C)}, l, B, C\}_{SK_a} \\ EOR(C) = \{f_{EOR(C)}, l, A, C\}_{SK_b} \\ EOO(k) = \{f_{EOO(k)}, l, B, k\}_{SK_a} \\ EOR(k) = \{f_{EOR(k)}, l, A, k\}_{SK_b} \end{array} \tag{2.16}$$

2.2.2 Protocoles de non-répudiation et TTP

Le développement des protocoles de non-répudiation a été énormément influencé par le résultat de Even et Yacobi [12]. Ils ont démontré qu'un protocole de non-répudiation entre deux agents ne peut garantir l'équité sans la participation d'un TTP. Le résultat se base sur le fait que sans TTP, il existera toujours une étape du protocole où l'un des participants possédera un item de plus que l'autre. Ainsi, si le protocole prend fin à cette étape, l'équité ne sera pas établie puisqu'un des agents aura un avantage sur l'autre

Un TTP représente une manière efficace d'assurer l'équité, mais peut aussi représenter un point d'engorgement de l'échange. Donc en dépit du résultat de Even et Yacobi, les chercheurs ont vu l'utilité de concevoir des protocoles de non-répudiation sans TTP. Ainsi, il existe aujourd'hui deux grandes types de protocole de non-répudiation : avec ou sans TTP. Nous allons étudier dans ce qui suit, un exemple d'un protocole sans TTP et deux exemples de protocoles avec TTP

Protocoles de non-repudiation sans TTP

Selon le resultat de Even et Yacobi [12], un protocole de non-répudiation ne peut garantir l'équité sans TTP. Il existe cependant une version moins stricte de l'équité que peut satisfaire un protocole de non-répudiation sans TTP. La propriété d'*équite probabiliste* permet à un agent malhonnête de tricher avec une certaine probabilité non zéro p inférieur à ε. On définit formellement ce concept ci-dessous.

Définition 2.2.10 (ε-équitable) *Un protocole de non-répudiation $P^n_{A,B}(M)$ est ε-équitable si et seulement si à la fin de chaque exécution T du protocole $P^n_A(M)$, on obtient*

$$\begin{cases} NRR \in \mathcal{K}_A(\mathsf{T}) \ \wedge NRO \in \mathcal{K}_B(\mathsf{T}) \ \wedge M \in \mathcal{K}_B(\mathsf{T}) \ ou \\ NRR \notin \mathcal{K}_A(\mathsf{T}) \ \wedge NRO \notin \mathcal{K}_B(\mathsf{T}) \ \wedge M \notin \mathcal{K}_B(\mathsf{T}) \end{cases} \text{avec probabilité} \geq (1-\varepsilon) \text{ où }, 0 < \varepsilon \leq \tag{2.17}$$

Dans [21], Markowitch et Roggeman présente un protocole de non-répudiation sans TTP, qui prétends être ε-équitable. Ce protocole introduit le concept de fausse clé. Le protocole décompose le message en deux composantes : $C = M_k$ et la clé k ($M \asymp C, k$). L'équité se base sur le fait que A transmettra un nombre aléatoire de fausses clés (nombre inconnu de B) avant d'envoyer la vraie et sur le fait que A fixe avec B une valeur d'attente de réponse maximale t.

Dans un premier message, A envoie C et sa preuve d'origine $EOO(C)$. Si B réponds avec la preuve de réception $EOR(C)$, A enverra soit la clé k avec sa preuve d'origine $EOO(k)$ ou soit une fausse clé r_1 avec sa preuve d'origine $EOO(r_1)$. En le recevant, B devra choisir d'y répondre.

Si B ne répond pas et que le message de A contient la bonne clé, il aura triché et gagné : il possédera le message et le NRO sans que A ait le NRR. Si la clé est fausse, le protocole s'arrêtera quand le délai t fixé par A sera dépassé. Si B répond et la clé est la bonne, les deux agents obtiennent leurs items requis. Mais si la clé est fausse, A enverra au moins un autre message contenant une clé (vraie ou fausse) avec la preuve d'origine et le processus se répétera.

Si $n-1$ représente le nombre de fausses clés transmises, alors le protocole MR se décrit ainsi :

$$
\begin{array}{lll}
1: & A \rightarrow B: & f_{EOO(C)}, B, l, C, EOO(C) \\
2: & B \rightarrow A: & f_{EOR(C)}, A, l, EOR(C) \\
3: & A \rightarrow B: & f_{EOO(r_1)}, B, l, r_1, EOO(r_1) \\
4: & B \rightarrow A: & f_{EOR(r_1)}, B, l, EOR(r_1) \\
\vdots & & \\
2n-1: & A \rightarrow B: & f_{EOO(r_{n-1})}, B, l, r_{n-1}, EOO(r_{n-1}) \\
2n: & B \rightarrow A: & f_{EOR(r_{n-1})}, A, l, EOR(r_{n-1}) \\
2n+1: & A \rightarrow B: & f_{EOO(k)}, B, l, k, EOO(k) \\
2n+2: & B \rightarrow A: & f_{EOR(k)}, A, l, EOR(k)
\end{array}
\tag{2.18}
$$

Pour toute composante m, les preuves sont générées par des signature digitales et sont définies ci bas.

$$
\begin{aligned}
EOO(m) &= \{f_{EOO(m)}, l, B, m\}_{SK_a} \\
EOR(m) &= \{f_{EOR(m)}, l, A, m\}_{SK_b}
\end{aligned}
\tag{2.19}
$$

L'équité probabiliste sera satisfaite, si B ne peut deviner avec probabilité supérieure ou égale à $1/n$ à quelle étape k sera transmise. Il est donc essentiel, que B ne puisse différencier entre un message qui contient la vraie ou une fausse clé. Les auteurs spécifient que l'algorithme de déchiffrement doit être choisi de telle sorte que le déchiffrement de C (même partielle) ne se réalise pas dans un temps inférieur à t. Remarquons, qu'avec le délai t, le protocole MR semble satisfaire la propriété de timeliness même si le canal entre A et B est non-fiable.

Le protocole MR possède deux caractéristiques intrinsèques des protocoles de non-répudiation sans TTP : un grand nombre de messages échangés et une limite placée sur la puissance de calcul des agents. En effet, dans le protocole MR, la probabilité de tricher dépend entièrement du nombre de fausses clés envoyées par A et le fait que B ne doit pas pouvoir déchiffrer un message dans un temps inférieur à t. Le protocole MR à l'avantage, que des limites sont imposées seulement sur le puissance de calcul de l'agent B et non sur les deux.

Protocoles de non-répudiation avec TTP

Un TTP est le troisième agent d'un protocole de non-répudiation. Dans son rôle, il ne peut agir en faveur de l'un ou l'autre des agents et doit toujours suivre le protocole. Le TTP agit normalement en tant que passerelle sécuritaire ou d'entrepôt.

On propose la définition suivante d'un TTP :

Définition 2.2.11 (TTP) *Un TTP est un agent du protocole $P_{\mathcal{A}}^{n}(M)$, $\mathcal{A} = \{\mathcal{A}, \mathcal{B}, \mathcal{TTP}\}$, qui par ses actions ne peut favoriser l'un ou l'autre des agents et dans chaque exécution* T *doit exécuter les étapes qui lui sont assignées.*

Le TTP peut confirmer la réception d'un message, appelé *soumission*, en générant une preuve de *confirmation*. Cette idée est formalisée par la définition suivante.

Définition 2.2.12 (Preuve de soumission et de confirmation) *La preuve de soumission du message m au TTP, dénotée par $Sub(m)$, est la preuve d'origine du message m générée par l'agent X source du message. La confirmation d'une soumission, dénotée par $Conf(m)$, est la preuve que le TTP a reçu le message m et $Sub(m)$.*

La confirmation agit en tant que preuve d'origine et de réception du message m pour les agents A et B. On peut formaliser cette notion par la définition suivante.

Définition 2.2.13 (La confirmation) *Dans un protocole $P_{\mathcal{A}}^{n}(M)$ et une exécution* T *donnée,*

1. $Conf(m) \in \mathcal{K}_A(\mathsf{T}) \Rightarrow EOR(m) \in \mathcal{K}_A(\mathsf{T})$ *et*
2. $Conf(m) \in \mathcal{K}_B(\mathsf{T}) \Rightarrow EOO(m) \in \mathcal{K}_B(\mathsf{T})$

Les chercheurs classifient les protocoles de non-répudiation avec TTP par rapport au niveau de participation du TTP. L'article [18] présente la taxonomie suivante :

1. **Inline** - Le TTP participe à chaque transmission d'une exécution.
2. **Online** - Le TTP participe à chaque exécution mais pas à chaque transmission.
3. **Offline** - Le TTP participe seulement à une exécution dans le cas de tricherie ou dans le cas d'une erreur de transmission.

Plus le TTP participe au protocole, plus il est facile de garantir les propriétés. Par contre, plus il participe, plus l'échange a des chances d'être ralenti. Dans ce qui suit, nous présentons des exemples de chaque type.

Un protocole de non-répudiation avec TTP inline

Un exemple simple d'un protocole $P_{\mathcal{A}}^{n}(M)$ avec TTP inline est présenté ci-dessous :

$$
\begin{array}{lll}
1: & A \rightarrow TTP: & C, Sub(C) \\
2: & TTP \rightarrow B: & C, Conf(C) \\
3: & B \rightarrow TTP: & EOR(C) \\
4: & TTP \rightarrow A: & C, Conf(C) \\
5: & A \rightarrow TTP: & k, Sub(k) \\
6: & TTP \rightarrow B: & k, Conf(k) \\
7: & TTP \rightarrow A: & Conf(k)
\end{array}
\tag{2.20}
$$

On remarque, que le TTP participe à chaque transmission entre A et B.

Le protocole de non-répudiation ZG avec TTP online

On peut améliorer la performance d'un protocole avec TTP en forçant les agents de communiquer avec le TTP pour obtenir leurs preuves. C'est la stratégie adoptée par Zhou et Gollman dans [35]. Ce protocole utilise encore la décomposition du message en son encryptage C et la clé symétrique k.

Le TTP online quant à lui, agit comme une base de données publique accessible pour lecture seulement par n'importe qui. Il publiera les données transmises par A et B, et ceux-ci accéderont aux preuves par une commande *get* de ftp par exemple.

Dans une première étape, A et B s'échangent les preuves d'origine et de réception de C. Ensuite l'initiateur A effectue une soumission du message k au TTP. Le TTP générera alors une confirmation de la réception de la clé, que A et B devront alors récupérer du TTP.

Le protocole est décrit ci-dessous :

$$
\begin{array}{lll}
1: & A \rightarrow B: & f_{EOO(C)}, B, l, C, EOO(C) \\
2: & B \rightarrow A: & f_{EOR(C)}, A, l, EOR(C) \\
3: & A \rightarrow TTP: & f_{Sub(k)}, B, l, k, Sub(k) \\
4: & B \leftrightarrow TTP: & f_{Conf(k)}, A, B, l, k, Conf(k) \\
5: & A \leftrightarrow TTP: & f_{Conf(k)}, A, B, l, k, Conf(k)
\end{array}
\tag{2.21}
$$

Les auteurs spécifient que pour obtenir les confirmations, les agents A et B devront exécuter la commande *ftp get* (indiqué par $\leftrightarrow$).

Comme auparavant, les preuves sont générées par des signatures digitales et sont :

$$\begin{aligned} EOO(C) &= \{f_{EOO(C)}, B, l, C)\}_{SK_a} \\ EOR(C) &= \{f_{EOR(C)}, A, l, C\}_{SK_b} \\ Sub(k) &= \{f_{Sub(k)}, A, B, l, k\}_{SK_a} \\ Conf(k) &= \{f_{EOO(k)}, A, B, l, k\}_{SK_{ttp}} \end{aligned} \tag{2.22}$$

Ce qui différencie ce protocole de bien d'autres, est le fait que B ne génère aucune preuve pour k et qu'il s'engage complètement au protocole une fois sa preuve de réception de C envoyée. En effet, par la définition 2.2.13, on a pour une exécution T donnée, que :

1. $Conf(m) \in \mathcal{K}_A(\mathsf{T}) \wedge EOR(C) \in \mathcal{K}_A(\mathsf{T}) \Rightarrow NRR \in \mathcal{K}_A(\mathsf{T})$ et
2. $Conf(m) \in \mathcal{K}_B(\mathsf{T}) \wedge EOO(C) \in \mathcal{K}_B(\mathsf{T}) \Rightarrow NRO \in \mathcal{K}_B(\mathsf{T})$

Selon les auteurs, le protocole est équitable si le canal de communication entre les agents et le TTP est opérationnel. Le canal de communication entre A et B peut être non fiable, à condition que chacun peut arrêter le protocole à n'importe quelle étape. Encore selon les auteurs, si un délai t sur le temps d'attente maximal est établi entre les agents, le protocole satisfera la propriété de timeliness. Nous discutons la validité de cette assertion dans la prochaine section.

Timeliness et le protocole ZG

A cause de sa simplicité et de son efficacité le protocole ZG a été favorablement reçu. Malheureusement, on a vite réalisé que le protocole ZG ne satisfaisait pas la propriété de timeliness. En effet, dans [16], les auteurs ont démontré qu'un initiateur pourrait obtenir un avantage en manipulant le temps d'envoi d'un de ses messages. L'attaque se décrit comme suit.

Supposons que B se fixe un temps t, qui représente sa limite maximale d'attente pour obtenir ses preuves du TTP après avoir envoyé $EOR(C)$. Nous supposons aussi que si ce temps d'attente est dépassé, il conclura que la session est terminée et effacera toutes les informations reliées à cette session. Imaginons maintenant que pour une session donnée, l'initiateur retient la transmission du message contenant la soumission $f_{Sub(k)}, B, l, k, Sub(k)$ pour un temps $d > t$. Dans ce cas, ayant reçu la soumission de A au temps d, le TTP publiera les preuves et A obtiendra ainsi son NRR. Par contre, B ayant effacé la valeur $EOO(C)$, ne pourra obtenir son NRO même s'il communique avec le TTP.

Cette faiblesse est grave, surtout quand on considère la facilité avec laquelle l'initiateur peut obtenir un avantage. Plusieurs solutions ont été conçues pour la corriger, incluant une solution offerte par les concepteurs du protocole ZG eux mêmes. Cette solution est

décrite dans [8], et consiste à inclure des valeurs de temps réel dans les messages, qui seront vérifiés par le juge en cas de dispute. Les valeurs de temps sont :

- T : temps-limite après lequel A et B ne pourrons obtenir les preuves du TTP.
- T_0 : temps actuel auqel le TTP publie ses preuves.
- T_1 : défini par B, le temps-limite avant lequel A doit transmettre $Sub(k)$ au TTP.

Ces valeurs sont inclues dans le protocole ZG comme suit :

$$\begin{array}{lll} 1: & A \rightarrow B: & f_{EOO(C)}, B, l, T, C, EOO(C,T) \\ 2: & B \rightarrow A: & f_{EOR(C)}, A, l, T_1, EOR(C,T,T_1) \\ 3: & A \rightarrow TTP: & f_{Sub(k)}, B, l, T, k, Sub(k,T) \\ 4: & B \leftrightarrow TTP: & f_{Conf(k)}, A, B, l, T, T_0, k, Conf(k,T,T_0) \\ 5: & A \leftrightarrow TTP: & f_{Conf(k)}, A, B, l, T, T_0, k, Conf(k,T,T_0) \end{array} \tag{2.23}$$

Les valeurs de temps sont incluses dans les preuves d'origine et de réception. Ainsi, un juge peut vérifier qu'elles ont été réellement générées par les agents concernés.

Un juge acceptera une preuve d'une session de ce protocole, si seulement si,

$$T_0 < T_1 < T$$

Cette condition garantit le timeliness de la façon suivante : si nous supposons que le temps d'envoi par A de $Sub(k)$ est égal au temps de publication T_0, alors la condition $T_0 < T_1$, assure que A ne peut retenir le message $Sub(k)$ indéfiniment. La condition $T_1 < T$, assure tout simplement que les agents peuvent obtenir leurs preuves. En effet, les auteurs précisent que le délai entre T et T_1 doit être assez grand pour permettre aux agents le temps de communiquer avec le TTP.

Dans [8], les auteurs critiquent cette solution, citant qu'elle est inefficace, étant donné qu'elle dépend des temps réels, qui exigent un mécanisme de synchronisation entre les agents. Ces mêmes auteurs offrent une solution basée sur des durées et non des temps réels. Cette solution exige un petit changement dans la transmission des messages du protocole original de ZG. Dans ce protocole, B transmet maintenant $EOR(C)$ directement au TTP, ce qui permet à B de communiquer directement avec le TTP, ce qui n'était pas possible dans le protocole ZG. B peut maintenant transmettre une durée d'attente au TTP, mais elle lui permet aussi de tricher en retenant ce message.

Le nouveau protocole ZG ou NZG est décrit ci-dessous :

$$\begin{array}{lll} 1: & A \rightarrow B: & f_{EOO}, B, l, C, EOO \\ 2: & B \rightarrow TTP: & f_{EOR}, A, l, t_B, EOO, EOR \\ 3: & A \rightarrow TTP: & f_{Sub}, B, l, t_A, k, Sub, EOO \\ 4: & B \leftrightarrow TTP: & f_{Conf}, A, B, l, T, t_A, t_B, k, EOR, Conf \\ 5: & A \leftrightarrow TTP: & f_{Conf}, A, B, l, T, t_A, t_B, k, EOR, Conf \end{array} \tag{2.24}$$

où

$$\begin{array}{l} EOO = EOO(C) \\ EOR = EOR(EOO(C), t_B) \\ Sub = Sub(k, t_A, EOO) \\ Conf = Conf(k, T, t_A, t_B, EOO, EOR) \end{array} \tag{2.25}$$

On remarque que le message de l'étape 3 peut être transmis avant ou après l'étape 1, sans affecter le protocole. Les valeurs de temps t_A et t_B ne sont pas des temps réels mais des durées et sont définies ci-dessous. Le temps T quant à lui est une valeur de temps réel.

- t_A : défini par A et représente la durée pour laquelle le TTP gardera la soumission Sub.
- t_B : défini par B et représente la durée pour laquelle le TTP gardera EOR
- T : défini par le TTP et représente le temps actuel auquel le TTP rend $Conf$ disponible.

Selon ce protocole, le TTP génère Conf si et seulement si les valeurs de EOR reçues par A et B concordent. Ces agents supposent que le TTP publie $Conf$ pour t unités de temps. La valeur t devra être assez grande pour que A et B aient le temps de l'obtenir. En supposant que le réseau entre A, B et le TTP ne tombera pas en panne pour plus de t_d unités de temps, les auteurs fixent $t = t_d + x$, $(x > 0)$.

B peut tenter de tricher en retenant EOR pour une durée d. Mais d ne peut dépasser t_A, sinon le TTP effacera Sub de son répertoire. Donc B est forcé d'envoyer son EOR dans le délai prescrit, sans quoi il n'obtiendra rien. Par contre, il peut faire en sorte, que A ne peut obtenir sa preuve en bloquant l'accès de A au TTP. Mais selon la définition de t, il existera toujours une durée $x > 0$ dans laquelle A pourra contacter le TTP.

Par ailleurs, si B a déjà envoyé son EOR, A peut tenter de tricher en retenant le Sub pour un temps d. Encore une fois, d ne peut dépasser t_B, sinon A n'obtiendra rien. A peut aussi tenter de bloquer l'accès au TTP pour un temps t_d, encore une fois, il y aura toujours une durée $x > 0$ dans laquelle B pourra contacter le TTP.

Les auteurs ont présenté le protocole NZG, un protocole ZG modifié, selon lequel B communique directement avec le TTP, sans l'ajout de messages additionnels. Ceci permet aux

agents de publier des durées de vie pour les preuves. Les durées facilitent l'implantation de ce protocole, puisqu'elles ne nécessitent aucun mécanisme de synchronisation d'horloges entre les agents. On doit supposer par contre que la vitesse des horloges est semblable.

Le protocole de non-répudiation KM avec TTP offline

Les protocoles de non-répudiation avec TTP offline offrent une amélioration net sur le plan de la performance, mais leur complexité augmente en conséquence. Ils sont qualifiés de protocoles *optimistes* puisqu'ils supposent *a priori* que chaque agent agira honnêtement. Le TTP intervient alors seulement quand il y a eu tricherie ou un problème de communication.

Pour ce faire, les protocoles avec TTP offline consistent en au moins deux et au plus trois sous-protocoles que l'on décrit ci-dessous :

- un sous-protocole *principal* qui respectent les propriétés de timeliness et de viabilité.
- un sous-protocole de *récuperation*, qui permet aux agents d'obtenir leurs preuves si le sous-protocole principal est arrêté après une étape précise.
- un sous-protocole d'*avortement* qui permet à un agent d'arrêter le sous-protocole principal avant une étape précise du protocole sans perte.

Le protocole de Kremer et Markowitch [15] ou KM avec offline TTP consiste en ces trois sous-protocoles. Le sous-protocole principal est une version du protocole naif de l'équation 2.14, auquel a été ajouté des contrôles qui le protègent contre les comportements malhonnêtes.

$$
\begin{array}{l}
1 : A \rightarrow B : f_{EOO(C)}, f_{Sub(\{k\}_{PK_{ttp}})}, B, l, C, EOO(C), \{k\}_{PK_{ttp}}, Sub(\{k\}_{PK_{ttp}}) \\
2 : B \rightarrow A : f_{EOR(C)}, A, l, EOR(C) \\
\textbf{\textit{si A dépasse le délai d'attente alors avortement}} \\
3 : A \rightarrow B : f_{EOO(k)}, B, l, k, EOO(k) \\
\textbf{\textit{si B dépasse le délai d'attente alors récuperation}} \\
\textbf{\textit{avec [X :=B, Y :=A] sinon}} \\
4 : B \rightarrow A : f_{EOR(k)}, A, l, EOR(k) \\
\textbf{\textit{si A dépasse le délai d'attente alors récuperation}} \\
\textbf{\textit{avec [X :=A, Y :=B]}}
\end{array}
\tag{2.26}
$$

Le premier message contient des éléments nouveaux : l'encryptage de k par la clé publique du TTP et sa preuve de soumission. Ces éléments permettront au TTP de générer une confirmation de k en cas d'une récupération. Le fait d'encrypter k empêche B de l'intercepter et de tricher.

Si le sous-protocole principal se déroule sans interruption et que les agents reçoivent tous leurs messages, ils sont sûres de recevoir leurs preuves requises ($EOO(C)$ et $EOO(k)$ pour A et $EOR(C)$ et $EOR(k)$ pour B) dans un délai fixe. C'est dans les cas échéants que ce protocole à recours aux sous-protocoles d'avortement et de récupération.

Si après son premier message, A ne reçoit de réponse de B à l'intérieur d'un certain délai, il exécutera le sous-protocole d'avortement. Ce protocole fournira à A une confirmation de l'avortement généré par le TTP et empêchera B de poursuivre le protocole (en exécutant le protocole de récupération par exemple). Le protocole d'avortement est décrit ci-dessous.

$$\begin{array}{l} 1 : A \rightarrow TTP : f_{Avorte}, l, B, Avorte \\ \quad \textbf{\textit{si avorté ou recupéré alors stop}} \\ \quad \textbf{\textit{sinon avorté}} = \textbf{\textit{vrai}} \\ 2 : TTP \rightarrow A : f_{Conf}, A, B, l, Conf_a \\ 3 : TTP \rightarrow B : f_{Conf}, A, B, l, Conf_a \end{array} \tag{2.27}$$

Les variables *avorté* et *récupéré* (qui sont fixé à faux au début du protocole) empêchent l'exécution d'un sous-protocole si ce sous-protocole a déjà été exécuté.

Dans le protocole principal, si B ne reçoit pas la réponse attendue de A à l'étape 3, il exécutera le sous-protocole de récupération. De même, si A ne reçoit pas la réponse attendue de B à l'étape 4, il exécutera le sous-protocole de récupération. Ce protocole permet à l'agent exécutant X de fournir au TTP les preuves nécessaires pour générer la confirmation $Conf(k)$, qui sera ensuite mise à la disposition des agents.

Le sous-protocole de récupération est défini comme suit :

$$\begin{array}{l} 1 : X \rightarrow TTP : f_{Rec_X}, f_{Sub(k_{PK_{ttp}})}, Y, l, h(C), EOR(C), EOO(C), Sub(\{k\}_{PK_{ttp}}), Rec(x) \\ \textbf{\textit{si avorté ou récupéré alors stop}} \\ \textbf{\textit{sinon récupéré}} = \textbf{\textit{vrai}} \\ 2 : B \leftrightarrow TTP : f_{Conf(k)}, A, B, l, k, Conf(k) \\ 3 : A \leftrightarrow TTP : f_{Conf(k)}, A, B, l, k, Conf(k) \end{array} \tag{2.28}$$

où $Rec(x) = \{f_{Rec(x)}, Y, l\}_{SK_x}$ est une preuve de l'exécution du protocole de récupération par l'agent X .

Comme dans le protocole ZG, les agents A et B doivent communiquer avec le TTP pour obtenir la confirmation $Conf(k)$.

Les auteurs affirment que le protocole KM satisfait les propriétés d'équité et de timeliness.

Mais à cause de sa complexité, il est très difficile de pouvoir confirmer ceci sans études plus approfondies. Les méthodes de vérification formelle ont justement pour but de prouver qu'un protocole satisfait les propriétés voulues et donc est correct. Dans la prochaine section nous examinerons les méthodes spécifiques qui ont été utilisées pour analyser les protocoles MR, ZG et KM respectivement.

2.2.3 Un mécanisme pour valider les preuves de non-répudiation

Généralement, les preuves de non-répudiation sont construites à partir de signatures digitales. Nous rappelons qu'une signature digitale est un mécanisme cryptographique pour confirmer la source d'un message. Si PK_a par exemple, est la clé publique de A et SK_a sa clé privée, alors la signature digitale du message M est la valeur $\{M\}_{SK_a}$. Un receveur B peut vérifier que le message M été réellement généré par A, en s'assurant que $M = \{\{M\}_{SK_a}\}_{PK_a}$.

Avant de vérifier une signature digitale, il est primordial de savoir si la paire de clés publiques (PK_x, SK_x) est valide et donc que la signature est valide. Nous entendons par la validité d'une paire de clés publiques, le fait que la clé privée n'a pas été compromise par un agent malhonnête. Généralement, la clé publique d'un agent X est échangée parmi les agents, sous forme de certificat, dénoté $Cert_X$. Un certificat contient l'identification du propriétaire de la clé, la clé publique, une date d'expiration et la signature du tout par un Certificate Authority ou CA. La signature du CA valide non seulement la clé publique mais aussi la clé privée pour un temps donné. Donc, elle valide toute signature générée par la clé privée. On suppose que tous les agents ont confiance en le CA, sinon la validation n'a aucune valeur.

Un certificat est valide au moment où il est généré et reste normalement valide pour une durée fixée par le CA. Mais il est possible que le certificat devient invalide avant la date d'expiration, si par exemple la clé privée associée est considérée compromise. Dans ce cas, le CA doit révoquer le certificat. La $r$évocation d'un certificat consiste tout simplement à ajouter son nom à une $l$iste de révocation que chaque agent peut consulter, pour déterminer si une clé a été compromise.

Ce système de révocation est efficace, et assure que toute signature générée par une clé révoquée sera jugée invalide (si, bien sûr, les agents consultent la liste et que celle-ci est gardée à jour). Malheureusement ce système n'est guère utile pour juger de la validité de signatures digitales qui ont été générée avant qu'une clé ne soit révoquée. Ceci est d'une importance primordiale dans les protocoles de non-répudiation où la vérification d'une preuve par un juge peut être faite bien après sa génération. C'est le problème étudié par Zhou et al. dans [33].

Une solution simple et efficace à ce problème est d'ajouter une valeur de temps à chaque signature, ainsi un juge peut facilement déterminer la date de sa création. Cette valeur devra être générée par une autorité en qui tous les agents ont confiance. Malheureusement, un tel mécanisme ajoutera des étapes à un protocole de non-répudiation puisque chaque preuve de non-répudiation devra d'abord être envoyée à l'autorité avant de se rendre au TTP.

Pour éviter d'ajouter de nouvelles étapes à un protocole, Zhou et al. ont eu l'idée d'enchaîner des preuves. Par cette méthode, les preuves de non-répudiation sont liées l'une à l'autre et sont validées au moment où le TTP valide la confirmation. Elles resteront valides aussi longtemps que la signature du TTP reste valide. Les auteurs modifient le protocole ZG de l'équation 2.29, pour démontrer cette méthode, que nous décrivons ci-dessous :

$$\begin{array}{lll} 1: & A \rightarrow B: & f_{EOO(C)}, B, l, C, EOO \\ 2: & B \rightarrow A: & f_{EOR(C)}, A, l, EOR \\ 3: & A \rightarrow TTP: & f_{Sub(k)}, B, l, k_P K_{ttp}, EOR, Cert_B, Sub(k, EOR(C), Cert_B) \\ 4: & B \leftrightarrow TTP: & f_{Conf(k)}, A, B, l, k, T, Conf \\ 5: & A \leftrightarrow TTP: & f_{Conf(k)}, A, B, l, k, T, Conf \end{array} \tag{2.29}$$

où

$$\begin{array}{ll} EOO = & \{f_{EOO(C)}, B, l, C)\}_{SK_a} \\ EOR = & \{f_{EOR(C)}, A, l, C, EOO\}_{SK_b} \\ Sub = & \{f_{Sub(k)}, A, B, l, k, EOR, Cert_B\}_{SK_a} \\ Conf = & \{f_{EOO(k)}, A, B, l, k, EOR, T\}_{SK_{ttp}} \end{array} \tag{2.30}$$

Ici, $Cert_B$ représente le certificat de l'agent B, qui sera utilisé pour vérifier EOR. La valeur T est le temps de génération du message $Conf$, et $k_P K_{ttp}$ représente l'encryption de la clé k par la clé publique du TTP. Cet encryption est nécessaire pour éviter qu'un agent B malhonnête obtient un avantage, si EOR est rendu invalide, en révoquant son certificat $Cert_B$ avant que le TTP génère Conf. Nous décrivons les étapes du processus de validation pour tous les agents ci-dessous.

Après avoir reçu EOO, B se servira de $Cert_A$ pour vérifier la signature. B n'a pas à vérifier la validité de $Cert_A$. Si la signature est vérifiée, B transmettra EOR à A. Celui-ci à son tour vérifiera la signature EOR avec $Cert_B$. Si la signature est vérifiée, A générera et transmettra Sub (qui inclut EOR et $Cert_B$). Comme pour B, A n'a pas à vérifier la validité de $Cert_B$.

Après avoir reçu Sub, le TTP fera les vérifications suivantes :

- Vérifie la validité de $Cert_A$.
- Utilise $Cert_A$ pour vérifier la signature Sub
- Vérifie la validité de $Cert_B$.

Le TTP générera $Conf$ seulement si les vérifications sont effectuées avec succès. Nous notons qu'il ajoute une valeur de temps à la confirmation ; le juge s'en servira pour déterminer la validité des preuves.

Par cette méthode, les preuves sont liées l'une à l'autre et il existe un seul point de validation maintenant au lieu de trois : la signature Conf du TTP. Donc si $Conf$ est validée, EOO et EOR sont automatiquement validées. Aussi, le fait que le TTP ajoute une valeur de temps à la preuve implique que le juge pourra vérifier la validité des preuves sans se soucier du fait qu'un certificat a été révoqué entre temps. En fait, puisqu'il avait confiance en le TTP au temps T, le juge n'a qu'à vérifier que $Cert_B$ est valide au temps T.

Ce résultat simplifie la tâche d'un juge énormément, puisqu'il pourra souvent être présenté avec des preuves générées il y a longtemps. Il est primordial par contre que le juge ait confiance en le TTP à ce temps T.

2.3 Vérification formelle des protocoles de non-répudiation

Avant d'utiliser un protocole, les agents participants doivent avoir des garantis que les propriétés que le protocole prétend satisfaire sont réellement satisfaites, donc qu'il est correct. La vérification formelle d'un protocole de sécurité consiste justement à tenter de prouver la correction du protocole utilisant une approche mathématique.

La vérification formelle consiste en deux approches générales. L'approche basée sur *les preuves*, qui consiste à prouver que le protocole respecte les propriétés utilisant une logique donnée et l'approche basée sur *les modèles*, qui consiste à construire un modèle du protocole et de vérifier s'il existe un état du modèle qui ne satisfait pas les propriétés.

Selon [14] ces deux approches partagent les même composantes de base :
- un langage de description du protocole.
- un langage de spécification des propriétés.
- un algorithme de vérification.

Dans une approche basée sur les preuves, la description du système consiste en un ensemble

de formules Γ d'une logique appropriée et la spécification des propriétés consiste aussi en une formule logique Φ. La vérification tente de construire une preuve que Γ implique Φ.

Dans une approche basée sur les modèles, le protocole est représenté par un modèle $\mathcal{M}$, tandis que la spécification est une formule logique Φ. La vérification consiste à déterminer si le modèle $\mathcal{M}$ satisfait la formule Φ. La plus grande contrainte de cette approche est que le modèle doit avoir un nombre fini d'états. Les modèles sont donc une abstraction souvent importante du système, incluant seulement les éléments qui nous intéressent.

2.3.1 Le model-checking

Le model-checking est une technique basée sur les modèles où la vérification est complètement automatisée. Étant donné un modèle $\mathcal{M}$ et une formule Φ, un model-checker indiquera automatiquement si le modèle vérifie la ou les propriétés. Le model-checking est surtout utile pour la vérification de systèmes qui ont un comportement concurrentiel et non déterministe. Ces comportements sont présents dans presque tous les protocoles de sécurité.

Un model-checker modélise un protocole de sécurité comme système concurrentiel. Le modèle est composé de plusieurs processus s'exécutant en parallèle et communiquant par des actions visibles, représentées par la transmission de messages ou à travers des variables partagées.

Le langage de description du protocole doit pouvoir exprimer les états dans lesquels les processus agents peuvent se trouver et comment ils communiquent. Plusieurs model-checkers utilisent un langage dits *à commande gardée* (guarded command langage). Dans ce langage, chaque processus agent est représenté par un ensemble de commandes de la forme :

$$i : guard_i(X) \rightarrow update_i(X, Y)$$

La commande i éxprime le fait que si l'ensemble des variables X satisfait la formule booléenne $guard_i$, alors le système transite à un état où l'ensemble des variables Y, contrôlées par l'agent, sont modifiées par la fonction $update_i$. Une étape d'exécution d'un tel modèle est définie comme suit : chaque joueur choisit une commande i dont la fonction $guard_i$ est vraie et l'exécute. Le nouvel état du système est la conjonction de l'effet de chaque fonction $update_i$ sur l'ensemble des variables.

La vérification consiste à s'assurer que les propriétés tiennent à chaque état du modèle. La spécification des propriétés requiert un langage logique qui permet de décrire l'état du

modèle à une étape donnée de son exécution. Cette logique est dite *temporelle*. LTL (Linear Temporal Language) [25] et CTL (Computation Tree Logic [10] en sont des exemples.

En CTL par exemple, la propriété d'équité 2.2.5 en terme de l'agent B, pourrait être formulée de la manière suivante [17] :

$$\forall[](NRO \rightarrow \exists <> NRR)$$

où NRO est une formule logique qui exprime le fait que B possède sa preuve d'origine et NRR une formule qui exprime le fait que A possède sa preuve de réception.

Cette formule CTL spécifie que pour chaque exécution du protocole ($\forall$), on a toujours ($[]$) que si B a sa preuve d'origine (NRO), il existe alors au moins une continuation de l'exécution du protocole ($\exists$) où A recevra éventuellement ($<>$) sa preuve de réception (NRR).

Il existe une grande variété de model-checkers, qui permettent la vérification de protocoles présentant des comportements variés. Dans la prochaine section, nous allons présenter deux model-checkers qui modélisent des comportements probabilistes et de jeux respectivement et décrire comment ils ont été utilisés pour analyser les protocoles MR et KM respectivement.

2.3.2 Model-checking probabiliste : vérification du protocole de non-répudiation MR par l'outil PRISM

Le but de l'article [19] de Lanotte, Maggiolo et Troina est de vérifier si le protocole MR de l'équation 2.18 est ε-équitable. Plus précisément, ils se servent du model-checker probabiliste PRISM [24] pour analyser l'effet de la variation de certains paramètres du protocole, comme par exemple le temps de transmission ou de déchiffrage et d'observer si l'équité est maintenu ou non.

Le protocole MR se caractérise surtout par le fait :

- qu'il permet des comportements probabilistes : l'initiateur A décide avec une certaine probabilité p d'envoyer soit un message bidon ou la vraie clé, tandis que B doit avec une certaine probabilité q, décider s'il répond ou non au message de A.
- qu'il place une contrainte sur les temps de réponse en fonction du temps de déchiffrement de B.

Les auteurs introduisent les paramètres suivants pour représenter ces caractéristiques.

- La probabilité p que A envoie la vraie clé.

- La probabilité q que B tente de tricher.
- L'intervalle de temps [ad,AD] dans lequel un message doit être reçu.
- L'intervalle de temps [dd,DD] dans lequel le déchiffrement doit être fait.

L'idée sera d'utiliser le model-checker PRISM [24], pour varier ces paramètres et d'examiner si l'équité probabiliste est toujours satisfaite. PRISM comprend un langage de commandes gardées qui permet de spécifier des contraintes probabilistes et des contraintes temporelles. Ces deux types de contraintes sont essentielles pour une modélisation complète du protocole MR.

PRISM modélise un protocole de sécurité par un ensemble de modules et variables globales. Les modules peuvent représenter des agents et des canaux de communications. Un module comporte des commandes gardées de la forme :

$$g \rightarrow \lambda_1 : u_1 + \ldots + \lambda_n : u_n$$

où g est un ensemble de contraintes sur les variables du système. Elles doivent être satisfaites pour que le module transite à un nouvel état, où l'on effectue une mise à jour u_i avec probabilité λ_i pour chaque $i = 1, \ldots, n$.

Une étape d'exécution du système consiste en l'exécution d'une commande de chaque module. Plusieurs états pourront en être le résultat, chacun avec une probabilité calculée d'après les probabilités de chaque mise à jours.

La spécification du protocole et des propriétés de non-répudiation avec PRISM

Le modèle PRISM du protocole de non-répudiation MR (on réfère le lecteur à l'équation 2.18) comporte trois modules : un module initiateur, un module receveur et un module canal de communication. Chacun comporte un ensemble de variables et de commandes gardées.

Dans le module de l'initiateur par exemple, les auteurs spécifient le fait que la réponse de B doit être reçue dans un délai inférieur à la limite AD par les deux commandes décrites ci bas.

$$\begin{array}{ll} (o = 2)\&(t + x \leq AD) \rightarrow & (o' = 3); \\ (o = 2)\&(t + x > AD) \rightarrow & (o' = 0); \end{array} \tag{2.31}$$

Ici, la variable o est la variable d'état de l'initiateur. La variable x représente le temps de déchiffrement et suit une distribution uniforme sur l'intervalle $[dd, DD]$. Tandis que la variable t représente le délai de réponse causé par le canal de communication et suit une distribution uniforme sur l'intervalle $[ad, AD]$.

La première commande de l'équation 2.31, spécifie que si l'initiateur a reçu le $EOR(C)$ ($o = 2$) et que le temps de déchiffrement plus le temps de transmission ne dépasse pas AD alors l'initiateur recevra une réponse valide de B ($o' = 3$). La deuxième commande spécifie que si l'initiateur a reçu le $EOR(C)$ et que le temps de déchiffrement plus le temps de transmission dépasse AD alors le protocole sera arrêté par l'initiateur ($o' = 0$).

Dans le module du receveur, les auteurs spécifient le fait que B peut tenter de tricher avec les trois commandes suivantes :

$$\begin{array}{rcll} (r=4) & \rightarrow & q : (r'=5) & +(1-q) : (r'=6)\&(x'=0); \\ (r=5)\&(l=0) & \rightarrow & v : (r'=6)\&(x'=1) & +v : (r'=6)\&(x'=2); \\ (r=5)\&(l=1) & \rightarrow & (r'=7) & \end{array} \tag{2.32}$$

r est la variable d'état du receveur et la variable l sert à indiquer si la vraie clé a été envoyée ou non.

La première commande spécifie que si B a reçu un message clé de l'initiateur ($r = 4$), le receveur peut tenter de tricher ($r = 5$) avec probabilité q. La deuxième commande spécifie que si l'initiateur a envoyé une fausse clé ($l = 0$) et le receveur tente de tricher alors le temps de déchiffrement x doit suivre une distribution uniforme v sur l'intervalle $[dd, DD]$ (dans cet exemple $DD = 2$). Si par contre l'initiateur a envoyé la vraie clé ($l = 1$) et que le receveur tente de tricher alors la troisième commande spécifie que le système transitera à un état où le receveur a gagné ($r = 7$).

Ces exemples démontrent la facilité par laquelle les comportements probabilistes et les contraintes de temps peuvent être spécifiés avec PRISM. PRISM permet aussi de spécifier les propriétés de non-répudiation utilisant le langage temporelle probabiliste PCTL [4]. Par exemple, si $r = 5$ représente l'état d'une exécution correcte, on peut spécifier que le protocole de non-répudiation finit toujours de façon équitable par la formule PCTL suivante :

$$P_{\geq 1}[true\mathcal{U}(r = 5)]$$

Cette formule spécifie que la probabilité de se rendre à l'état où $r = 5$ doit être plus grande ou égale à 1.

On peut aussi estimer la probabilité qu'un receveur triche. Donc si $r = 7$ est l'état dans leqel B a triché, alors la formule :

$$P_{\geq v}[true\mathcal{U}(r = 7)] \tag{2.33}$$

spécifie que la probabilité que B triche est plus grande ou égale à v.

Cette dernière formule PCTL permet aux auteurs d'analyser comment la variation des paramètres ad, AD, dd, DD, p et q affecte l'équité du protocole. Pour des valeurs précises de ces paramètres, les auteurs calculerons, par une vérification de PRISM, la valeur v pour laquelle la formule de l'équation 2.33 ne tient pas. Si $v > p$, on conclura que le protocole ne satisfait pas l'équité.

Parmi les nombreux tests qu'ils ont effectués, on en souligne deux. Ils ont étudié le cas où $ad + dd > AD$, donc le cas d'un réseau idéal. Comme on peut s'attendre, la probabilité v est égale à p. Ils ont aussi étudié le cas d'un réseau lent où $ad + dd \leq AD$. Dans ce cas, ils ont observé que la probabilité v peut dépasser p. Ce résultat n'est pas surprenant, puisque le protocole MR spécifie que le temps de déchiffrement minimal dd ne doit jamais dépasser le temps d'attente maximal AD.

Dans cette étude, les auteurs ont démontré la puissance de l'outil PRISM pour analyser un protocole ayant un comportement probabiliste. La modélisation par commandes gardées est intuitive et la formulation de la propriété d'équité simple. Les auteurs ont bien cerné les paramètres les plus importants à étudier et leurs tests ont touché les points critiques du protocole. Ils ont prouvé qu'une variation permise des paramètres du protocole MR ne peut compromettre le protocole.

2.3.3 Model-checking basé sur la théorie des jeux

Dans [17], Kremer et Raskin proposent une nouvelle approche de vérification basée sur la théorie de jeux. Cette approche repose sur les observations suivantes des auteurs :

- les agents d'un protocole de non-répudiation agissent en tant que joueurs, dont le but est de trouver une stratégie qui leur rapporte un gain quelconque (par exemple : obtenir une preuve sans en fournir une).
- Le protocole de non-répudiation représente lui-même une stratégie qui doit défendre les agents honnêtes contre toutes stratégies possible d'agents malhonnêtes.

Dans cet optique, les propriétés de non-répudiation peuvent être formulées en terme de stratégies. Par exemple, la propriété d'équité en terme de l'initiateur A peut être exprimée comme suit :

> *une coalition du receveur B et tous les canaux de communication n'a pas de stratégie qui permet à B d'obtenir son NRO sans que A ait une stratégie pour obtenir son NRR*

L'avantage d'une telle formulation est qu'il exprime, formellement et assez naturellement,

deux importantes caractéristiques des protocoles de non-répudiation : la coopération, c.-à-d. que certains joueurs peuvent coopérer pour en défaire un autre et l'état de conflit qui existe entre les agents A et B. On note que le TTP est un joueur un peu particulier, il ne peut coopérer ni être en conflit avec un joueur. En effet, le TTP a une stratégie unique : exécuter l'action spécifiée e par le protocole.

La spécification du protocole et des propriétés de non-répudiation avec MOCHA

Le protocole KM avec TTP offline (équation 2.26), comporte plusieurs sous-protocoles qui sont exécutés dans un ordre précis par les participants honnêtes. Les auteurs observent que les stratégies disponibles à un agent malhonnête sont limitées. En effet, puisque le contenu des messages transmis est protégé par une signature digitale, leur intégrité est garantie et donc tous les messages reçus seront bien formés. Aussi, puisque chaque message contient un identificateur l unique de l'exécution T, deux exécutions différentes ne peuvent être entrelacées. Donc selon les auteurs, la seule stratégie qui reste possible à un agent malhonnête est d'exécuter les sous-protocoles dans un ordre inattendu. Ils considèrent que le modèle de jeux est celui qui peut modéliser le mieux cette situation.

MOCHA est un model-checker qui comprend le langage de modélisation dit *à commande gardée de jeux* , et la logique temporelle ATL (Alternating-time Temporal Logic) [2] qui permet d'exprimer les propriétés en terme de stratégies.

Le langage à commande gardée de jeux permet d'associer à un joueur $A \in \mathcal{A}$ un ensemble de commandes de la forme :

$$guard(X) \rightarrow update(X, Y')$$

Cette commande spécifie que si l'ensemble de toutes les variables booléennes X satisfait à la formule booléenne *guard*, alors le système transite à un état où les variables Y contrôlées par le joueur A sont mise à jour par la fonction *update*. Une étape de l'exécution du modèle est définie comme suit : chaque joueur exécute une commande dont la fonction *guard* est évaluée vraie et le nouvel état est la conjonction de l'effet de chaque fonction *update* sur l'ensemble des variables contrôlées par les joueurs.

Les auteurs décrivent comment chaque joueur du protocole incluant A, B, le TTP et les canaux de communications peuvent être modélisés par ce langage. Pour les joueurs A et B, chaque action dépend de la connaissance de certains messages (clés, messages encryptés, etc.). Ainsi les fonctions *guard* sont conçues pour contraindre une action sur

la connaissance de ces éléments. Par exemple, pour spécifier l'encryptage d'un message M par un agent A, ils proposent la commande suivante :

$$M_a \wedge K_a \rightarrow C_a' := true$$

Cette commande spécifie que si l'agent A connaît le message M (la variable M_a est vraie), et la clé k (la variable K_a est vraie) alors le modèle peut transiter à un état ou A connaît l'encryptage (la variable C_a devient vraie).

Pour l'envoi d'un message m, le joueur doit connaître tous les messages atomiques du message m. Ainsi, si le message m contient les messages atomiques l, t, C et $EOO(C)$ et les variables La, Ta, Ca et $EOOa$ représentent l'état des connaissances de A, par rapport à ces éléments, alors on spécifie l'envoie d'un message m par la commande :

$$La \wedge Ta \wedge Ca \wedge EOOa \wedge \neg STOPa \wedge \neg SENDm \rightarrow SENDm' := true;$$

C'est-à-dire que si A a les connaissances appropriées et n'a pas terminé son exécution ($\neg STOPa$) et que le message m n'a pas déjà été envoyé ($\neg SENDm$) alors A pourra transmettre le message.

Les auteurs associent un joueur à chacun des canaux de communications. Par exemple, le joueur *canal non-fiable* a deux actions possibles : transmettre ou ne pas transmettre. Le joueur *canal opérationnel* quant à lui, envoie toujours le message (donc une seule action). Le joueur canal résilient est un peu plus complexe, il doit inclure une action qui force la transmission après un certain délai.

Par exemple, la transmission d'un message (représenté par m) par le joueur canal opérationnel peut être spécifiée par la commande :

$$SENDm \wedge \neg m \rightarrow m' := true;$$

Le TTP quant à lui, doit exécuter les actions qui lui sont assignées dans l'ordre convenu. Les auteurs proposent alors la méthodologie suivante pour modéliser le TTP : chaque fois qu'une action est prise par le TTP, un numéro de séquence est incrémenté et le TTP doit effectuer l'action indiquée par le numéro de séquence et nulle autre.

Les auteurs suivent une méthodologie précise pour construire le modèle complet du protocole KM. Dans un premier temps les connaissances initiales de chaque joueur sont fixées ; on pense notamment aux connaissances des clés cryptographiques publiques et

symétriques. Ensuite on modélise les fonctions cryptographiques pour chaque joueur. Finalement, dans une troisième étape on modélise l'envoi et la réception de message.

Le résultat central de ce travail est la formulation des propriétés de non-répudiation en terme de stratégies. La logique ATL permet justement cette formulation et les auteurs fournissent quelques exemples en guise d'introduction. Supposons que l'ensemble des joueurs est $\mathcal{A} = \{a, b, c\}$, alors la formule

$$\ll a \gg <> p$$

spécifie que le joueur a a une stratégie pour atteindre un état ou la proposition p sera vraie éventuellement. Tandis que

$$\neg \ll b, c \gg []p$$

spécifie que la coalition des joueurs b et c n'ont pas de stratégie pour empêcher a d'atteindre un état où la proposition p est toujours vraie.

Ainsi, si la propriété d'équité en terme du joueur A formulée en stratégie est la suivante :

> *une coalition du receveur B et tous les canaux de communication n'a pas de stratégie qui leur permet d'atteindre un état où B a son NRO sans que A ait une stratégie pour obtenir son NRR*

alors, si Com représente le joueur canal de communication , la formule ATL équivalente est :

$$\neg \ll B, Com \gg <> (NRR \wedge \neg \ll A \gg <> NRO)$$

De même pour l'agent B , l'équité s'exprime en stratégie comme :

> *une coalition de l'initiateur A et tous les canaux de communication n'a pas de stratégie qui leur permet d'atteindre un état ou A a son NRR sans que B ait une stratégie pour obtenir son NRO*

et la formule ATL équivalente s'écrit de la façon suivante :

$$\neg \ll A, Com \gg <> (NRO \wedge \neg \ll B \gg <> NRR)$$

L'équité du protocole est donc spécifiée par la conjonction de ces deux formules. Les propriétés de timeliness et de viabilité peuvent aussi être exprimées par des formules ATL. On note que la formulation de ces propriétés pour le protocole KM doit être modifiée pour tenir compte des preuves des composantes. Donc, que posséder NRR est équivalent

à posséder le $EOR(C)$ et $EOR(k)$ ou $Conf(k)$. De même posséder la preuve NRO est équivalent à posséder les preuves $EOO(C)$ et $EOO(k)$ ou $Conf(k)$.

Les auteurs soulignent qu'il y a plusieurs avantages d'une formulation en ATL. Ce langage permet d'exprimer la coopération et la compétitivité qui existent entre les joueurs. Selon les auteurs le langage CTL par exemple ne permet pas cette expressivité.

Leur but est d'utiliser le model-checker MOCHA pour analyser la correction du protocole KM avec TTP offline en termes des propriétés d'équité, viabilité et timeliness. Le modèle est construit en tenant compte des suppositions suivantes :

- le canal de communication entre les agents A et B est non-fiable
- le canal de communication entre les agents et le TTP est résilient
- le contenu des messages ne peut être modifié
- deux exécutions ne peuvent être entelacées

Les deux premières suppositions viennent directement des concepteurs du protocole [15], tandis que les deux dernières représentent une simplification correcte étant donné la présence d'étiquettes et d'un identificateur d'exécution.

Les auteurs rapportent que l'outil MOCHA n'a trouvé aucune faille dans le protocole KM. Néanmoins, ils font l'observation intéressante que si le sous-protocole d'avortement est soustrait du protocole, alors la propriété de timeliness ne peut être respectée : A peut attendre un temps infini pour une réponse de B.

L'idée d'exprimer les propriétés de non-répudiation comme stratégies est complètement nouvelle et nous la croyons fondamentale. La formulation en stratégies se fait de façon naturelle et les raisonnements qui s'ensuivent sont très intuitifs. Nous remarquons aussi que contrairement à la plupart des articles, la vérification est complète, c-à-d que toutes les propriétés de non-répudiation sont comprises dans la vérification.

2.3.4 Les logiques de croyances

Contrairement au model-checking, la vérification basée sur les preuves ne cherche pas à vérifier chaque état possible du système, mais plutôt à construire une preuve que le système satisfait bien la ou les propriétés désirées. L'avantage est qu'elle évite les restrictions sur le nombre d'états. Son plus gros désavantage est que la construction des preuves peut s'avérer très difficile même si certains outils informatisés existent.

Il existe plusieurs méthodes de vérification basées sur les preuves qui ont été développées spécifiquement pour les protocoles de sécurité. Une des première fut la logique BAN [9] qui a été conçue pour les protocoles d'authentification. BAN a été conçue pour pouvoir raisonner sur les croyances des agents et comment ces croyances peuvent se modifier pendant l'exécution du protocole. En effet, à partir des croyances initiales et des croyances acquises le long de l'exécution, on tente de déterminer les croyances finales avec l'aide d'un ensemble restreint d'axiomes. Le protocole est jugé correct, si les croyances finales des agents satisfont la ou les propriétés désirées de l'authentification.

Plusieurs extensions à la logique BAN on été développées pour élargir son expressivité. La logique [29] a été conçue pour rassembler toutes les extensions en un langage compact mais puissant.

Dans la prochaine section, nous allons analyser comment les auteurs Zhou et Gollman ont utilisé la logique *SVO* pour tenter de vérifier leur propre protocole de non-répudiation ZG avec TTP online.

Vérification du protocole *ZG* par la logique SVO

Dans [35], Zhou et Gollman montrent comment les propriétés de non-repudiation peuvent être formulées dans le langage de la logique de croyance SVO et comment on peut utiliser cette même logique pour vérifier le protocole de non-répudiation *ZG* avec *TTP* online de l'équation 2.29. Il est important de noter que *SVO* a été conçue pour des types de protocoles de sécurités spécifiques : les protocoles d'authentification et d'échange de clés. Donc la vérification d'un protocole de non-répudiaton avec SVO représente un départ de l'intention originale de la logique.

La logique *SVO* consiste en deux règles logique *modus ponens* et *necessitation* (voir [14]) et 20 axiomes. Les auteurs se serviront de seulement 4 de ces 20 axiomes.

- **Axiome 1.** *Believing* : un agent croit tout ce qui suit logiquement de ses croyances.
- **Axiome 4.** *Source Association* : si la clé publique est disponible, tout agent qui reçoit un message M, peut en déduire l'identité de l'initiateur du message.
- **Axiome 7.** *Receiving* : un agent qui reçoit l'enchaînement de messages $X_1, \ldots, X_n$, reçoit aussi les éléments atomiques X_i $(1 \leq i \leq n)$.
- **Axiome 14.** *Saying* : un agent qui a dit l'enchaînement de messages $X_1, \ldots, X_n$, a aussi dit et vu les éléments atomiques X_i $(1 \leq i \leq n)$.

Puisque le protocole *ZG* n'est pas un protocole d'authentification ou d'échange de clés, il y a des éléments qui ne sont pas abordés par les axiomes. On pense notamment au rôle

du TTP, qui construit une base de données et la rend disponible aux agents, mais aussi aux étiquettes qui assurent l'identification de l'encryptage C de M et la clé k. Les auteurs devront développer des règles spécifiques pour ces éléments.

La vérification d'un protocole par la logique SVO consiste à construire une série de prémisses qui nous permettront, à partir des axiomes, d'établir les croyances des agents à la fin du protocole. Pour vérifier si ces croyances satisfont les propriétés, on formule les propriétés en terme de croyances, que l'on appelle *buts génériques*. Ceci permet de déterminer par simple comparaison, si les croyances générées par la preuve satisfont les propriétés.

Un des résultats originaux de ce travail est le fait que les propriétés et les buts génériques sont spécifiés en terme des croyances du juge indépendant. Dans cet étude, les auteurs considèrent seulement les propriétés de non-répudiation d'origine et de réception (définition 2.2.3). En terme des croyances du juge celles-ci s'énoncent comme suit :

> *à la fin du protocole, le juge doit croire que l'agent A a dit le message M et que B a reçu le message M.*

On note que les preuves ne sont pas mentionnées dans cette formulation. Elles sont remplacées par les croyances du juge. Ceci simplifie grandement la formulation de ces propriétés.

La verification avec SVO doit suivre un nombre d'étapes précises [29]. Nous les détaillons ci bas.

1. Développer des prémisses initiales à partir des spécifications du protocole. Par exemple : des prémisses sur les croyances en les clés publiques des agents participants.
2. En supposant que chaque agent reçoit les messages qui lui sont envoyés, développer des prémisses sur la réception de ces messages.
3. Développer des prémisses sur ce que chaque agent croit qu'il a reçu.
4. Développer des prémisses sur la signification que chaque agent associe aux messages qu'il croit avoir reçus.
5. Utiliser la logique pour établir les croyances des agents à la fin du protocole.

Comme nous sommes intéressés en les croyances du juge J, les prémisses seront construites en termes de ses croyances. Dans la première étape les auteurs formulent les trois prémisses suivantes basées sur la croyance du juge J dans les clés publiques des agents :

$$\begin{array}{ll} P1. & \text{J believes } PK_a \\ P2. & \text{J believes } PK_b \\ P3. & \text{J believes } PK_{ttp} \end{array} \tag{2.34}$$

Selon les auteurs, on peut supposer que le juge J reçoit, les messages reçus par chaque agents. Donc, après l'exécution du protocole, J aura reçu les messages

$$f_{EOO}, f_{EOR}, f_{CON}, A, B, l, M, C, k, EOR(C), Conf(k)$$

Dans la deuxième étape, ils élaborent donc deux nouvelles hypothèses basées sur cette supposition.

Dans l'étape 3, les auteurs construisent trois prémisses basées sur ce que J croit avoir reçu. Elles sont basées sur le fait que puisque J croit en les clés publiques, il peut vérifier les signatures des preuves qu'il a reçu. Dans la logique SVO, la notation $SV(X, k, Y)$ signifie qu'en appliquant la clé publique k au message X, on confirme que X est le résultat de la signature de Y avec la clé privée associée à k. Ainsi ils développent trois nouvelles prémisses basées sur la signature des messages reçu.

$$\begin{array}{ll} P1. & \text{J believes } SV((f_{EOO(C)}, B, l, C, EOO(C)), PK_a, (f_{EOO(C)}, B, l, C)) \\ P2. & \text{J believes } SV((f_{EOR(C)}, A, l, C, EOR(C)), PK_b, (f_{EOR(C)}, A, l, C)) \\ P3. & \text{J believes } SV((f_{Conf(k)}, A, B, l, k, Conf(k)), PK_{ttp}, (f_{Conf(k)}, A, B, l, k)) \end{array} \quad (2.35)$$

Les auteurs ont maintenant besoin d'introduirent quatre prémisses pour pouvoir raisonner sur le rôle du TTP et de l'encryptage symétrique. Ces éléments ne sont pas inclus dans la logique SVO et donc ces prémisses représentent des extensions hors du cadre conceptuel de la logique SVO.

Pour la neuvième prémisse, les auteurs formule une première extension de la logique. En se basant sur la supposition que tout message reçu par le TTP correspond à une soumission (f_{sub}) . Le fait que le TTP peut dire les messages (A, B, l, k) implique nécessairement selon les auteurs que A a dit (A,B,l,k) et que B a reçu (A, B, l, k) (ils présument que le canal de communication entre agents et TTP est opérationnels).

Une deuxième extension de la logique est la suivante : pour prouver que B a reçu un message, le juge doit vérifier la signature de la preuve de réception. Par contre l'axiome 4 permet seulement de prouver que B a dit le message et non de raisonner qu'il l'a reçu. C'est une grande distinction et pour la contourner, les auteurs doivent construire une prémisse qui spécifie que si B a dit $(f_{EOR(C)}, A, B, l, C)$ alors il a reçu (A, B, l, C).

Finalement, ils ont besoin de deux prémisses qui lient C avec k. Une première prémisse dit que si A a dit (A, B, l, C) et (A, B, l, k) alors il a dit M et une deuxième, de façon équivalente, dit que si B a reçu (A, B, l, C) et (A, B, l, k) alors il a reçu M.

Ces douze prémisses sont suffisantes pour prouver que les deux buts génériques sont satisfaits. Donc le protocole KM satisfait les propriétés de non-répudiation d'origine et de réception. Les auteurs n'ont pas tenté de vérifier les propriétés d'équités, timeliness ou de viabilité. Puisque la logique de SVO ne peut raisonner sur un agent malhonnête, ces propriétés semblent hors de portée de cette logique. Les auteurs soulignent que l'avantage de cette logique est la simplicité avec laquelle les propriétés de non-répudiation d'origine et de réception peuvent être exprimées en terme des croyances du juge.

Cet article représente une première tentative à utiliser une vérification basée sur preuves pour vérifier un protocole de non-répudiation. Étant donné les limites de l'expressivité du langage, nous nous demandons si cette approche est valide.

Chapitre 3

La correction des protocoles N-R avec TTP online

Résumé

Les protocoles de non-répudiation peuvent être conçus avec ou sans un tiers agent de confiance ou TTP. Un TTP offre un important avantage : les propriétés de non-répudiation sont souvent plus faciles à garantir. Cependant, dépendant de son niveau de participation dans le protocole, le TTP peut devenir un important point d'engorgement. Donc, il est essentiel que les avantages que le TTP offre en terme de sécurité ne nuisent pas à la performance du protocole.
Notre objectif dans ce chapitre est d'établir des conditions de correction pour un sous-ensemble des protocoles de non-répudiation avec TTP online. Nous démontrons que les conditions développées sont suffisantes pour garantir les conditions d'équité et de timeliness. Ces conditions seront des outils précieux pour la conception de protocoles corrects.

3.1 Introduction

Le TTP est un agent d'un protocole de non-répudiation qui, par ses actions, ne peut favoriser l'un ou l'autre des agents et, pour chaque exécution du protocole, doit exécuter les étapes qui lui sont assignées. Les rôles principaux d'un TTP sont, entres autres, d'acheminer des messages entre les agents, de confirmer la réception d'un message et de sa preuve de *soumission* en générant une preuve de *confirmation*, et de rendre cette confirmation disponible aux agents.

Les chercheurs ont établi plusieurs catégories de protocole de non-répudiation avec TTP,

qui sont en fonction du niveau de participation du TTP dans le protocole. Généralement, plus la participation du TTP est grande, moins le protocole sera performant. Les catégories, en ordre décroissant de participation du TTP, sont :

1. **Inline** - Le TTP participe à chaque transmission d'une exécution.
2. **Online** - Le TTP participe à chaque exécution mais pas à chaque transmission.
3. **Offline** - Le TTP participe seulement à l'exécution dans le cas de tricherie ou dans le cas d'une erreur de transmission.

Le niveau de participation du TTP a une conséquence importante non seulement sur la performance mais aussi sur la complexité du protocole. Ainsi un protocole avec TTP offline pourrait être beaucoup plus performant qu'un protocole avec TTP inline, mais peut s'avérer très complexe et sa correction sera difficile à vérifier.

Les protocoles de non-répudiation avec TTP online représentent un compromis entre les protocoles avec TTP inline et offline. Pour obtenir un protocole efficace et le moins complexe que possible, le TTP online intervient de façon passive, ainsi il ne peut par exemple initier un échange avec un agent, mais peut émettre un message à tous les agents ou répondre à des requêtes simples.

Dans le protocole de Zhang et Shi [34] par exemple, A transmet $C = \{M\}_k$ à B. Le TTP quant à lui, publie la valeur de k (transmis au TTP par A), à un moment décidé par A et B et dans un endroit accessible aux deux agents. En cas de dispute, les agents contacteront le TTP pour déterminer qui a triché. Cela implique malheureusement que le TTP devra garder une copie des clés pour un temps illimité. Un autre exemple est le protocole de Rabin [26], dans lequel le TTP émet aux agents un message contenant la clé de décryptage à des intervalles de temps régulier.

Dans le protocole Z-G [35], le TTP agit en tant que base de données de confirmation, de telle façon que si un agent lui envoi une preuve de soumission correcte, le TTP génèrera la confirmation et la rendra disponible aux agents qui lui en font la requête. On remarque que la version originale de ce protocole ne satisfait pas la propriété de timeliness. D'ailleurs, dans [8], les auteurs ont proposé une variante qui satisfait la propriété de timeliness, et qui ne compromet ni l'efficacité ni la simplicité du protocole original.

Dans ce chapitre nous établirons des conditions de correction pour une classe précise de protocoles de non-répudiation avec TTP online. Dans un premier temps, nous présenterons la définition et la syntaxe des protocoles de non-répudiation avec TTP online. Ensuite,

nous présenterons des conditions syntaxiques suffisantes pour garantir respectivement les propriétés d'équité et de timeliness. Finalement, nous présenterons un exemple concret prouvé correct dans la littérature et démontrerons qu'il satisfait à nos conditions.

3.2 Syntaxe de protocoles de non-répudiation avec TTP online

Un protocole de non-répudiation avec TTP online comprend exactement trois agents : l'initiateur A qui transmet le message M, le receveur B et un TTP online. La caractéristique fondamentale d'un TTP online est qu'il doit jouer un rôle passif dans le protocole, donc qu'il ne peut initier un échange mais répond plutôt à des requêtes. Il peut par exemple agir comme base de données, par laquelle les agents peuvent obtenir leurs items par un mécanisme de requête tel que ftp. Il peut aussi choisir de faire un 'broadcast' à un intervalle de temps régulier comme dans le protocole de Rabin [26].

Définition 3.2.1 (Les capacités du TTP online) *En plus des caractéristiques normales (il ne peut favoriser l'un ou l'autre des agents, doit exécuter les étapes qui lui sont assignées et peut confirmer la réception d'un message) un TTP online ne peut initier un échange avec un agent, doit minimiser le nombre d'échanges avec les agents et s'il génère une confirmation de la réception d'une preuve, il doit la publier à tous les agents.*

Puisqu'il y a trois agents dans un protocole de non-répudiation avec TTP online, on doit pouvoir préciser la destination de chaque message. Nous supposons aussi, que seul le ou les destinataires désignés recevront le message.

Définition 3.2.2 (Un message d'un protocole de non-répudiation) *Le message m_i^c transmi par un des agents A, B ou TTP à l'étape i du protocole, ne peut être reçu que par les agents éléments de $c \subseteq \{A, B, TTP\}$.*

Pour simplifier la notation, nous désignons par x le singleton $\{x\} \subseteq \{A, B, TTP\}$. Nous notons par m^{ftp}, le cas particulier d'un message m mis à la disposition de tous les agents par le TTP.

Nous présentons maintenant la syntaxe d'un protocole de non-répudiation avec TTP online.

Définition 3.2.3 (Un protocole de non-répudiation avec TTP online) *Un protocole de non-répudiation avec TTP online comprenant un initiateur A, un receveur B et*

un TTP, consiste en une séquence de n étapes de communications et la transmission du message M et est dénotée par $P^n_{A,B,TTP}(M)$.

Une étape i de $P^n_{A,B,TTP}(M)$ *peut être de deux formes :*

i. $X \rightarrow Y : {m_i}^z$ *où* $X \in \{A, B\}$, $Y \in \{A, B, TTP\}$ *et* $z \subset \{A, B, TTP\}$

ou

i. $X \leftrightarrow TTP : {m_i}^{ftp}$ *où* $X \in \{A, B\}$

Le symbole '$\leftrightarrow$*' représente une réponse du TTP à une requête de X.*

Nous associons à chaque étape de la forme i une valeur réelle T_i^X, qui représente le temps écoulé entre le début du protocole et l'étape i selon l'horloge de l'agent X.

Une session du protocole $P^n_{A,B,TTP}(M)$, est une exécution spécifique, soit partielle ou complète, de $P^n_{A,B,TTP}(M)$. Notons qu'une exécution partielle comprend s étapes ($s < n$) et est la conséquence d'un arrêt de l'exécution du protocole par l'un ou l'autre des agents à l'étape précise s. Un agent peut arrêter un protocol si, par exemple, il n'a pas reçu un message attendu en dedans d'un intervalle de temps t ou que le message reçu n'est pas de la forme exigée.

Un message m, transmis d'un agent A à un agent B, peut prendre plusieurs formes. Il peut être composé d'un seul message atomique (numéro de carte de crédit, texte, etc.), ou être une concaténation de plusieurs messages atomiques, donc $m = m_1.m_2$. Il peut aussi être un message encrypté dénoté par $\{m_1\}_{m_2}$, ou m_1 est un message et m_2 est la clé d'encryption.

Nous supposons que les agents participants à une session d'un protocole $P^n_{A,B,TTP}(M)$ ont les capacités habituelles en ce qui concerne l'encryption, le décryptage, la concaténation et la division de message. Par exemple, ils peuvent décrypter le message $C = \{m\}_k$ seulement s'ils possèdent la clé k. Ces capacités peuvent être définies plus formellement par la notion de fermeture d'un ensemble de messages. Nous présentons la définition de fermeture d'un ensemble M de messages ci-dessous :

Définition 3.2.4 (Fermeture) *Supposons que M représente un ensemble de messages. La fermeture de M, noté* $M \Downarrow$*, est défini comme étant le plus petit ensemble de messages qui satisfait les conditions :*

1. $M \subseteq M \Downarrow$
2. *Si* $k \in M \Downarrow$ *et* $\{m\}_k \in M \Downarrow$ *alors* $m \in M \Downarrow$

3. *Si* $k \in M \Downarrow$ *et* $m \in M \Downarrow$ *alors* $\{m\}_k \in M \Downarrow$
4. *Si* $m \in M \Downarrow$ *et* $m' \in M \Downarrow$ *alors* $m.m' \in M \Downarrow$
5. *Si* $m.m' \in M \Downarrow$ *alors* $m \in M \Downarrow$
6. *Si* $m.m' \in M \Downarrow$ *alors* $m' \in M \Downarrow$

L'enchaînement d'une séquence de messages est dénoté par l'opérateur ".". La séquence vide est dénotée par ϵ et nous avons $s.\epsilon = \epsilon.s = s$ et ce pour chaque séquence s. On utilise $s \sqsubseteq^{even} t$ pour dénoter une sous séquence préfixe t de longueur paire et de $s \sqsubseteq^{odd} t$ pour une sous séquence préfixe t de longueur impaire.

Nous avons aussi besoin du concept de l'ouverture de M défini comme étant l'ensemble des composantes de M.

Définition 3.2.5 (Ouverture) *Supposons que M représente un ensemble de messages. L'ouverture de M, noté $M \downarrow$, est défini comme étant le plus petit ensemble de messages qui satisfait les conditions :*

1. *Si* $m.m' \in M \downarrow$ *alors* $m \in M \downarrow$
2. *Si* $m.m' \in M \downarrow$ *alors* $m' \in M \downarrow$
3. *Si* $\{m\}_k \in M \downarrow$ *alors* $\{m, k\} \subseteq\in M \downarrow$

Le message M est rarement transmis en clair dans un protocole de non-répudiation avec TTP. Plutôt, on transmet une version encryptée de M suivie d'une ou plusieurs clés nécessaires pour son décryptage. Il est important pour ce type de protocole, que B ait reçu toutes les composantes pour pouvoir reconstruire M complètement ou même partiellement. Dans ce cas, la fonction d'encryption est dite *tout-ou-rien*. Nous définissons une fonction tout-ou-rien formellement comme suit :

Définition 3.2.6 (Fonction tout-ou-rien) *Une fonction bijective f est dite tout-ou-rien, si $M = f(p_1, \ldots, p_l)$ et nous pouvons reconstruire complètement ou partiellement M si et seulement si nous connaissons tous les paramètres p_i, $i = 1, \ldots, l$. Les valeurs p_i sont dites composantes de M.*

Nous supposons pour le reste de ce travail, que dans un protocole $P^n_{A,B,TTP}(M)$, l'agent A a accès à une fonction tout-ou-rien f, et ne transmet pas M, mais plutôt l'ensemble des composantes $\{p_1, \ldots, p_l\}$. Nous reprenons ici la définition de composantes de M en introduisant la notion de fonction tout-ou-rien.

Définition 3.2.7 (Composantes de M) *Si $M = f(c_1, \ldots, c_l)$, nous dénotons le fait que posséder M est équivalent à posséder les l composantes $\{c_1, ..., c_l\}$ par*

$$M \asymp \{c_1, ..., c_l\}$$

et nous dénotons par $\mathcal{C} = \{c_1, ..., c_l\}$, l'ensemble des composantes utilisées par un protocole.

Les preuves de non-répudiation NRR et NRO d'un protocole avec TTP sont une combinaison de preuves générées par l'un ou l'autre des agents et le TTP. Le TTP peut confirmer la réception d'un message appelé *soumission* en générant une preuve de *confirmation.* Cette idée est formalisée par la définition suivante :

Définition 3.2.8 (Preuve de soumission et de confirmation) *La preuve de soumission du message m au TTP, dénotée par $Sub(m)$, est la preuve d'origine du message m générée par l'agent X source du message. La confirmation d'une soumission, dénotée par $Conf(m)$, est la preuve que le TTP a reçu le message m et $Sub(m)$.*

La confirmation agit en tant que preuve d'origine et de réception du message m pour les agents A et B. On peut formaliser cette notion par la définition 3.2.9. Nous rappelons que les *connaissances* d'un agent acquises pendant une session du protocole $P^n_{A,B,TTP}(M)$ représentent l'ensemble des messages que chaque agent possède avant la session et les messages qu'il reçoit pendant la session. L'ensemble des connaissances d'un agent X à la fin de la session T du protocole $P^n_{A,B,TTP}(M)$ est dénoté par $\mathcal{K}_X(\mathsf{T})$.

Définition 3.2.9 (La confirmation) *Pour un protocole $P^n_{A,B,TTP}(M)$ et une exécution T donnée,*

1. $Conf(m) \in \mathcal{K}_A(\mathsf{T}) \Rightarrow EOO(m) \in \mathcal{K}_A(\mathsf{T})$ *et*
2. $Conf(m) \in \mathcal{K}_B(\mathsf{T}) \Rightarrow$
$EOR(m) \in \mathcal{K}_B(\mathsf{T})$

Nous définissons dans ce qui suit, les preuves de non-répudiation d'origine NRO et de réception NRR en terme de confirmation pour le cas où $\mathcal{C} = \{c_1, \ldots, c_l\}$.

Définition 3.2.10 (Preuves) *Supposons que $\mathcal{C} = \{c_1, \ldots, c_l\}$. Pour chaque session T, nous avons :*

$$\begin{cases} (EOR(c_1) \in \mathcal{K}_A(\mathsf{T}) \wedge EOR(c_2) \in \mathcal{K}_A(\mathsf{T}) \\ \wedge \ldots \wedge EOR(c_{l-1}) \in \mathcal{K}_A(\mathsf{T}) \wedge conf(c_l) \in \mathcal{K}_A(\mathsf{T})) \Rightarrow NRO \in \mathcal{K}_A(\mathsf{T}) \\ \quad et \\ (EOO(c_1) \in \mathcal{K}_B(\mathsf{T}) \wedge EOO(c_2) \in \mathcal{K}_B(\mathsf{T}) \\ \wedge \ldots \wedge EOO(c_{l-1}) \in \mathcal{K}_B(\mathsf{T}) \wedge conf(c_l) \in \mathcal{K}_B(\mathsf{T})) \Rightarrow NRR \in \mathcal{K}_B(\mathsf{T}). \end{cases} \tag{3.1}$$

3.3 Conditions garantissant l'équité

Nous proposons des conditions qui garantiront que tout protocole de non-répudiation avec TTP online satisfera les propriétés d'équité et de timeliness. Les conditions sont basées sur un mécanisme d'engagement-confirmation selon lequel un agent arrête son exécution de la session courante s'il n'a pas reçu une garantie de l'engagement de l'autre pour la session donnée et si l'engagement est reçu, il s'attend à obtenir une confirmation du TTP pour la dernière composante seulement.

3.3.1 Les conditions

Nous supposons que $M \asymp \{c_1, \ldots, c_l\}$, avec $l > 1$ et que l'initiateur A ne transmet aucune partie de M en clair, mais seulement les composantes.

La première condition assure que l'agent A transmet toujours la première composante directement au receveur B. Celle-ci sera utilisée pour obtenir l'engagement initial de B à l'exécution courante du protocole. Sans un engagement de B à l'exécution courante en forme d'une preuve de réception, A ne voudra pas poursuivre le protocole.

Condition 3.3.1 (Initialisation) *Supposons que* $M \asymp \{c_1, \ldots, c_l\}$. *Un protocole de non-répudiation* $P^n_{A,B,TTP}(M)$ *satisfait la condition d'initialisation si :*

$$c_1 \in m_i^b \downarrow, \quad 1 \leq i \leq n \tag{3.2}$$

Comme nous supposons qu'un agent peut faire une requête au TTP pour obtenir une information voulue, nous voulons limiter les composantes et preuves que le TTP reçoit pendant le déroulement du protocole. Ces limites sont placées par la deuxième condition.

Condition 3.3.2 (TTP online) *Supposons que* $M \asymp \{c_1, \ldots, c_l\}$. *Un protocole de non-répudiation* $P^n_{A,B,TTP}(M)$ *satisfait la condition TTP online si :*

$$\begin{gathered} \forall\, i \in [1,n] \;\; et \;\; k \in [1, l-1] : \\ EOR(c_k) \notin m_i^{ttp} \downarrow \;\; \wedge \;\; EOO(c_k) \notin m_i^{ttp} \downarrow \;\; \wedge \;\; c_k \notin m_i^{ttp} \downarrow \end{gathered} \tag{3.3}$$

La troisième condition garantit que la preuve d'origine $EOO(c_k)$, $k = 1, \ldots, l-1$ ou la soumission $sub(c_l)$, est transmise à B seulement si la composante c_k ou c_l a déjà été transmise à B. Cette condition assure un ordre plus intuitif à la transmission de la composante et de sa preuve d'origine.

Condition 3.3.3 (Composante en premier) *Supposons que* $M \asymp \{c_1, \ldots, c_l\}$. *Un protocole de non-répudiation* $P^n_{A,B,TTP}(M)$ *satisfait la condition de la composante en premier si :*

$$\forall\, i, j \in [1,n], j \leq i \;\; et \;\; k \in [1, l-1] :$$

$$\begin{cases} EOO(c_k) \in m_i^b \downarrow \;\Rightarrow\; c_k \in m_j^b \downarrow \\ sub(c_l) \in m_i{}^{ttp} \downarrow \;\Rightarrow\; c_l \in m_j{}^{ttp} \downarrow \end{cases} \tag{3.4}$$

Dans l'exemple ci-dessous, où le nombre de composantes $l = 2$, nous présentons une autre possibilité que celle imposée par la condition. Si A transmet c_1 et sa preuve d'origine au TTP et non à B, alors une confirmation de la première composante peut être obtenue par une requête faite au TTP. On remarque que dans ce cas, B pourra tricher en retenant le dernier message.

$$\begin{array}{lll} 1: & A \rightarrow TTP: & c_1, EOO(c_1) \\ 2: & A \leftrightarrow TTP: & c_1, conf(c_1) \\ 3: & B \leftrightarrow TTP: & c_1, conf(c_1) \\ 4: & A \rightarrow B: & c_2, EOO(c_2) \\ 5: & B \rightarrow A: & EOR(c_2) \end{array}$$

La quatrième condition assure que si la preuve de réception de c_k, $k = 1, \ldots, l-1$, est transmise à A alors la preuve d'origine de c_k a déjà été envoyée à B. Celle-ci impose que B donne son engagement au protocole à cette étape seulement s'il a déjà reçu celui de A.

Condition 3.3.4 (Initiateur honnête) *Supposons que* $M \asymp \{c_1, \ldots, c_l\}$. *Un protocole de non-répudiation* $P^n_{A,B,TTP}(M)$ *satisfait la condition de l' initiateur honnête si :*

$$\begin{array}{c} \forall\, i, j \in [1,n],\; j < i \;\; et \;\; k \in [1, l-1] \;: \\ EOR(c_k) \in m_i{}^a \downarrow \\ \Rightarrow EOO(c_k) \in m_j{}^b \downarrow \end{array} \tag{3.5}$$

Une autre possibilité que celle imposée par la condition 3.3.4 est présentée dans le protocole de l'équation ci-dessous avec $l = 2$. Ici, B envoie son engagement directement au TTP. Et dans ce cas, l'ordre de transmission des composantes par A sera moins important.

$$\begin{array}{lll} 1: & A \rightarrow TTP: & c_2, sub(c_2) \\ 2: & A \rightarrow B: & c_1, EOO(c_1) \\ 3: & B \rightarrow TTP: & EOR(c_1) \\ 5: & A \leftrightarrow TTP: & c_2, EOR(c_1), conf(c_2) \\ 6: & B \leftrightarrow TTP: & c_2, EOR(c_1), conf(c_2) \end{array}$$

Nous notons dans ce cas, que A doit procéder sans avoir reçu un engagement de B.

La prochaine condition concerne la transmission par B de la preuve de réception de c_k, $k = 1, \ldots, l-1$. Elle dicte que si A transmet c_k, pour $k = 2, \ldots, l$, alors nécessairement la preuve de réception de c_{k-1} doit être déjà été envoyée à A. Cette preuve représente l'engagement de B à l'étape et la session courante. Il est clair que A ne peut transmettre c_l à B.

Condition 3.3.5 (Receveur honnête) *Supposons que $M \asymp \{c_1, \ldots, c_l\}$. Un protocole de non-répudiation $P^n_{A,B,TTP}(M)$ satisfait la condition du receveur honnête si :*

$$\forall\, i, j \in [1, n],\ j < i \ \ et \ \ k \in [2, l-1] :$$

$$\begin{cases} c_k \in {m_i}^b \downarrow \ \Rightarrow \ EOR(c_{k-1}) \in {m_j}^a \downarrow \\ c_l \in {m_i}^{ttp} \downarrow \ \Rightarrow \ EOR(c_{l-1}) \in {m_j}^a \downarrow \end{cases} \tag{3.6}$$

La prochaine condition concerne encore la dernière composante et sa confirmation $conf(c_l)$. Cette condition assure que le TTP publie la confirmation seulement s'il a obtenu l'information nécessaire pour vérifier la soumission et de générer la confirmation. Elle assure aussi que le TTP ne transmet jamais la dernière composante sans transmettre sa confirmation et vice-versa.

Condition 3.3.6 (TTP honnête) *Supposons que $M \asymp \{c_1, \ldots, c_l\}$. Un protocole de non-répudiation $P^n_{A,B,TTP}(M)$ satisfait la condition du TTP honnête si pour chaque session T on a que :*

$$\forall\, i, j, k, r \in [1, n],\ i \leq j < k, r \ :$$

$$\begin{cases} (c_l \in {m_i}^{ttp} \downarrow \ \wedge \ sub(c_l) \in {m_j}^{ttp} \downarrow) \\ \Rightarrow (c_l \in {m_k}^{ftp} \downarrow \ \wedge \ conf(c_l) \in {m_r}^{ftp} \downarrow) \\ \\ c_l \in {m_i}^{ftp} \downarrow \ \Leftrightarrow \ \ conf(c_l) \in {m_j}^{ftp} \downarrow \end{cases}$$

Ces conditions définissent une classe de protocole de non-répudiation avec TTP online. Une caractéristique de cette classe est que le receveur B transmet les preuves $EOR(c_k)$, $k = 1, \ldots, l-1$ à A. Nous notons qu'il existe au moins une autre possibilité : B transmet ces preuves au TTP. Les conditions pour cette classe de protocoles seraient très semblables à celles définies ci haut.

Les conditions 3.3.1, 3.3.2, 3.3.3, 3.3.4, 3.3.5 et 3.3.6 imposent un ordre stricte à la transmission des messages. Par contre elles n'imposent pas que les paires de messages c_k et $EOO(c_k)$, $(k = 1, \ldots, l-1)$, c_l et $sub(c_l)$ et c_l et $conf(c_l)$ doivent être transmises dans le même message. L'équation 3.3.1 ci-dessous montre justement un exemple d'un protocole qui satisfait aux conditions.

$$
\begin{array}{rll}
1: & A \rightarrow B: & c_1 \\
\vdots & & \\
i_2: & A \rightarrow B: & EOO(c_1) \\
\vdots & & \\
i_3: & B \rightarrow A: & EOR(c_1) \\
\vdots & & \\
j_1: & A \rightarrow B: & c_{l-1} \\
\vdots & & \\
j_2: & A \rightarrow B: & EOO(c_{l-1}) \\
\vdots & & \\
j_3: & B \rightarrow A: & EOR(c_{l-1}) \\
\vdots & & \\
k_1: & A \rightarrow TTP: & c_l \\
\vdots & & \\
k_2: & A \rightarrow TTP: & sub(c_l) \\
\vdots & & \\
m_1: & A \leftrightarrow TTP: & c_l, conf(c_l) \\
\vdots & & \\
m_2: & B \leftrightarrow TTP: & c_l, conf(c_l)
\end{array}
$$

Ces conditions ne sont pas suffisantes pour garantir la propriété de timeliness. En effet dans l'exemple 3.3.1, A peut retenir le message c_l pour un temps illimité, ainsi, B, ne recevant pas de réponse du TTP (à l'étape m_2), pourrait conclure que la session est terminée et pourrait effacer tous les messages accumulés pendant la session, c'est à dire : c_k, $EOO(c_k)$ et $EOR(c_k)$, pour $k = 1, \ldots, l-1$. Si A reprend ensuite le protocole à l'étape k_1, il obtiendra la confirmation $conf(c_l)$ du TTP. Ayant effacé les messages qui lui ont été transmis, B aura été floué, même s'il peut communiquer avec le TTP.

La dernière condition assure qu'un agent ne peut manipuler le temps de la transmission des messages pour tricher. Pour cela, elle stipule que le temps maximal d'attente du TTP

pour recevoir c_l et $sub(c_l)$ ne doit pas dépasser le temps maximal d'attente de A et de B pour les éléments c_l et $conf(c_l)$ venant du TTP.

Supposons que T_{ftp} représente le délai maximal depuis le début du protocole qu'un agent attendra une réponse du TTP, et T_{ttp} le délai maximal depuis le début du protocole que le TTP attendra pour recevoir c_l et $sub(c_l)$. Reprenons notre protocole générique en notant les temps de transmission entre parenthèses à côté de chaque numéro d'étape.

$$
\begin{array}{rll}
1\ (T_{A_1}): & A \rightarrow B: & c_1 \\
\vdots & & \\
i_2: & A \rightarrow B: & EOO(c_1) \\
\vdots & & \\
i_3: & B \rightarrow A: & EOR(c_1) \\
\vdots & & \\
j_1: & A \rightarrow TTP: & c_l \\
\vdots & & \\
j_2\ (T_{A_{j_2}}): & A \rightarrow TTP: & sub(c_l) \\
\vdots & & \\
k_1\ (T_{A_{k_1}}): & A \leftrightarrow TTP: & c_l, conf(c_l) \\
\vdots & & \\
k_2\ (T_{B_{k_2}}): & B \leftrightarrow TTP: & c_l, conf(c_l)
\end{array}
$$

Ainsi, si $T_{ftp} \geq T_{B_{k_2}}$ et $T_{ttp} \geq T_{A_{j_2}}$, la condition de timeliness s'exprime en terme de T_{ftp} et de T_{ttp} comme suit :

Condition 3.3.7 (Timeliness) *Le protocole $P^n_{A,B,TTP}(M)$ satisfait la propriété de timeliness si :*

$$T_{ttp} \leq T_{ftp} \tag{3.7}$$

Nous démontrons maintenant que les conditions sont suffisantes pour garantir la propriété de l'équité et de timeliness. Pour cela, nous devons supposer que le canal de communication entre A, B et le TTP est, soit résilient ou opérationnel.

Proposition 3.3.8 *Supposons que le canal de communication entre A, B et le TTP est soit résilient ou opérationnel. Si les conditions 3.3.1, 3.3.2, 3.3.3, 3.3.4, 3.3.5, 3.3.6 et 3.3.7 sont satisfaites alors le protocole de non-répudiation $P^n_{A,B,TTP}(M)$ est équitable.*

Preuve:

Nous rappelons que par la définition des composantes et des fonctions tout-ou-rien, si $M \asymp \{c_1, \ldots, c_l\}$ alors pour chaque session T, nous avons :

$$\begin{cases} (EOR(c_1) \in \mathcal{K}_A(\mathsf{T}) \wedge EOR(c_2) \in \mathcal{K}_A(\mathsf{T}) \\ \wedge \ldots \wedge EOR(c_{l-1}) \in \mathcal{K}_A(\mathsf{T}) \wedge conf(c_l) \in \mathcal{K}_A(\mathsf{T})) \Rightarrow NRO \in \mathcal{K}_A(\mathsf{T}) \\ \text{et} \\ (EOO(c_1) \in \mathcal{K}_B(\mathsf{T}) \wedge EOO(c_2) \in \mathcal{K}_B(\mathsf{T}) \\ \wedge \ldots \wedge EOO(c_{l-1}) \in \mathcal{K}_B(\mathsf{T}) \wedge conf(c_l) \in \mathcal{K}_B(\mathsf{T})) \Rightarrow NRR \in \mathcal{K}_B(\mathsf{T}). \end{cases} \tag{3.8}$$

Ainsi, nous devons montrer que si les conditions tiennent, alors pour chaque session T du protocole, on a soit :

$$\begin{cases} NRR \in \mathcal{K}_A(\mathsf{T}) \ \wedge \ NRO \in \mathcal{K}_B(\mathsf{T}) \ \wedge \ M \in \mathcal{K}_B(\mathsf{T}) \text{ ou} \\ NRR \notin \mathcal{K}_A(\mathsf{T}) \ \wedge \ NRO \notin \mathcal{K}_B(\mathsf{T}) \ \wedge \ M \notin \mathcal{K}_B(\mathsf{T}) \end{cases} \tag{3.9}$$

Comme nous supposons que les canaux de communications sont soit résilients ou opérationnels, un message transmis se rendra toujours à sa destination. Donc il est impossible que le réseau de communication empêche une session de satisfaire l'équation 3.9. Il nous reste alors à vérifier que les agents ne peuvent tricher.

Premièrement, nous allons démontrer que dans le modèle imposer par les conditions, A ne peut tricher. Il n'existe que deux scénarios par lesquels A peut tricher :

- A tente de tricher en ne transmettant pas un ou plusieurs messages
- A tente de tricher en retenant un ou plusieurs messages pour un temps donné après quoi il le transmet.

Pour le premier scénario, nous allons procéder par contradiction. Donc supposons que $P^n_{A,B,TTP}(M)$ satisfait les conditions 3.3.1, 3.3.3, 3.3.4, 3.3.5, 3.3.6 et 3.3.7 et que A peut tricher. Par l'équation 3.9, A a triché pour une session T de $P^n_{A,B,TTP}(M)$ si

$$\begin{cases} (EOR(c_1) \in \mathcal{K}_A(\mathsf{T}) \ \wedge \ \ldots \ \wedge \ EOR(c_{l-1}) \in \mathcal{K}_A(\mathsf{T}) \ \wedge \ conf(c_l) \in \mathcal{K}_A(\mathsf{T})) \\ \wedge (EOO(c_1) \notin \mathcal{K}_B(\mathsf{T}) \ \vee \ \ldots \ \vee \ EOO(c_{l-1}) \notin \mathcal{K}_B(\mathsf{T}) \vee \\ c_1 \notin \mathcal{K}_B(\mathsf{T}) \ \vee \ \ldots \ \vee \ c_l \notin \mathcal{K}_B(\mathsf{T}) \ \vee conf(c_l) \notin \mathcal{K}_B(\mathsf{T})) \end{cases} \tag{3.10}$$

Nous procéderons cas par cas.

- Cas 1. Supposons que $EOO(c_k) \notin \mathcal{K}_B(\mathsf{T})$ pour un $k \in [1, l-1]$ alors soit que B ou soit que le TTP n'a jamais reçu ces preuves (nous supposons que A pourrait faire une requête pour les obtenir du TTP). Donc,

$$EOO(c_k) \notin {m_i}^b \ \wedge \ EOO(c_k) \notin {m_i}^{ftp}, \forall i \in [1, n]. \tag{3.11}$$

Par la condition 3.3.4, si $EOO(c_k) \notin {m_i}^b$, alors

$$EOR(c_k) \notin {m_i}^a, \forall\, i \in [1, n]. \tag{3.12}$$

et par la condition 3.3.2,

$$EOR(c_k) \notin {m_i}^{ftp}, \forall\, i \in [1, n]. \tag{3.13}$$

Donc

$$EOR(c_k) \notin \mathcal{K}_A(\mathsf{T})$$

- Cas 2. Supposons maintenant que $c_k \notin \mathcal{K}_B(\mathsf{T})$ pour un $k \in [1, l-1]$, alors

$$c_k \notin {m_i}^b \downarrow \ \wedge\, c_k \notin {m_i}^{ftp} \downarrow, \forall i \in [1, n]. \tag{3.14}$$

mais si $c_k \notin {m_i}^b$, par la condition 3.3.3,

$$EOO(c_k) \notin {m_i}^b, \ \forall i \in [1, n]. \tag{3.15}$$

Utilisant le raisonnement de cas 1, il en suit que $EOR(c_k) \notin \mathcal{K}_A(\mathsf{T})$ pour un $k \in [1, l-1]$.

- Cas 3. Supposons maintenant que $c_l \notin \mathcal{K}_B(\mathsf{T})$, alors

$$c_l \notin {m_i}^b \ \wedge \ c_l \notin {m_i}^{ftp}, \forall i \in [1, n]. \tag{3.16}$$

Par la condition 3.3.6,

$$c_l \notin {m_i}^{ttp} \downarrow \ \vee\, sub(c_l) \notin {m_j}^{ttp}, \ \forall\, i, j \in [1, n]$$

Ainsi, par la définition de la confirmation $conf(c_l)$, nous en déduisons que le TTP ne peut générer la confirmation et donc par conséquence A ne peut l'obtenir.

- Cas 4. Finalement, supposons que $conf(c_l) \notin \mathcal{K}_B(\mathsf{T})$, alors nécessairement, $conf(c_l) \notin m_i^{ftp}$, $i \in [1,n]$. Ainsi par la condition 3.3.6,

$$c_l \notin {m_i}^{ttp} \downarrow \ \vee \ sub(c_l) \notin {m_j}^{ttp}, \forall\, i,j \in [1,n].$$

  De laquelle nous concluons encore que le TTP ne peut générer la confirmation.

Pour le deuxième scénario, nous supposons que les conditions tiennent et que A tente de tricher en retenant un ou plusieurs messages pour un temps t donné avant de le transmettre. Nous notons par ρ, le délai de transmission causé par un canal de communication opérationnel.

Nous remarquons premièrement, que retenir c_k ou $EOO(c_k)$ pour $k = 1, \ldots, l-1$, ne lui donnera aucun avantage. Dans chacun de ces cas, soit que B attendra le temps voulu, ou soit que B arrêtera l'exécution. Dans le premier cas, le protocole déroulera en satisfaisant les conditions 3.3.1, 3.3.2, 3.3.3, 3.3.4, 3.3.5 et 3.3.6 et donc l'équité sera satisfaite. Dans le deuxième cas, aucun des agents ne pourra obtenir un avantage à ce stade du protocole.

Supposons alors qu'il retient c_l pour un delai t, mesuré depuis le début du protocole, avant de le transmettre. La condition 3.3.7 nous contraint aux cas suivants :

- Cas 1 $(t+\rho) \leq T_{ftp}$ et $(t+\rho) \leq T_{ttp}$. Dans ce cas, le protocole se déroulera selon les conditions 3.3.1, 3.3.2, 3.3.3, 3.3.4, 3.3.5 et 3.3.6 et l'équité sera satisfaite.
- Cas 2 $T_{ttp} < (t+\rho) \leq T_{ftp}$. Dans ce cas, le TTP arrêtera son exécution avant d'avoir reçu les items de A. Donc la confirmation ne sera jamais générée et donc il sera impossible pour A de l'obtenir. Le fait que B ne reçoit aucune réponse du TTP ne donnera aucun avantage à A.
- Cas 3 $T_{ttp} \leq T_{ftp} < (t+\rho)$. Encore une fois, le TTP arrête son exécution avant d'obtenir quoi que ce soit de A, ainsi pas moyen pour A d'obtenir la confirmation.

Supposons maintenant que B peut tricher. Il y a deux scénarios par lesquels B peut tricher :

- B tente de tricher en ne pas transmettant un ou plusieurs messages
- B tente de tricher en retenant un ou plusieurs messages pour un temps donné après quoi il le transmet.

Pour le premier scénario, nous allons procéder par contradiction. Donc supposons

que $P^n_{A,B,TTP}(M)$ satisfait les conditions 3.3.1, 3.3.2, 3.3.3, 3.3.4, 3.3.5, 3.3.6 et 3.3.7 et que B peut tricher. B a triché pour une session T de $P^n_{A,B,TTP}(M)$ si

$$\begin{cases} (c_1 \in \mathcal{K}_B(\mathsf{T}) \wedge \ldots \wedge c_l \in \mathcal{K}_B(\mathsf{T}) \wedge \\ EOO(c_1) \in \mathcal{K}_B(\mathsf{T}) \wedge \ldots \wedge EOO(c_{l-1}) \in \mathcal{K}_B(\mathsf{T}) \wedge conf(c_l) \in \mathcal{K}_B(\mathsf{T})) \\ \wedge (EOR(c_1) \notin \mathcal{K}_A(\mathsf{T}) \vee \ldots \vee EOR(c_{l-1}) \notin \mathcal{K}_A(\mathsf{T}) \vee conf(c_l) \notin \mathcal{K}_A(\mathsf{T})) \end{cases} \tag{3.17}$$

Considérons les cas un par un.

- Cas 1. Supposons que $conf(c_l) \notin \mathcal{K}_A(\mathsf{T})$, alors nécéssairement, $conf(c_l) \notin m_i^{ftp} \downarrow, \forall i \in [1,n]$, et il en suit que $conf(c_l) \notin \mathcal{K}_B(\mathsf{T})$. Par la condition 3.3.6, nous en déduisons que

$$c_l \notin m_i^{ftp} \downarrow, \forall i \in [1,n]$$

.

- Cas 2. Supposons que $EOR(c_k) \notin \mathcal{K}_A(\mathsf{T})$, pour un $k \in [1, l-1]$, alors

$$EOR(c_k) \notin {m_i}^a \wedge EOR(c_k) \notin {m_j}^{ftp}, \forall i,j \in [1,n]. \tag{3.18}$$

Mais si $EOR(c_k) \notin {m_i}^a, \forall i \in [1,n]$, et si $k = l-1$, alors par la condition 3.3.5,

$$c_l \notin m_i^{ttp} \downarrow, \ \forall i \in [1,n]$$

.

Ainsi, par la définition de la confirmation, nous en déduisons que le TTP ne peut générer et donc publier la confirmation et ainsi, $conf(c_l) \notin m_i^{ftp}$, $\forall i \in [1,n]$. On en déduit que $conf(c_l) \notin \mathcal{K}_B(\mathsf{T})$.

Maintenant, si $EOR(c_k) \notin {m_i}^a, \forall i \in [1,n]$, et si $k \in [1, l-2]$, alors par la condition 3.3.5

$$c_k \notin m_i^b \downarrow, \ \forall i \in [1,n]$$

.

Pour le deuxième scénario, nous procédons encore par contradiction. Nous supposons que les conditions tiennent et que B tente de tricher en retenant le message $EOR(c_k)$, $k \in [1, l-1]$, pour un temps t avant de le transmettre.

Dans ce cas, soit que A arrête l'exécution ou qu'il attends le message et continue le

protocole. S'il arrête l'exécution, B ne pourra tricher, puisque selon l'ordre établie par les conditions, aucun des agents possèdent assez d'items pour obtenir un avantage. S'il continue et que les conditions 3.3.1, 3.3.2, 3.3.3, 3.3.4, 3.3.5 et 3.3.6 sont satisfaites, selon notre preuve pour le premier scénario, B ne pourra tricher

□

3.3.2 Exemple concret

Dans la section 2.2.2, nous avions présenté le protocole Z-G qui est un exemple d'un protocole online très efficace et d'une belle simplicité. Dans ce qui suit, nous allons démontrer que le protocole Z-G dans sa forme originale satisfait les conditions 3.3.1, 3.3.2, 3.3.3, 3.3.4, 3.3.5 et 3.3.6. Nous montrerons aussi une variante proposée par [16], qui satisfait la condition 3.3.7 et donc la propriété de timeliness.

Dans ce protocole, les agents utilisent des signatures digitales pour générer les preuves, donc on suppose que chaque agent X possède une clé privée SK_x pour signer et les clés publiques de tous les autres agents pour effectuer la vérification. Dans notre cas, ces clés sont PK_a, PK_b et PK_{ttp}.

Initialement, A et B s'échangent les preuves d'origine et de réception de C. Ensuite l'initiateur A envoie k et une preuve de sa soumission au TTP. Le TTP générera alors une confirmation de la réception de la clé, que A et B devront alors récupérer du TTP.

Le protocole est décrit ci-dessous :

$$\begin{array}{lll} 1: & A \rightarrow B: & f_{EOO(C)}, B, l, C, EOO(C) \\ 2: & B \rightarrow A: & f_{EOR(C)}, A, l, EOR(C) \\ 3: & A \rightarrow TTP: & f_{Sub(k)}, B, l, k, Sub(k) \\ 4: & B \leftrightarrow TTP: & f_{Conf(k)}, A, B, l, k, Conf(k) \\ 5: & A \leftrightarrow TTP: & f_{Conf(k)}, A, B, l, k, Conf(k) \end{array} \tag{3.19}$$

Les auteurs spécifient que pour obtenir les confirmations, les agents A et B devront exécuter une requête de type *ftp get* (indiqué par $\leftrightarrow$).

Les preuves du protocole sont énumérées ci-dessous et sont générées par des signatures digitales. Nous rappelons que f_x représente une étiquette identifiant le type de preuve et l identifie uniquement la session.

$$\begin{aligned} EOO(C) &= \{f_{EOO(C)}, B, l, C\}_{SK_a} \\ EOR(C) &= \{f_{EOR(C)}, A, l, C\}_{SK_b} \\ Sub(k) &= \{f_{Sub(k)}, A, B, l, k\}_{SK_a} \\ Conf(k) &= \{f_{EOO(k)}, A, B, l, k\}_{SK_{ttp}} \end{aligned} \tag{3.20}$$

Le protocole tel que présenté n'est pas sous une forme qui nous permettra de faire la vérification des conditions directement. En effet, les preuves sont sous formes de signatures digitales, tandis que les conditions de corrections sont formulées en termes de preuves partielles de réception et d'origine.

Avant de montrer notre résultat principal, il nous faudra démontrer que la connaissance d'une signature digitale d'un message m et un ensemble d'identificateur, est équivalente à la connaissance d'une preuve partielle de non-répudiation d'origine $EOO(m)$, de réception $EOR(m)$, de soumission $sub(m)$ ou de confirmation $conf(m)$.

Par la définition 2.2.9 des preuves partielles les valeurs $EOR(m)$, $EOO(m)$, $sub(m)$ et $conf(m)$ permettent à un juge ou au TTP de décider si la composante m a été envoyée par un agent spécifique durant une session T précise.

Dans les cas où deux messages sont reçus en désordre où deux sessions s'entrelacent, il devient difficile d'associer la signature digitale à la bonne session et à la bonne composante. Supposons par exemple que B soumet le message m et la signature digitale $\{m\}_{SK_A}$ au juge. Si le juge possède la clé publique PK_A, il pourra conclure que A a généré m et donc a envoyé m. Mais il ne pourra pas savoir automatiquement à quelle composante associer m ni à quelle session associer la preuve.

Pour prévenir de telles situations, en s'inspirant du travail de Syverson sur les protocoles fail-stop [30], les auteurs ont ajouté à chaque signature digitale les identificateurs suivants : une identification du receveur (A ou B), une identification unique de la session l et une fonction étiquette $f_{(m)}$ qui identifie le type de la preuve partielle (soit EOO, EOR, sub ou $conf$) et la composante associée à la preuve partielle (soit C ou k).

La proposition suivante démontre que la connaissance de la signature digitale de m et des identificateurs est équivalente à la connaissance de la preuve partielle de m designée par $f_{(m)}$.

Proposition 3.3.9 *Pour une session* T *du protocole Z-G, si*

- l *est un identificateur unique de la session* T

- $f_{r(m)}$ *identifie uniquement une preuve partielle de type* r *pour le message* m*, où* $r \in \{EOO, EOR, sub, conf\}$.
- $X \in \{A, B, TTP\}$ *est un identificateur unique de l'agent qui transmet le message.*

alors

$$\begin{cases} \{f_{EOO(C)}, B, l, C\} \in \mathcal{K}_B(\mathsf{T}) \wedge \{f_{EOO(C)}, B, l, C\}_{SK_A} \in \mathcal{K}_B(\mathsf{T}) \Rightarrow EOO(C) \in \mathcal{K}_B(\mathsf{T}) \\ \{f_{EOR(C)}, A, l\} \in \mathcal{K}_A(\mathsf{T}) \wedge \{f_{EOR(C)}, A, l, C\}_{SK_B} \in \mathcal{K}_A(\mathsf{T}) \Rightarrow EOR(C) \in \mathcal{K}_A(\mathsf{T}) \\ \{f_{sub(k)}, A, B, l, k\} \in \mathcal{K}_{TTP}(\mathsf{T}) \wedge \{f_{sub(k)}, A, B, l, k\}_{SK_A} \in \mathcal{K}_{TTP}(\mathsf{T}) \Rightarrow sub(k) \in \mathcal{K}_{TTP}(\mathsf{T}) \\ \{f_{conf(k)}, A, B, l, k\} \in \mathcal{K}_A(\mathsf{T}) \wedge \{f_{conf(k)}, A, B, l, k\}_{SK_A} \in \mathcal{K}_A(\mathsf{T}) \Rightarrow conf(k) \in \mathcal{K}_A(\mathsf{T}) \\ \{f_{conf(k)}, A, B, l, k\} \in \mathcal{K}_B(\mathsf{T}) \wedge \{f_{conf(k)}, A, B, l, k\}_{SK_A} \in \mathcal{K}_B(\mathsf{T}) \Rightarrow conf(k) \in \mathcal{K}_B(\mathsf{T}) \end{cases}$$

Preuve:

Le premier cas concerne l'agent B et la preuve d'origine $EOO(C)$. Selon la définition 2.2.9 des preuve partielles, une preuve $EOO(m)$ générée par A est associée à une composante m et une session T précise. Supposons que pour une session T

$$\{f_{EOO(C)}, B, l, C\} \in \mathcal{K}_B(\mathsf{T}) \wedge \{f_{EOO}, B, l, C\}_{SK_A} \in \mathcal{K}_B(\mathsf{T})$$

Nous devons démontrer que les identificateurs B, l et $f_{EOO(C)}$ sont suffisants pour associer uniquement une signature digitale à sa composante et sa session. Puisque B possède la clé publique de A, il peut vérifier que A a vraiment généré le message $\{f_{EOO(C)}, B, l, C\}$. Possédant ainsi les valeurs B et l générées par A, l'agent B peut alors conclure que le message C lui a été destiné et est associé uniquement à la session T. L'étiquette $f_{EOO(C)}$ quant à lui, lui permet de conclure que la signature digitale est une preuve d'origine pour le message C. Donc les identificateurs lui permet d'associer le message à une étape spécifique du protocole, ce qui etait l'intention de Syverson.

On peut procéder à une preuve similaire pour les quatre autres cas.

□

Cette proposition nous permet de réécrire le protocole ZG en terme des preuves partielles EOR, EOO, $conf$ et sub. Ceci simplifie la notation en éliminant les identificateurs et le protocole devient :

$$\begin{array}{lll} 1: & A \rightarrow B: & C, EOO(C) \\ 2: & B \rightarrow A: & EOR(C) \\ 3: & A \rightarrow TTP: & k, sub(k) \\ 4: & A \leftrightarrow TTP: & k, conf(k) \\ 5: & B \leftrightarrow TTP: & k, conf(k) \end{array}$$

Nous pouvons maintenant présenter notre résultat principal.

Proposition 3.3.10 *Le protocole* ZG *satisfait les propriétés d'équité et de timeliness.*

Preuve:

Nous allons démontrer que le protocole comme énoncé dans l'équation 3.3.2 satisfait les conditions 3.3.1, 3.3.2, 3.3.3, 3.3.4, 3.3.5 et 3.3.6. Avec une petite modification, elle satisfera aussi à la condition 3.3.7.

Tout d'abord nous observons que $l = 2$ et que $\mathcal{C} = \{C, k\}$, avec $c_1 = C$ et $c_2 = k$. Encore par simple observation, nous voyons que ni C, $EOO(C)$, et ni $EOR(C)$ sont transmis au TTP. Donc le protocole satisfait à la condition 3.3.2.

Puisque

$$C \in m_1 \downarrow \quad \text{et} \quad EOO(C) \in m_1 \downarrow \tag{3.21}$$

et

$$k \in m_3 \downarrow \quad \text{et} \quad sub(k) \in m_3 \downarrow . \tag{3.22}$$

les conditions 3.3.1 et 3.3.3 sont automatiquement satisfaites. Il en suit que les propriétés 3.3.4 et 3.3.5 sont aussi satisfaites puisque

$$EOR(C) \in m_2 \downarrow . \tag{3.23}$$

La condition 3.3.6 est satisfaite par le fait que $k \in m_3 \downarrow \ \wedge \ sub(k) \in m_3 \downarrow \ \wedge \ k \in m_{4,5} \downarrow \ \wedge \ conf(k) \in m_{4,5} \downarrow$. Aussi, par le fait que dans les étapes 4 et 5, k et sa confirmation $conf(k)$ sont dans le même message.

Comme il n'y aucun condition sur le temps, il est clair que tel que présenté, le protocole Z-G ne peut respecter la propriété de timeliness. Les auteurs Kim et al. [16], ont proposé une variante pour tenter de satisfaire la propriété de timeliness. Pour ce faire, ils introduisent trois paramètres de temps :

- T : temps réel après lequel, le TTP effacera les valeurs k et $conf(k)$.
- T_0 : temps réel auquel le TTP publie k et $conf(k)$.
- T_1 : temps réel défini par B, avant lequel A doit transmettre k et $sub(k)$ au TTP.

La modification du protocole consiste a avoir A générer et transmettre la valeur A. B génère et transmet T_1 et le TTP génère et transmet T_0.

$$\begin{array}{lll} 1: & A \rightarrow B: & f_{EOO(C)}, B, l, T, C, EOO(C) \\ 2: & B \rightarrow A: & f_{EOR(C)}, A, l, T_1, EOR(C) \\ 3: & A \rightarrow TTP: & f_{Sub(k)}, B, l, T, k, Sub(k) \\ 4: & B \leftrightarrow TTP: & f_{Conf(k)}, A, B, l, T, T_0, k, Conf(k) \\ 5: & A \leftrightarrow TTP: & f_{Conf(k)}, A, B, l, k, T, T_0, Conf(k) \end{array} \tag{3.24}$$

où

$$\begin{array}{ll} EOO(C) = & \{f_{EOO(C)}, B, l, T, C\}_{SK_a} \\ EOR(C) = & \{f_{EOR(C)}, A, l, T_0, C\}_{SK_b} \\ Sub(k) = & \{f_{Sub(k)}, A, B, l, T, k\}_{SK_a} \\ Conf(k) = & \{f_{EOO(k)}, A, B, l, T, T_0, k\}_{SK_{ttp}} \end{array} \tag{3.25}$$

Nous remarquons que B peut abandonner la session après l'étape 1 s'il n'est pas satisfait de la valeur T. Pour que les preuves soient acceptées par le juge, la condition suivante sur les paramètres de temps doit être satisfaite :

$$T_0 < T_1 < T \tag{3.26}$$

Donc le TTP garantie qu'il publiera k et $conf(k)$ avant la limite T_1 fixée par B. En choisissant T_1 qui respecte cette inégalité et en l'envoyant au TTP, B s'assure de deux choses :

- A ne peut attendre que B arrête son exécution avant d'envoyer k et $sub(k)$.
- Il y aura toujours un délai égal à $T - T_1 > 0$, en dedans duquel B peut effectuer sa requête avant que le TTP n'efface la confirmation.

Pour la condition 3.3.7, nous avions défini T_{ftp} comme étant le délai maximal depuis le début du protocole qu'un agent attendra une réponse du TTP et T_{ttp} comme étant le délai maximal depuis le début du protocole que le TTP attendra pour recevoir c_2 et $sub(c_2)$ de A. Supposons que le protocole débute au temps réel Y. Puisque T représente le temps maximal qu'un agent peut obtenir la confirmation du TTP, nous avons l'égalité $T_{ftp} = T - Y$. Aussi, T_1 représente le temps maximal que A doit transmettre $sub(k)$ ainsi, nous avons que, $T_{ttp} = T_1 - Y$. Ainsi, selon l'equation 3.26, la condition de Timeliness 3.3.7 est satisfaite.

□

3.4 Conclusion

Dans ce chapitre nous avons établi une syntaxe pour les protocoles de non-répudiation avec TTP online. Avec cette syntaxe, nous avons établi des conditions suffisantes pour garantir les propriétés d'équité et de timeliness. Les conditions forment une classe spécifique de protocole caractérisée par le fait que le receveur B n'envoie jamais de message au TTP. Les conditions pourraient être modifiées facilement pour former une classe de protocole où B transmet au TTP.

Chapitre 4

La correction des protocoles N-R sans TTP

Résumé

Les protocoles de non-répudiation sans TTP tentent d'offrir les services de non-répudiation sans l'intervention d'un TTP. Sans les garanties que peuvent fournir un TTP, il est primordial d'établir un mécanisme pour échanger le message M et les preuves sans que l'un ou l'autre des agents ne puisse obtenir un avantage. Un mécanisme possible consiste à envoyer M parmi des fausses copies de celui-ci, de telle façon à ce que le receveur ne peut déterminer s'il possède le vrai M, avant d'avoir transmis les preuves nécessaires à l'initiateur.
Dans ce chapitre nous allons établir des conditions de corrections pour les protocoles de non-répudiation sans TTP, en se servant de ce mécanisme de fausses copies. Nous démontrons que pour ce type de protocoles, les conditions syntaxiques développées sont suffisantes pour garantir une équité probabiliste et la propriété de timeliness. Ces conditions seront des outils précieux pour la conception de protocoles de non-répudiation sans TTP corrects.

4.1 Introduction

Un TTP dans un protocole de non-répudiation offre un important avantage quant à garantir les propriétés de non-répudiation. Puisqu'il doit exécuter chaque étape qui lui est désignée, il est assez aisé de concevoir un protocole qui garanti la propriété d'équité. On pense notamment aux protocoles de Coffey et Saidha [11] et le protocole Z-G [35]. Cependant il n'est pas aussi facile d'assurer que le protocole sera efficace, le TTP peut représenter un point d'engorgement important du protocole. Pour cette raison, les cher-

cheurs ont vu l'utilité de concevoir des protocoles de non-répudiation sans TTP. Il y a en fait plusieurs scénarios pour lesquels un protocole de non-répudiation sans TTP pourrait être avantageux. Pensons par exemple à un système offrant un service de non-répudiation avec le TTP où celui-ci peut être très sollicité ou même tomber en panne. Un tel système pourrait se servir d'un protocole sans TTP pour dépanner et continuer d'offrir le service de non-répudiation quand le TTP n'est pas online.

Dans ce chapitre nous établirons des conditions de correction pour les protocoles de non-répudiation sans TTP. Plus précisément, nous allons concevoir des conditions syntaxiques suffisantes pour garantir respectivement une équité probabiliste et la propriété de timeliness. Nous les qualifions de conditions suffisantes, puisqu'elles définissent une classe de protocoles correcte, et de conditions syntaxiques, puisqu'elles sont définies en terme de la syntaxe d'un protocole, donc en terme du contenu et l'ordre de transmission des messages.

Un protocole de non-répudiation sans TTP, se définit très naturellement comme jeux entre deux joueurs qui tentent d'obtenir un gain en terme de preuve de non-répudiation. Ainsi, nous empruntons des résultats de la théorie de jeux pour démontrer qu'un protocole satisfaisant les conditions est restreint à une équité probabiliste. Ceci n'est pas un nouveau résultat mais confirme dans un langage un peu plus intuitif, le résultat de Even et Yacobi [12].

Le chapitre comporte trois sections. Dans un premier temps, nous introduisons la syntaxe et les définitions des protocoles de non-répudiation sans TTP. Dans la deuxième section, nous présentons les conditions de correction pour les propriétés d'équité et de timeliness avec une preuve de leur suffisance. Dans la troisième section nous utiliserons ces conditions pour démontrer que le protocole M-R est ε-équitable. Finalement, dans la quatrième section nous démontrerons que sans TTP, un protocole satisfaisant les conditions ne peut faire mieux qu'une équité probabiliste.

4.2 Syntaxe des protocoles de non-répudiation sans TTP

Un protocole de non-répudiation sans TTP comprend exactement deux participants, l'initiateur A qui transmet le message M et le receveur B et consiste en une série de transmission entre A et B.

Ci-dessous, nous présentons la définition syntaxique formelle d'un protocole de non-répudiation sans TTP. Elle inclut la notion de temps d'attente maximal pour un message,

d'où un agent attendra un temps maximum t pour la réception d'un message, après quoi il arrêtera l'exécution.

Définition 4.2.1 (Syntaxe d'un protocole NR sans TTP) *Un protocole de non-répudiation sans TTP entre un initiateur A et un receveur B, comprenant n étapes de communication, la transmission du message M, et un temps d'attente maximal, est dénoté par $P^{n,t}_{A,B}(M)$. Il consiste en une séquence $a_1, \ldots, a_n$ de n d'étapes de communication de la forme :*

$$(i.A \rightarrow B : m_i) \tag{4.1}$$

ou

$$(i.B \rightarrow A : m_i) \tag{4.2}$$

Si à une étape $i.X \rightarrow Y : m_i$, le temps d'attente pour le message m_i dépasse t, l'agent Y arrêtera l'exécution du protocole.

Si l'on veut garantir une équité probabiliste, le message M ne peut être transmis dans sa forme originale. Pour prévenir cela, divers mécanismes sont utilisés. Par exemple, il existe une classe de protocoles de non-répudiation sans TTP, où l'initiateur transmet à chaque étape des informations partielles de M, ainsi la probabilité que le receveur puisse reconstruire M croit avec chaque message reçu. Il existe une autre classe de protocole, dans laquelle l'initiateur transmet l'encryption de M et la clé de décryptage. Pour garantir que le protocole est ε-équitable, la probabilité que le receveur détermine quand la vraie clé a été transmise ne doit pas dépasser ε. Pour ce travail, nous considérons une classe qui comprend cette dernière, dans laquelle l'initiateur ne transmet jamais M, mais plutôt les paramètres $\{p_1, \ldots, p_r\}$ d'une fonction tout-ou-rien $f(M)$, dites composantes de M.

Nous rappelons qu'à chaque composante p_i, on peut associer une preuve d'origine de non-répudiation notée $EOO(p_i)$ et une preuve de réception de non-répudiation notée $EOR(p_i)$.

Il est clair que si A envoie seulement les composantes de M avec leurs preuve d'origine, alors B pourra facilement tricher en n'envoyant pas une preuve de réception pour une des composantes. Pour réduire la probabilité que B puisse tricher, A peut aussi envoyer une ou plusieurs fausses composantes. Une fausse composante est un message qui ne contient aucune information de M et peut être distinguée de la vraie composante seulement par le fait que l'on ne peut reconstruire M avec elle. Nous définissons formellement une fausse composante comme suit.

Définition 4.2.2 (ième fausse composante de M) *Si $M = f(p_1, \ldots, p_r)$, où f est une fonction tout-ou-rien, un message m est la fausse composante de p_i, si et seulement si,*

m ne donne aucune information sur le message M et possédant p_i *et m, B peut distinguer la vraie de la fausse seulement par le fait que* $M \neq f(p_1, \ldots, p_{i-1}, m, p_{i+1}, \ldots, p_r)$.

A chaque vraie composante p_i, $i = 1, \ldots, r$, *on peut associer* z_i $(z_i \geq 0)$ *fausses composantes , que nous dénotons par* $\overline{p_{ij}}$, $j = 1, \ldots, z_i$.

Nous notons l'ensemble de toutes les composantes transmises par A par $\mathcal{C} = \{c_1, \ldots, c_l\} = \{p_1, \ldots, p_r\} \bigcup \{\overline{p_{11}}, \ldots, \overline{p_{1,r_1}}, \ldots, \overline{p_{r,1}}, \ldots, \overline{p_{r,z_r}}\}$.

Finalement, nous supposons qu'ayant reçu p_i ou $\overline{p_{ij}}$, un agent peut toujours identifier l'indice i à laquelle elle correspond.

Nous passons maintenant à la présentation des conditions de corrections pour la propriété de l'équité-ε.

4.3 Conditions garantissant l'équité-ε et le timeliness

Dans la section 2.2.1, nous avions défini les cinq principales propriétés des protocoles de non-répudiation : les propriétés de non-répudiation d'origine et de réception, la propriété d'équité, la propriété de timeliness et la propriété de viabilité. Du point de vue de la vérification, les propriétés de non-répudiation d'origine et de réception ne sont pas très intéressantes, puisque la transmission des preuves de NRO et NRR est suffisantes pour les garantir. Par contre, les propriétés d'équité, de timeliness et de viabilité sont plus complexes et subtiles, et la correction d'un protocole en terme d'une de celles-ci, est beaucoup plus difficile à vérifier.

Les conditions que nous développerons satisferont implicitement les propriétés de non-répudiation d'origine et de réception. Ainsi, notre preuve concentrera plutôt sur la démonstration que les propriétés d'équité probabiliste et timeliness tiennent.

De façon informelle, les conditions que nous développerons représentent des règles qui empêchent les agents de tricher avec une probabilité plus grande que ε, où ε représente une borne positive non nulle que l'on veut être aussi petite que possible. Elles sont de types syntaxiques, et donc sont formulées en termes du contenu des messages et de l'étape où un message est transmis. Plus précisément, les conditions sont écrites en termes des composantes de M et les preuves de réception et d'origine associées.

Les quatre premières conditions s'enchaînent de façon à imposer un ordre à la transmission

des messages, tandis que la cinquième impose une limite supérieure au temps de calcul de la fonction tout-ou-rien.

4.3.1 Les conditions

La première condition, dite la condition *composante en premier*, force l'initiateur A à transmettre la preuve d'origine $EOO(c_i)$, pour une composante c_i, seulement s'il a déjà transmis la composante c_i. Cette condition empêche aussi la transmission de la preuve avant la composante, ce qui n'est pas une condition nécessaire mais donne un ordre plus intuitif à la transmission des messages.

Condition 4.3.1 (Composante en premier) *Supposons que* $\mathcal{C} = \{c_1, ..., c_l\}$. *Un protocole de non-répudiation* $P^{n,t}_{A,B}(M)$ *satisfait la condition de la composante en premier si :*

$$\begin{aligned} &\forall i \in [1,n], j \in [1,n],\ j \leq i, k \in [1,l] \ : \\ &EOO(c_k) \in m_i \downarrow \Rightarrow c_k \in m_j \downarrow \end{aligned} \tag{4.3}$$

La deuxième condition, dite la *condition d'initialisation*, assure que le protocole commence toujours par la transmission de la première composante c_1. Sans cette condition, il n'y aura aucune garantie que le protocole débutera.

Condition 4.3.2 (Initialisation) *Supposons que* $\mathcal{C} = \{c_1, ..., c_l\}$. *Un protocole de non-répudiation* $P^{n,t}_{A,B}(M)$ *satisfait la condition d'initialisation si :*

$$c_1 \in m_i \downarrow,\ 1 \leq i \leq n \tag{4.4}$$

La troisième condition, dite la condition du *receveur honnête* concerne la transmission des preuves de réception par B. Elle reflète le comportement honnête d'un receveur ayant reçu une preuve d'origine. Elle dicte que si un receveur transmet une preuve de réception pour une composantes c_i, alors nécessairement, il y a eu transmission d'une preuve d'origine à une étape antérieure j par A.

Condition 4.3.3 (Receveur honnête) *Supposons que* $\mathcal{C} = \{c_1, ..., c_l\}$. *Un protocole de non-répudiation* $P^{n}_{A,B}(M)$ *satisfait la condition du receveur honnête, si :*

$$\begin{aligned} &\forall i \in [1,n], j \in [1,n],\ j < i \text{ et } k \in [1,l] : \ EOR(c_k) \in m_i \downarrow \\ &\Rightarrow EOO(c_k) \in m_j \downarrow \end{aligned} \tag{4.5}$$

à ce point, on remarque que ces trois conditions imposent un ordre de transmission à la première composante c_1 et également à ses preuves d'origine et de réception. Plus précisément, un protocole satisfaisant les premières trois conditions suivra l'ordre suivante : transmission de c_1, suivit par la transmission de la preuve d'origine $EOO(c_1)$ et finalement la transmission de $EOR(c_1)$. La quatrième et dernière condition imposera un ordre identique aux composantes $c_2, ..., c_l$ et leur preuves.

La condition 4, dite de la *prochaine composante* concerne les composantes $c_2, \ldots, c_l$ et dicte que si c_i, $i = 1, \ldots, l$ a été transmise, alors la preuve de réception de c_{i-1} a déjà été transmise à une étape antérieure. Cette condition assure que A ne peut envoyer une nouvelle composante sans avoir la preuve de réception de la précédente.

Condition 4.3.4 (Prochaine composante) *Supposons que* $\mathcal{C} = \{c_1, ..., c_l\}$. *Un protocole de non-répudiation* $P^n_{A,B}(M)$ *satisfait la condition de la prochaine composante si :*

$$\begin{aligned} \forall i \in [1,n], j \in [1,n],\ j < i \text{ et } k \in [2,l] :\ & c_k \in m_i \downarrow \\ \Rightarrow\ & EOR(c_{k-1}) \in m_j \downarrow \end{aligned} \tag{4.6}$$

Les quatre conditions assurent que chaque composante requiert au plus trois messages pour assurer la transmission des preuves. Par le fait que l'initiateur A génère $EOO(m)$ et que le receveur B génère $EOR(m)$, la forme générique d'un protocole $P^{n,t}_{A,B}(M)$, qui transmet le message M en l composantes, donc $\mathcal{C} = \{c_1, \ldots, c_l\}, l \leq n$, pourrait être de la forme suivante :

$$
\begin{array}{rll}
1: & A \rightarrow B: & c_1 \\
\vdots & & \\
i_2: & A \rightarrow B: & EOO(c_1) \\
\vdots & & \\
i_3: & B \rightarrow A: & EOR(c_1) \\
\vdots & & \\
j_1: & A \rightarrow B: & c_k \\
\vdots & & \\
j_2: & A \rightarrow B: & EOO(c_k) \\
\vdots & & \\
j_3: & B \rightarrow A: & EOR(c_k) \\
\vdots & & \\
k_1: & A \rightarrow B: & c_l \\
\vdots & & \\
k_2: & A \rightarrow B: & EOO(c_l) \\
\vdots & & \\
k_3: & B \rightarrow A: & EOR(c_l)
\end{array}
$$

avec $1 \leq i_2 < i_3 < j_1 \leq j_2 < j_3 < k_1 \leq k_2 < k_3$ et $l \leq n$.

Nous notons que dans le modèle ci-dessus, la composante et la preuve d'origine pourraient être combinées dans un seul message et les conditions seront toujours respectées.

La cinquième et dernière condition impose que le temps de calculer M à partir d'un ensemble de composantes doit toujours être plus grand que le temps entre la transmission d'une composante c_i et la réception de la preuve de réception de c_i. Ceci pour éviter que B puisse déterminer s'il a reçu toutes les vraies composantes sans arrêter le protocole.

Condition 4.3.5 (Timeliness) *Supposons que $M = f(p_1, \ldots, p_r)$ et que $C = \{c_1, ..., c_l\}$. Si $\mathcal{T}_M$ représente le temps nécessaire à B de calculer M à partir de $f()$, on doit avoir*

$$\forall i \in [1, l],\ EOR(c_i) \in m_k \downarrow\ \wedge\ c_i \in m_l \downarrow \Rightarrow\ T_k - T_l < \mathcal{T}_M \tag{4.7}$$

sinon A arrête le protocole.

Nous démontrons maintenant que les cinq conditions sont suffisantes pour garantir la propriété de l'équité-ε. Pour cela nous devons supposer que chaque message transmis sera reçu.

Proposition 4.3.6 *Supposons que le canal de communication entre A et B est soit résilient ou opérationnel. Si les conditions 4.3.1, 4.3.2, 4.3.3, 4.3.4 et 4.3.5 sont satisfaites alors le protocole de non-répudiation $P^{n,t}_{A,B}(M)$ est ε-équitable et respecte la propriété de timeliness.*

Preuve:
On propose que les conditions 4.3.1, 4.3.2, 4.3.3, 4.3.4 et 4.3.5 définissent une classe de protocoles ε-équitables.

Nous devons montrer que si les conditions tiennent alors pour chaque session T du protocole, on a que :

$$\begin{cases} NRR \in \mathcal{K}_A(\mathsf{T}) \ \wedge \ NRO \in \mathcal{K}_B(\mathsf{T}) \ \wedge \ M \in \mathcal{K}_B(\mathsf{T}) \text{ ou} \\ NRR \notin \mathcal{K}_A(\mathsf{T}) \ \wedge \ NRO \notin \mathcal{K}_B(\mathsf{T}) \ \wedge \ M \notin \mathcal{K}_B(\mathsf{T}) \end{cases} \tag{4.8}$$

avec probabilité $1 - \varepsilon$.

Comme nous supposons que les canaux de communications sont soit résilients ou opérationnels, un message transmis se rendra toujours à sa destination. Donc il est impossible que le réseau de communication empêche une session de satisfaire l'équation 4.8. Il nous reste alors à vérifier que les agents ne peuvent tricher avec probabilité supérieure à ε, $(0 < \varepsilon < 1)$.

Supposons que ψ et ϕ sont respectivement les probabilités que A et B trichent. Nous allons démontrer que si $P^{n,t}_{A,B}(M)$ satisfait les conditions 4.3.1, 4.3.2, 4.3.3, 4.3.4 et 4.3.5, alors $\psi + \phi < \varepsilon$, pour toute session T.

Il y a deux scénarios à vérifier :
- un agent qui tente de tricher en manipulant les messages
- un agent qui tente de tricher en manipulant le temps d'envoi des messages.

Nous allons étudier ces deux scénarios séparément.

Premièrement, nous allons démontrer que dans le modèle imposé par les conditions, A ne peut tricher en manipulant seulement les messages et donc que $\psi = 0$. Nous procédons par contradiction.

Supposons que $P^{n,t}_{A,B}(M)$ satisfait les conditions 4.3.1, 4.3.2, 4.3.3 et 4.3.4 et que $\psi > 0$ pour au moins une session T de $P^{n,t}_{A,B}(M)$. Si f() est une fonction tout-ou-rien et $M = f(p_1, \ldots, p_r)$, alors par la définition 2.2.4, A a triché pour une session T de $P^{n,t}_{A,B}(M)$ si

$$\begin{cases} EOR(p_i) \in \mathcal{K}_A(\mathsf{T}), \forall i \in [1, r] \\ (p_j \notin \mathcal{K}_B(\mathsf{T}) \text{ ou } EOO(p_j) \notin \mathcal{K}_B(\mathsf{T})), \ j \in [1, r] \end{cases} \tag{4.9}$$

Puisque $EOR(p_i) \in \mathcal{K}_A(\mathsf{T}), \forall i \in [1,r]$, et puisque B génère les preuves partielles de réception, on doit nécessairement avoir

$$\forall i \in [1,r],\ EOR(p_i) \in m_k \downarrow, k \in [1,n]$$

La condition 4.3.3, garantie alors la transmission des preuves d'origine respectives et donc nécessairement leur réception par B. C'est-à-dire

$$\begin{aligned} & EOR(p_i) \in \mathcal{K}_A(\mathsf{T}), \forall i \in [1,r] \\ \Rightarrow\ & EOO(p_j) \in \mathcal{K}_B(\mathsf{T}), \forall j \in [1,r] \end{aligned} \tag{4.10}$$

Il s'en suit de la condition 4.3.1

$$p_j \in \mathcal{K}_B(\mathsf{T}), \forall j \in [1,r] \tag{4.11}$$

De la définition 2.2.8 et les équations 2.13 et 2.12, nous déduisons que $\{NRR, M\} \subseteq \mathcal{K}_B(\mathsf{T})$. Ceci contredit la supposition initiale que l'initiateur A peut tricher et nous concluons que $\psi = 0$.

Supposons maintenant que A tente de tricher en manipulant le temps d'envoie des messages. Sa seule option est de retenir au moins un message m_i , $1 < i < n$, pour un temps α avant de le transmettre. Nous supposons aussi que λ est égal au délai causé par le canal de communication.

Nous procédons par contradiction en supposant que les conditions tiennent et que $\psi > 0$. Supposons que A retient le message m_i pour un temps α , alors il y a deux cas à considérer.

- $\alpha + \lambda < t$: par la définition de t, le protocole se déroulera comme prévu selon les conditions.
- $\alpha + \lambda \geq t$: encore par la définition de t, B arrêtera le protocole à l'étape i, sans obtenir le message. Le résultat est donc équivalent à celui où A ne transmet pas le message m_i et selon le premier scénario, A ne peut tricher en retenant un message.

Nous concluons que l'on a encore $\psi = 0$.

Nous allons prouver maintenant que la probabilité ϕ que B triche satisfait $\phi < \varepsilon$ ($0 < \varepsilon < 1$) pour toute session T. Si f() est une fonction tout-ou-rien et $M =$

$f(p_1, \ldots, p_r)$, alors par la définition 2.2.4, B a triché pour une session T de $P^n_{A,B}(M)$ si

$$\begin{cases} (EOO(p_i) \in \mathcal{K}_B(\mathsf{T}) \text{ ou } p_i \in \mathcal{K}_B(\mathsf{T})), \forall i \in [1,r] \\ EOR(p_j) \notin \mathcal{K}_A(\mathsf{T}),\ j \in [1,r] \end{cases} \tag{4.12}$$

Encore ici, il y a deux scénarios à considérer.
- un agent qui tente de tricher en manipulant les messages
- un agent qui tente de tricher en manipulant le temps d'envoi des messages.

Nous commençons encore par le premier en procédant par contradiction.

Supposons que $p_i \in \mathcal{K}_B(\mathsf{T}), \forall i \in [1,r]$. Puisque A génère M et ses composantes, alors

$$\forall i \in [1,r],\ p_i \in m_k \downarrow,\ k \in [1,n]$$

Alors par la condition 4.3.4, nous déduisons que :

$$\forall i \in [1,r-1],\ EOR(p_i) \in m_k \downarrow,\ k \in [1,n]$$

et donc puisque tout message transmis est reçu,

$$EOR(p_i) \in \mathcal{K}_A(\mathsf{T}), \forall i \in [1,r-1] \tag{4.13}$$

De la même façon, supposons que $EOO(p_i) \in \mathcal{K}_B(\mathsf{T}), \forall i \in [1,r]$. Puisque A génère M et ses composantes, alors

$$\forall i \in [1,r],\ EOO(p_i) \in m_k \downarrow,\ k \in [1,n]$$

Alors par la condition 4.3.1, nous déduisons que :

$$\forall i \in [1,r],\ p_i \in m_k \downarrow,\ k \in [1,n]$$

et suivant la même logique que précédemment, nous déduisons que

$$EOR(p_i) \in \mathcal{K}_A(\mathsf{T}), \forall i \in [1,r-1] \tag{4.14}$$

Donc si $P^{n,t}_{A,B}(M)$ satisfait les conditions 4.3.1, 4.3.2, 4.3.3 , 4.3.4 et 4.3.5, nous pouvons conclure que si B reçoit toutes les composantes et les preuves associées, alors A recevra une preuve de réception pour chaque vraie composante sauf la dernière (p_r) avec probabilité 1. Ainsi, B peut tricher d'une seule et unique façon : en n'envoyant pas la preuve de réception $EOR(p_r)$.

Supposons que q est l'étape où A a transmis $EOO(p_r)$. Selon notre résultat, si B ne manipule pas le temps, il peut tricher d'une seule et unique façon : tenter de deviner la valeur de q avant d'envoyer $EOR(p_r)$. Mais, selon le modèle générique du protocole, A traite chaque composante de la même façon : ayant reçu une preuve de réception de B en dedans du délai garanti par la condition 4.3.5, il envoie toujours une composante et sa preuve. Donc il n'y a aucun indice externe qui peut indiquer à B quand une vraie ou une fausse composante a été transmise.

Puisque B peut toujours identifier l'indice d'une composante, la probabilité ϕ que B triche est uniquement fonction du nombre total z_r de fausses composantes envoyées avant la transmission de p_r. Donc, pour éviter que $\phi = 1$, A doit nécessairement envoyer au moins une fausse composante $\overline{p_{r1}}$ avant d'envoyer p_r.

Si B peut manipuler le temps d'envoi de ces messages, il pourra retenir la transmission de $EOR(p_r)$ ou de $EOR(\overline{p_{ri}})$, $i = 1, \ldots, z_r$, pour lui donner le temps de calculer f(). Mais la condition 4.3.5 garantit que B ne peut reconstruire M sans arrêter l'exécution du protocole.

Ainsi, même en manipulant le temps, B ne peut rien apprendre sur l'identité des composantes. Ainsi, il ne peut faire mieux que tenter de deviner l'étape à laquelle p_r a été envoyée. Donc la probabilité que B triche ϕ en manipulant le temps est encore fonction du nombre z_r de fausses composantes transmises par A.

Nous remarquons que le protocole peut rapidement devenir inutilisable si A envoie des fausses composantes pour plus d'une vraie composante. En effet, dans un tel cas, B devra dépenser un temps énorme à simplement différencier les vraies des fausses composantes. Le résultat ci haut indique qu'envoyer des fausses composantes pour des vraies composantes autres que la dernière ne diminuera pas la probabilité que B triche. Ainsi une approche plus valable, serait d'envoyer des fausses composantes seulement pour la dernière vraie composante.

Dans ce cas, si z_r est le nombre de fausses composantes de la dernière vraie composante et que p_r a été choisie d'une distribution uniforme par exemple, la probabilité ϕ que B triche serait simplement $1/z_r$. Si nous fixons $\varepsilon \geq \phi$, nous concluons que $\psi + \phi \leq \varepsilon$ et notre résultat tient.

□

Nous avons montré que les cinq conditions 4.3.1, 4.3.2, 4.3.3, 4.3.4 et 4.3.5 sont suffisantes pour garantir qu'un protocole $P^n_{A,B}(M)$ soit ε-équitable. Nous avons ainsi défini une classe de protocoles caractérisés par le fait que l'agent A ne peut tricher et que B peut tricher d'une seule et unique façon : en n'envoyant pas la preuve de réception pour la dernière composante. La probabilité ϕ peut être borné par $\phi \leq \varepsilon$ avec $0 < \varepsilon \leq 1$.

Pour une implémentation spécifique, les concepteurs doivent déterminer le meilleur mécanisme pour fixer la borne ε aussi petite que possible. Il y a plusieurs mécanismes possibles. Imaginons par exemple un protocole où A cache le message M dans un image et que le temps pour déterminer qu'un image quelconque contient M dépasse le temps entre la transmission de l'image et la réception de la preuve de réception, alors A pourrait envoyer par exemple s faux images avant d'envoyer la
bonne.

Dans la prochaine section, nous allons présenter l'exemple concret du protocole M-R, et montrer qu'il satisfait les cinq conditions.

4.4 Exemple concret : le protocole M-R

Nous allons démontrer que le protocole de non-répudiation sans TTP M-R conçu par Markowitch et Roggerman, [21], et défini auparavant dans la section 2.2.2, satisfait aux cinq conditions 4.3.1, 4.3.2, 4.3.3, 4.3.4 et 4.3.5. Nous en tirerons la conclusion que le protocole appartient à la classe de protocole défini par nos conditions et caractérisée par l'envoi de fausses clés.

Nous rappelons que le protocole M-R est caractérisé par le fait que l'initiateur transmet l'encryption du message M, $C = \{M\}_k$, suivit de $s-1$ ($1 \leq s$) fausses clés et finalement de la vraie clé k. Le receveur B doit répondre à chaque message de A avec une preuve de réception dans un délai inférieur au temps d'attente maximale t. Nous reproduisons le protocole M-R ci-dessous pour faciliter la compréhension de notre analyse.

$$
\begin{array}{rlll}
1: & A \rightarrow B: & f_{EOO_{(C)}}, B, l, C, \{f_{EOO_{(C)}}, B, l, C\}_{K_A^-} \\
2: & B \rightarrow A: & f_{EOR_{(C)}}, A, l, \{f_{EOR_{(C)}}, A, l, C\}_{K_B^-} \\
3: & A \rightarrow B: & f_{EOO_{k,1}}, B, l, 1, v_1, \{f_{EOO_{k,1}}, B, l, 1, v_1\}_{K_A^-} \\
4: & B \rightarrow A: & f_{EOR_{k,1}}, A, l, 1, , \{f_{EOR_{k,1}}, A, l, 1, v_1\}_{K_B^-} \\
 & \vdots & \\
2s+1: & A \rightarrow B: & f_{EOO_{k,s}}, B, l, s, v_s, \{f_{EOO_{k,s}}, B, l, s, v_s\}_{K_A^-} \\
2s+2: & B \rightarrow A: & f_{EOR_{k,s}}, A, l, s, \{f_{EOR_{k,s}}, A, l, s, v_s\}_{K_B^-}
\end{array}
$$

Nous voulons montrer que ce protocole satisfait aux conditions 4.3.1, 4.3.2, 4.3.3 et 4.3.4. Malheureusement, le protocole tel que présenté n'est pas sous une forme qui nous permettra de faire la vérification directement. En effet, nous remarquons que les preuves sont sous forme de signatures digitales, tandis que les conditions de corrections sont formulées en terme des preuves partielles de réception et d'origine.

Avant de montrer notre résultat principal, il nous faudra démontrer que la connaissance d'une signature digitale d'un message m et un ensemble d'identificateur, est équivalente à la connaissance d'une preuve partielle de non-répudiation d'origine $EOO(m)$ ou de réception $EOR(m)$.

Par la définition des preuves partielles (2.2.9), les valeurs $EOR(m)$ et $EOO(m)$ permettent à un juge de décider que la composante m a été envoyée durant une session T précise. Supposons alors que B reçoit le message m et la signature digitale $\{m\}_{SK_A}$. On suppose que le juge possède la clé publique PK_A, ainsi il pourra conclure que A a généré m et donc envoyé m. Mais il ne pourra pas savoir automatiquement à quelle composante associer m ni à quelle session associer la preuve.

Considérons par exemple le cas où deux messages sont reçus en désordre ou le cas où deux sessions s'entrelacent. Dans ces cas, il devient difficile d'associer la signature digitale à la bonne session et à la bonne composante. Imaginons par exemple, le cas d'une session du protocole M-R, où la preuve d'origine de la clé k, $EOO(k)$, est reçu avant la preuve de réception de la fausse clé v_i. Dans ce cas, B pourrait bien envoyer au juge la signature digitale de la fausse clé v_i comme preuve d'origine de la vraie clé k.

Pour prévenir de telles situations, les auteurs du protocole M-R ont ajouté à chaque signature digitale les identificateurs suivants : une identification du receveur (A ou B), une identification unique de la session l, une fonction étiquette $f_{(m)}$ qui identifie l'usage de la preuve et une identification de l'index de la clé v_i ($1 \leq i \leq s$). La proposition suivante démontre que ces éléments sont suffisants pour permettre à un agent de relier une preuve à une session et une composante précise et donc de prouver que la connaissance de signature digitale est équivalente à la connaissance de la preuve partielle.

Proposition 4.4.1 *Pour une session* T *du protocole M-R, supposons que* $PK_B \in \mathcal{K}_A(\mathsf{T})$ *et* $PK_A \in \mathcal{K}_B(\mathsf{T})$. *Si*

- l *est un identificateur unique de la session* T
- $f_{EOO(C)}$ *et* $f_{EOR(C)}$ *identifient respectivement une preuve d'origine ou de réception pour le message* C.
- $f_{EOO(v_i,i)}$ *et* $f_{EOR(v_i,i)}$ *identifient respectivement une preuve d'origine ou de réception pour le message clé* v_i.
- x *identifie l'index de la clé* v_x

alors on a toujours que

$$\begin{cases} \{f_{EOO(C)}, B, l, C\} \in \mathcal{K}_B(\mathsf{T}) \wedge \{f_{EOO(C)}, B, l, C\}_{SK_A} \in \mathcal{K}_B(\mathsf{T}) \\ \Rightarrow EOO(C) \in \mathcal{K}_B(\mathsf{T}) \\ \{f_{EOR(C)}, A, l\} \in \mathcal{K}_A(\mathsf{T}) \wedge \{f_{EOR(C)}, A, l, C\}_{SK_B} \in \mathcal{K}_A(\mathsf{T}) \Rightarrow EOR(C) \in \mathcal{K}_A(\mathsf{T}) \\ \{f_{EOO(v_x)}, B, l, x, v_x\} \in \mathcal{K}_B(\mathsf{T}) \wedge \{f_{EOO(v_x)}, B, l, x, v_x\}_{SK_A} \in \mathcal{K}_B(\mathsf{T}) \\ \Rightarrow EOO(v_x) \in \mathcal{K}_B(\mathsf{T}) \\ \{f_{EOR(v_x)}, A, l, x\} \in \mathcal{K}_A(\mathsf{T}) \wedge \{f_{EOR(v_x)}, A, l, x, v_x\}_{SK_B} \in \mathcal{K}_A(\mathsf{T}) \\ \Rightarrow EOR(v_x) \in \mathcal{K}_A(\mathsf{T}) \end{cases}$$

Preuve:

Selon la définition des preuves partielles 2.12, une preuve $EOR(m)$ générée par B ou une preuve $EOO(m)$ générée par A est associée à une composante m et une session T précise. Nous devons démontrer que les identificateurs sont suffisants pour associer uniquement une signature digitale à sa composante et sa session.

Examinons tout d'abord le premier cas qui concerne l'agent B. Supposons alors que pour une session T on a que

$$\{f_{EOO(C)}, B, l, C\} \in \mathcal{K}_B(\mathsf{T}) \wedge \{f_{EOO}, B, l, C\}_{SK_A} \in \mathcal{K}_B(\mathsf{T})$$

Puisque B possède la clé publique de A, il peut vérifier que A a vraiment généré le message $\{f_{EOO(C)}, B, l, C\}$.

Possédant les valeurs B et l générées par B, l'agent B peut vérifier que la signature digitale lui a été destinée et est associée uniquement à la session T. L'étiquette $f_{EOO(m)}$ quant à elle, lui permet de conclure que la signature digitale est une preuve d'origine pour le message C.

On peut procéder à une preuve similaire pour les autres cas.

□

Cette proposition nous permet de reformuler le protocole M-R en terme des preuves partielles EOR et EOO. Ceci simplifie énormément la notation et le protocole devient :

$$\begin{array}{rlll} 1: & A \rightarrow B: & C, EOO(C) \\ 2: & B \rightarrow A: & EOR(C) \\ 3: & A \rightarrow B: & v_1, EOO(v_1) \\ 4: & B \rightarrow A: & EOR(v_1) \\ \vdots & & & \\ 2s-1: & A \rightarrow B: & v_{s-1}, EOO(v_{s-1}) \\ 2s: & B \rightarrow A: & EOR(v_{s-1}) \\ 2s+1: & A \rightarrow B: & v_s, EOO(v_s) \\ 2s+2: & B \rightarrow A: & EOR(v_s) \end{array} \tag{4.15}$$

Procédons maintenant à la présentation de notre résultat concernant l'équité du protocole M-R.

Proposition 4.4.2 *Le protocole* M-R *est ε-équitable.*

Preuve:

Nous devons démontrer que le protocole comme énoncé dans l'équation 4.15 satisfait les conditions 4.3.1, 4.3.2, 4.3.3 , 4.3.4 et 4.3.5.

Tout d'abord, puisque $\mathcal{C} = \{C, v_1, \ldots, v_{s-1}, k\}$, nous devons vérifier que les conditions tiennent pour exactement $s+1$ composantes. Nous observons que

$$C \in m_1 \downarrow \quad \text{et} \quad EOO(C) \in m_1 \downarrow \tag{4.16}$$

et que pour chaque $i \in [1, s]$, on a que

$$v_i \in m_{2i+1} \downarrow \quad \text{et} \quad EOO(v_i) \in m_{2i+1} \downarrow . \tag{4.17}$$

ce qui satisfait les conditions 4.3.1 et 4.3.2. Il en suit que les propriétés 4.3.3 et 4.3.4 sont aussi satisfaites puisque

$$EOR(C) \in m_2 \downarrow \text{ et } EOR(v_i) \in m_{2i+2} \downarrow \forall i \in [1, s]. \tag{4.18}$$

Il nous reste à démontrer que le protocole respecte la propriété de timeliness. Tout d'abord les auteurs observent que chaque clé (vraie ou fausse) est de la même longueur. Ils proposent ensuite un algorithme de décryptage qui limite la vitesse de décryptage.

Considérons l'algorithme de décryptage qui possède les caractéristiques suivantes :

- $C = \{M\}_k$ doit être déchiffré au complet avant d'obtenir une portion quelconque du message M
- Le temps de décryptage complet de C doit toujours dépasser le temps t.

Ces conditions sont suffisantes pour empêcher B d'identifier la vraie clé dans un délai inférieur à t. Donc le protocole M-R satisfait la condition 4.3.5. Par contre B peut toujours tricher en devinant la valeur s. Si ϕ représente la distribution utilisée par A pour choisir s, B pourra deviner s avec probabilité $\leq \phi$. Nous concluons que $0 < \phi \leq \varepsilon$.

□

4.5 L'ε-équité est une stratégie optimale

Les protocoles de non-répudiation sans TTP sont des situations de compétition entre deux agents et il est facile de les imaginer comme étant des jeux, où les agents sont des joueurs, et dont le but est de choisir une séquence d'actions ou une stratégie qui leurs permettra de tricher.

Selon Berger [7], le but de la théorie des jeux est de déterminer les stratégies optimales dans une situation compétitives impliquant au moins deux antagonistes intelligents appelés joueurs. Pour un protocole $P^n_{A,B}(M)$ qui satisfait les conditions 4.3.1, 4.3.2, 4.3.3, 4.3.4 et 4.3.5, nous voulons utiliser cette même théorie pour déterminer s'il existe des stratégies disponibles à A ou à B, qui leur rapporteront un gain quelconque avec une probabilité supérieure à ε. Pour ce faire, nous devons modéliser les protocoles de non-répudiation sans TTP $P^{n,t}_{A,B}(M)$ comme jeux.

Nous définissons un jeu entre les joueurs A et B, dit jeu à 2-joueurs, comme une séquence de coups, où les joueurs prennent des tours l'un après l'autre pour jouer leur coup. Chaque jeu a un résultat précis et un joueur peut l'influencer par sa séquence de coups, ou stratégie. Nous dénotons par $\mathcal{A}$ l'ensemble des stratégies disponibles à A et par $\mathcal{B}$, l'ensemble des stratégies disponibles à B.

Le résultat d'un jeu est habituellement défini en terme de gain ou perte monétaire. Ainsi, à chaque paire de stratégies (S_A, S_B), $S_A \in \mathcal{A}$ et $S_B \in \mathcal{B}$, on associe un résultat égal à la valeur de la fonction $L(S_A, S_B)$ appelé la fonction *payout* . Si $L(S_A, S_B)$ représente un gain pour A et une perte de valeur égale pour B, alors le jeux est dit *zero-sum.* Pour de tels jeux, une perte pour l'un est un gain pour l'autre, donc chaque joueur tente de

maximiser son gain en maximisant la perte de son adversaire. La coopération n'a aucun sens dans les jeux *zero-sum*, puisque chaque joueur paie pour le gain de l'autre.

Dans [7], Berger montre plusieurs méthodologies possibles pour déterminer une stratégie optimale[1] dans les jeux à 2-joueurs. Ils démontrent aussi que la détermination d'une telle stratégie dépend de l'existence de stratégies maximin et minimax pour A et B respectivement. Ci-dessous, nous présentons les définitions des stratégies maximin et minimax tirées de [7].

Définition 4.5.1 (Stratégie maximin) *Une stratégie maximin pour le joueur A est une stratégie $S_A^* \in \mathcal{A}$, qui maximise $inf_{S_B \in \mathcal{B}} L(S_A, S_B)$, c.-à-d. une stratégie pour laquelle :*

$$inf_{S_B \in \mathcal{B}} L(S_A^*, S_B) = sup_{S_A \in \mathsf{A}} inf_{S_B \in \mathcal{B}} L(S_A, S_B) \tag{4.19}$$

Nous dénotons par $\underline{V}$ la valeur ou le résultat de la stratégie maximin.

De façon similaire, Berger définit une stratégie minimax pour le joueur B comme suit :

Définition 4.5.2 (Stratégie minimax) *Une stratégie minimax pour le joueur B est une stratégie $S_B^* \in \mathsf{B}$, qui minimise $sup_{S_A \in \mathsf{A}} L(S_A, S_B$ c.-à-d. une stratégie pour laquelle :*

$$sup_{S_A \in \mathsf{A}} L(S_A, S_B^*) = inf_{S_B \in \mathsf{B}} sup_{S_A \in \mathsf{A}} L(S_A, S_B)$$

(4.20)

Nous dénotons par $\overline{V}$ la valeur ou le résultat d'une stratégie minimax.

De façon intuitive, il est facile de comprendre pourquoi les stratégies maximin et minimax sont des stratégies optimales. La stratégie minimax est la stratégie qui minimise le gain maximal du joueur A, tandis que la stratégie maximin est celle qui maximise la perte minimum du même joueur A. Donc pour un jeux zero-sum, si A joue sa stratégie maximin et B sa stratégie minimax, nous devons avoir que $\overline{V} = \underline{V}$

Un jeu à 2-joueurs zero-sum pour lequel $\overline{V} = \underline{V}$ est dit avoir valeur V. Si des stratégies maximin et minimax peuvent être déterminées pour un tel jeu, alors il est un jeu zero-sum

[1]Berger utilise le terme Bayesien à la place d'optimale, puisque c'est une stratégie qui minimise le *posterior expected loss.*

strictement déterminé. Cette dernière notion est très importante pour la définition des stratégies optimales.

Considérons un jeu à 2-joueurs strictement déterminé dans lequel les joueurs choisissent d'utiliser leurs stratégies maximin et minimax. Si B utilise sa stratégie minimax, sa perte ne pourrait jamais être plus grande que V, tandis que si A se sert de sa stratégie maximin, la perte de B ne pourra être moins que V. Donc, si A a une stratégie maximin et le jeu a une valeur V, la stratégie minimax de B sera une stratégie optimale par rapport à la stratégie de A et vice-versa.

Ce résultat est énoncé plus formellement dans le théorème suivant, dû à Berger. Nous l'énonçons en terme du maximin au lieu du minimax.

Théorème 4.5.3 *Supposons que S_B^* est une stratégie minimax pour le joueur B et que le jeu a une valeur. Alors toute stratégie maximin est un stratégie optimale par rapport à S_B^* .*

Preuve:
Voir [7]. □

L'objectif de l'analyse des jeux à 2-joueurs zero-sum consiste a déterminer s'il possède une valeur V et des stratégies minimax et maximin. Si c'est le cas, on dit que le jeu peut être résous et des stratégies optimales pour A et B trouvées. Malheureusement, pour la plupart des jeux, cette approche directe n'est pas possible. Une autre approche suggérée par Berger, consiste à deviner des stratégies optimales et utiliser le théorème qui suit (tiré de [7]) pour vérifier qu'elles sont réellement maximin et minimax et que le jeu a une valeur.

Théorème 4.5.4 *Supposons que S_A^* et S_B^* sont des stratégies de A et B, et supposons que pour toutes $S_A \in \mathsf{A}$ et toutes $S_B \in \mathsf{B}$,*

$$L(S_A, S_B^*) \leq L(S_A^*, S_B)$$

alors le jeu a valeur $V = L(S_A^*, S_B^*)$ et est strictement déterminé avec stratégie maximin S_A^* et stratégie minimax S_B^*.

Preuve:
Voir [7]. □

4.5.1 La non-répudiation comme jeux à 2-joueurs zero-sum

Notre but est de modéliser le protocole $P^{n,t}_{A,B}(M)$ comme un jeu à 2-joueurs zero-sum et utiliser les résultats de Berger cités ci-haut pour trouver les stratégies optimales. Dans notre modèle, le joueur A représente l'initiateur, tandis que B représente le receveur. On dira qu'un joueur a gagné une partie s'il peut tricher (nous utilisons la définition de tricher définie dans la definition 2.2.4). Dans ce cas, l'autre joueur doit lui payer 1, autrement, il n'y aucun gain ou perte. Ceci est clairement un jeu à 2-joueurs zero-sum et représente bien les objectifs des agents malhonnêtes prenant part à un protocole de non-répudiation.

Nous définissons maintenant ce que l'on entend par stratégie dans un tel jeu. Une stratégie est définie comme étant une séquence d'étapes, soit de longueur paire (pour le joueur A), soit de longueur impaire (pour le joueur B).

Définition 4.5.5 (Stratégie) *Supposons que $P^{n}_{A,B}(M) = a_1 \dots a_n$ est un protocole de non-répudiation sans TTP. L'ensemble des stratégies du joueur A, dénoté $\mathcal{A}$, est un ensemble de sous séquences de $P^{n,t}_{A,B}(M)$ de longueurs pair, c.- à -d. :*

$$\mathcal{A} = \{t \sqsubseteq^{evn} P^{n,t}_{A,B}(M)\}$$

L'ensemble des stratégies pour le joueur B, dénoté $\mathcal{B}$, est l'ensemble des sous séquences de $P^{n,t}_{A,B}(M)$ de longueurs impair, c.-a-d. :

$$\mathcal{B} = \{t \sqsubseteq^{odd} P^{n,t}_{A,B}(M)\}$$

Considérons les ensembles de stratégies $\mathcal{A}$ et $\mathcal{B}$ pour les joueurs A et B respectivement. Si nous associons la probabilité $\pi(S_A, S_B)$ que A triche et la probabilité $\delta(S_A, S_B)$ que B triche, ou $S_A \in \mathcal{A}$ et $S_B \in \mathcal{B}$, alors la fonction payout est :

$$L(S_A, S_B) = \pi(S_A, S_B) - \delta(S_A, S_B) \tag{4.21}$$

Par la prochaine proposition, nous allons montrer que le jeu défini pour le protocole $P^{n,t}_{A,B}(M)$ a une valeur V et des stratégie maximin et minimax. En plus, la stratégie maximin de A satisfait les conditions 4.3.1, 4.3.2 et 4.3.4 tandis que la stratégie minimax pour B satisfait les conditions 4.3.3 et 4.3.5.

Proposition 4.5.6 (Stratégie optimale pour le jeu de non-répudiation) *Les stratégies qui satisfont la condition 4.3.3 et 4.3.5 sont minimax pour B, et il existe au moins une stratégie qui satisfait les conditions 4.3.1, 4.3.2 et 4.3.4 qui est maximin pour A. Cette stratégie est optimale par rapport à la stratégie minimax du joueur B.*

Preuve:
Supposons que $\mathcal{B}' \subseteq \mathcal{B}$ représente l'ensemble des stratégies du joueur B qui satisfont les conditions 4.3.3 et 4.3.5 Supposons aussi que $\mathcal{A}' \subseteq \mathcal{A}$ représente l'ensemble des stratégies du joueur A qui satisfont les conditions 4.3.1, 4.3.2 et 4.3.4. Par la proposition 4.3.6, si $\S_A \in \mathcal{A}'$ et $\S_B \in \mathcal{B}'$, alors $\pi(S_A, S_B) = 0$. Ceci implique que pour toute stratégie minimax $S_B^* \in \mathcal{B}'$ la fonction payout est tout simplement :

$$L(S_A, S_B^*) = -\delta(S_A, S_B^*).$$

La proposition 4.3.6 impose que la somme des probabilités que A et B trichent est toujours inférieure ou égale à ε. Les probabilités individuelles vont satisfaire la même relation d'ordre. En particulier, pour $S_A \in \mathcal{A}'$, on aura que $\delta(S_A, S_B) \leq \varepsilon$. Mais le joueur A peut choisir, d'un nombre infini de stratégie, la stratégie pour laquelle ε est aussi petite que possible. Par exemple, dans le protocole M-R, si le nombre de fausses clés $s - 1$ suit une distribution uniforme U(1,N) alors

$$L(S_A, S_B) = \pi(S_A, S_B) - 1/(N-1) \tag{4.22}$$

et pour un $S_A^* \in \mathcal{A}'$

$$L(S_A^*, S_B) = \pi(S_A, S_B) - lim_{N\to\infty} 1/(N-1). \tag{4.23}$$

Il va sans dire qu'un tel protocole serait moins que pratique étant donné le grand nombre de messages qui pourraient être envoyé.

Ainsi, il existe au moins un $S_A^* \in \mathcal{A}'$ tel que $\delta(S_A^*, S_B) = 0$, et puisque cette stratégie n'affecte pas la probabilité que A triche ou non, la fonction payout est :

$$L(S_A^*, S_B) = \pi(S_A^*, S_B).$$

Mais π et δ sont des probabilités, avec valeurs entre 0 et 1. Si nous comparons les expressions pour les stratégies minimax et maximin (S_B^* et S_A^* respectivement), nous observons que pour toute $S_A \in \mathcal{A}'$ et $S_B \in \mathcal{B}'$

$$L(S_A, S_B^*) \leq L(S_A^*, S_B)$$

Ainsi par le théorème 4.5.4 , la valeur du jeu est :

$$V = L(S_A^*, S_B^*) = 0$$

Puisque le jeu est strictement déterminé avec S_A^* et S_B^* des stratégies maximin et minimax, nous déduisons par le théorème 4.5.3, que S_A^* est optimale par rapport a la stratégie minimax S_B^* .

□

Nous avons démontré que si les agents du jeu de non-répudiation sont intelligents et utilisent leurs stratégies minimax et maximin, alors : quand B utilise sa stratégie minimax satisfaisant la condition 4.3.3, sa perte ne sera jamais plus que $\overline{V} = 0$. De plus, quand A utilise sa stratégie maximin satisfaisant les conditions 4.3.1, 4.3.2 et 4.3.4, la perte de B ne pourra jamais être moins que $\underline{V} = 0$.

Donc les conditions qui garantissent l'équité ε sont optimales pour le modèle de jeux défini pour le protocole de non-répudiation $P^n_{A,B}(M)$. Ce résultat implique que la classe de protocoles définie par les conditions 4.3.1, 4.3.2, 4.3.4, 4.3.3 et 4.3.5 ne peut faire mieux qu'une équité probabiliste.

4.6 Conclusion

Dans ce chapitre nous avons développé cinq conditions syntaxiques et démontré qu'elles sont suffisantes pour garantir une équité probabiliste et pour satisfaire la propriété de timeliness. Comme elles sont syntaxiques, un développeur pourrait facilement s'en servir pour concevoir une application de commerce électronique par exemple, qui doit offrir des services de non-répudiation. Il est intéressant à noter, que les conditions seront très efficaces pour la situation où le développeur veut offrir plusieurs services incluant la non-répudiation dans la même application.

Nous avons aussi démontré qu'un protocole qui satisfait ces conditions ne pourra satisfaire plus d'une équité probabiliste, c'est-à-dire qu'il y aura toujours une probabilité, aussi minime qu'elle soit, que le receveur puisse tricher. Ce résultat n'est que confirmation du résultat de Even et Yacobi [12], mais souligne la façon naturelle dont une théorie des jeux peut être utilisée pour analyser les propriétés de protocole de non-répudiation.

Chapitre 5

Conclusion

Le but de ce mémoire fut de concevoir des conditions suffisantes pour garantir les propriétés de timeliness et d'équité pour les protocoles de non-répudiation. Nous avons visé à concevoir des conditions syntaxiques qui permettraient l'implémentation facile des services de non-répudiation dans une application quelconque.

Dans ce travail, nous avons vu que les protocoles de non-répudiation peuvent varier énormément selon le niveau de participation du TTP. La présence d'un TTP peut garantir l'équité, tandis que sans TTP, on ne peut offrir mieux qu'une équité probabiliste. Vu cette différence importante, nous avons choisi d'élaborer des conditions pour un protocole de non-répudiation avec TTP et un sans TTP. Parmi les protocoles de non-répudiation avec TTP, nous avons choisi d'étudier les protocoles avec TTP online, tandis que parmi les protocoles sans TTP, nous avons choisi la famille de protocoles qui utilise un mécanisme de faux messages.

Les conditions élaborées donnent un ordre fixe aux messages, et démontrent que des protocoles existants sont corrects. Sauf pour les conditions de timeliness, elles respectent la forme syntaxique. Il est plus difficile de formuler des conditions de temps complètement avec la syntaxe des message

s. La force de ces conditions est surtout dans le fait qu'elles définissent une recette pour l'élaboration de protocoles corrects. Ainsi en suivant à la lettre les conditions, un développeur pourrait élaborer des applications qui garantiront des services de non-répudiation.

Les conditions sont cependant restreintes à des types précis de protocoles. Donc, il y aurait un besoin d'élaborer des conditions pour plusieurs autres types, pour donner autant de choix que possible aux développeurs. Par exemple, on verrait une grande nécessité d'élaborer des conditions pour les protocoles avec TTP offline qui sont reconnus pour leurs complexité.

Bibliographie

[1] K. Adi, M. Debbabi et M. Mejri. *A new logic for electronic commerce protocols.* Theoretical Computer Science 291 (2003), pages 223-283. Elsevier.

[2] R. Alur, T. Henzinger et O. Kupferman. *Alternating-time temporal logic.* Proceedings of the 38th Annual Symposium on Foundations of Computer Science, pages 100-109. IEEE Computer Society Press, 1997.

[3] R. Alur, T. Henzinger, F.Mang, S. Qadeer, S. Rajamani et S. Tasiran. *MOCHA : modularity in model checking.* CAV 98 : Computer-aided Verification, Lecture Notes in Computer Science 1427, pages 521-525. Springer-Verlag, 1998.

[4] D. Beauquier. *On Probabilistic Timed Automata.* Theoretical Computer Science 292, 65-84, 2003.

[5] M. Ben-Or, 0. Goldreich, S. Micali et R.L. Rivest. *A fair protocol for signing contracts.* IEEE Trans. Inform. Theory 36 (I) (1990) 40-46.

[6] C. Baier et M. Kwiatowska. *Model Checking for a Probabilistic Branching Time Logic with Fairness.* Distributed Computing 11(3), pages 125-155, 1998.

[7] J. Berger, *Statistical Decision Theory and Bayesian Analysis.* Springer Series in Statistics, Springer-Verlag, 1985.

[8] L. Botao et L. Junzhou. *On Timeliness of a Fair Non-repudiation Protocol.* InfoSecu'04, Nov 14-16, 2004, Shanghai, China.

[9] M. Burrows, M. Abadi et R. Needham. *A logic for authentication.* Technical Rpeort 39, Digital Systems Research Center, février 1989 (revisé en février 1990).

[10] E.M. Clarke et E. A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. Dans Logic of Programs Workshop, number 131 in LNCS. Springer Verlag, 1981.

[11] T. Coffey et P. Saidha. *Non-repudiation with mandatory proof of receipt.* ACMCCR : Computer Communication Review 26.

[12] S. Even and Y. Yacobi. *Relations among public Key Signature Systems.* Technical Report #175, Comp. Sci. Dept., Technion, Haifa, Israel, March, 1980.

[13] H. Houmani et M. Mejri . *Secure Protocols for Secrecy.* LICS'03 Satellite Workshop on Foundations of Computer Security - FCS'03. Elsevier, 2003.

[14] M. Huth et M. Ryan. *Logic in Computer Science : Modelling and reasoning about systems.* Cambridge University Press,2000.

[15] S. Kremer et O. Markowitch, *A multi-party optimistic non-repudiation protocol.* In Proceedings of teh 3rd International Conference on Information Security and Cryptology (ICISC 2000), Volume 2015 of Lecture Notes in Computer Science, Seoul, South Korea, 2000. Springer-Verlag.

[16] K. Kim, S. Park et J. Baek. *Improving Fairness and Privacy of Zhou-Gollman's Fair Non-Repudiation Protocol.* In 2000 IEEE International Conference on Communication, Volume :3, 2000, pp. 743-1747.

[17] S. Kremer et J-F. Raskin. *A game-based verification of non-repudiation and fair exchange protocols.* In CONCUR 2001, volume 2154 of Lecture Notes in Computer Science, pages 551-565, Springer-Verlag, 2001.

[18] S. Kremer, O. Markowitch et J. Zhou. *An intensive survey of non-repudiation protocols.* Computer Communications, 25(17) :1606-1621, 2002.

[19] R. Lanotte, A. Maggiolo-Schettini et A. Troina. *Automatic Analysis of a Non-Repudiation Protocol.* Electronic Notes in Theoretical Computer Science.

[20] G. Lowe. An attack on the Needham-Schroeder public-key protocol using FDR. Lecture Notes in Computer Scinece, 1055 :147-157,1996.

[21] O. Markowitch et Y. Roggeman, *Probabilistic Non-Repudiation without Trusted Third Party.* In Second Conference on Security in Communication Networks, Amalfi, Italy, 1999.

[22] C. Meadows. *Formal Verification of Cryptographic Protocols : A survey.*

[23] G. Norman and V. Shmatikov. *Analysis of a Probabilistic Contract Signing.* In Formal Aspects of Security (FASec'02) , volume 2629 of LNCS, pages 81-96, 2002.

[24] PRISM URL : http : //www.cs.bham.ac.uk/ dxp/prism.

[25] A. Pnueli. *The temporal logic of programs.* Proc. 18th IEEE Symposium on Foundations of Computer Science. Providence, RI. IEEE Press. pp.46-57.

[26] M. Rabin. *Transaction Protection by Beacons.* Journal of Computer and System Sciences, 27 (2), 1983, pp. 256-267.

[27] V. Shmatikov et J. Mitchell. *Analysis of a fair exchange protocol.* In Symposium on Network and Distributed Systems Security (NDSS 2000), pages 119-128, San Diego, CA, 2000.

[28] P. Syverson, *Weakly secret bit commitment : Applications to lotteries and fair exchange.* In Proceedings of the 1998 IEEE Computer Security Foundations Workshop (CSFW11), 1998.

[29] P. Syverson et I. Cervesato. *The Logic of Authentification Protocols.* FOSAD'00, Bertinoro, Italy, Septembre 2000.

[30] P. Syverson et L. Gong. *Fail-Stop Protocols : An Approach to Designing Secure Protocols.* First International Working Conference on Dependable Computing for Critical Applications, Sept. 1995.

[31] J. D. Tygar. *Atomicity in electronic commerce.* In Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing (PODC '96), pages 8–26, Philadelphia, PA, May 1996. ACM Press.

[32] Y. Yin, X. Wang et H. Yu. *Finding collisions in the full Sha-1.* Crypto '05. Santa Barbara, CA, 2005.

[33] J. Zhou, C. You et K. Lam. *On the efficient Implementation of Fair Non-repudiation.*Computer Communication Review 28 (5) (1998) 50-60.

[34] N. Zhang et Q. Shi. *Achieving non-rpudiation of receipt.* The Computer Journal, 39 (10) , 1996, pp.844-853.

[35] J. Zhou et D. Gollmann. *Towards verification of non-repudiation protocols.* In International Refinement Workshop and Formal Methods Pacific, pages 370-380, Canberra, Australia, 1998. Springer-Verlag.