UNIVERSITÉ DE RENNES 1

THÈSE DE DOCTORAT DE

L'UNIVERSITE DE RENNES 1

Ecole Doctorale N°601
Mathèmatique et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par

Alexandre DEBANT

Symbolic verification of distance-bounding protocols - Application to payment protocols

Thèse présentée et soutenue à IRISA, RENNES, le 17 novembre 2020
Unité de recherche : UMR 6074

Rapporteurs avant soutenance :

Bruno Blanchet, Directeur de recherche Inria, Paris, France
Sjouke Mauw, Professeur, University of Luxembourg, Luxembourg

Composition du jury :

Président :	David Pichardie,	Professeur, École normale supérieure de Rennes, France
Examinateurs :	Bruno Blanchet,	Directeur de recherche Inria, Paris, France
	Ioana Boureanu,	Lecturer, University of Surrey, Royaume-Uni
	Cas Cremers,	Professeur, CISPA, Allemagne
	Sjouke Mauw,	Professeur, University of Luxembourg, Luxembourg
Directrice :	Stéphanie Delaune,	Directrice de recherche CNRS, Rennes, France

Symbolic verification of distance-bounding protocols

Application to payment protocols

by

Alexandre Debant

Univ Rennes, CNRS, IRISA, Rennes, France

Supervisor

Stéphanie Delaune
Univ Rennes, CNRS, IRISA, Rennes, France

PhD defence committee

Reviewers:	Dr. Bruno BLANCHET	INRIA Paris, France
	Prof. Dr. Sjouke MAUW	Université du Luxembourg, Luxembourg
President:	Prof. Dr. David PICHARDIE	École normale supérieure de Rennes, France
Examiners:	Dr. Ioana BOUREANU	Université de Surrey, Royaume-Uni
	Prof. Dr. Cas CREMERS	CISPA, Allemagne
Supervisor:	Dr. Stéphanie DELAUNE	Univ Rennes, CNRS, IRISA, Rennes, France

Funding

This work has been partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 714955-POPSTAR).

Résumé

L'essor récent des nouvelles technologies, comme la Communication par Champ Proche (NFC), a permis l'apparition de nombreuses applications. Á titre d'exemples, nous pouvons mentionner les cartes d'accès de bâtiment, les clefs mains libres pour véhicules, ou encore les cartes de paiement sans contact. En 2019, les transactions sans contact représentaient près de la moitié des transactions effectuées en face-à-face [1]. En raison de la récente crise sanitaire du COVID-19, Mastercard a observé une adhésion accélérée à ce nouveau mode de paiement permettant de limiter les risques de contaminations [2] et cela devrait perdurer les années suivantes.

En raison du caractère personnel des données échangées lors des transactions, assurer la sécurité des paiements sans contact apparait d'une grande importance. Cela parait d'autant plus nécessaire que le caractère "sans contact" des communications offre une surface d'attaque d'autant plus grande comparée aux paiements avec contact, i.e. requérant l'insertion de la carte dans le terminal de paiement. En effet, au-delà de l'absence de code PIN à utiliser, un attaquant a la possibilité d'intercepter facilement les messages échangés. Un scénario d'attaque spécifique au caractère sans contact des transactions est alors l'attaque par relai (présenté en Figure 1) : un attaquant relaie, au travers d'un réseau de communication extrêmement rapide (e.g. le Wi-Fi), les messages échangés entre un terminal de paiement et une carte, située possiblement à plusieurs dizaines de mètres. Bien que la technologie NFC ne permette l'échange de messages que sur une courte distance, de tels scénarios ont été démontrés faisables en pratique dans le cadre des paiements sans contacts [Han05, FHMM10, SC13, CGdR+15]. Ils permettraient donc à des personnes malhonnêtes d'abuser de personnes honnêtes pour effectuer des paiements à leur place.

Afin de prévenir de telles attaques, des protocoles délimiteurs de distance ont été proposés ; ces derniers ayant pour objectif d'assurer la proximité physique entre le terminal de paiement et la carte au cours de la transaction. Dans ce manuscrit, nous proposerons différentes approches permettant d'établir des preuves formelles de sécurité pour ces protocoles. De plus, nous nous attacherons à rendre ces preuves automatiques par le biais de procédures et d'outils, afin de permettre l'analyse de nombreux protocoles.

[1] https://usa.visa.com/visa-everywhere/blog/bdp/2019/05/13/tap-to-pay-1557714409720.html

[2] https://mastercardcontentexchange.com/newsroom/press-releases/2020/april/mastercard-study-shows-consumers-globally-make-the-move-to-contactless-payments-for-everyday-purchases-seeking-touch-free-payment-experiences/

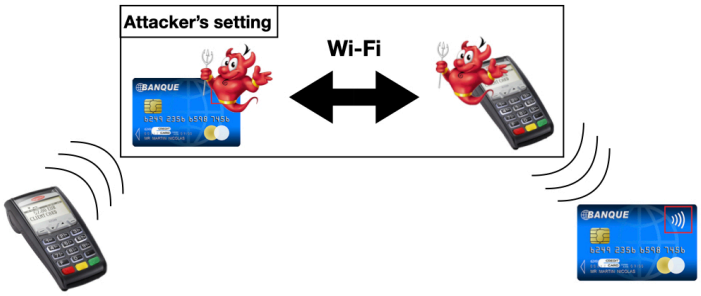


Figure 1: Attaque par relai contre un protocole de paiement sans contact.

Les protocoles délimiteurs de distance

Le premier protocole délimiteur de distance a été proposé par Brands and Chaum en 1993 [BC93]. Depuis, de nombreux protocoles ont été proposés (plus de 40 entre 2005 et aujourd'hui), mais ce n'est qu'en 2016 qu'un tel protocole est apparu dans la spécification officielle pour les paiements sans contact [EMV16].

Afin de pouvoir analyser ces protocoles dans la suite de ce manuscrit, il est important de noter qu'ils partagent tous une structure commune. En effet, comme présenté en Figure 2, nous pouvons distinguer trois phases : une première phase permet l'initialisation du protocole en échangeant des données propres à chaque session. Le cœur du protocole peut ensuite avoir lieu : le *vérifieur* (e.g. le terminal de paiement) génère des challenges qu'il envoie au *prouveur* (e.g. la carte) qui doit alors y répondre. Cet échange doit être effectué aussi vite que possible car le temps écoulé sera utilisé pour estimer la distance mutuelle des agents. En notant $t_{\mathtt{out}}$ la date d'émission du challenge, $t_{\mathtt{in}}$ la date de réception de la réponse, $t_{\mathtt{proc}}$ le temps de calcul requis par la carte, et c_0 la vitesse de communication des messages, nous pouvons dériver la formule suivante estimant la distance :

$$\mathsf{Dist}(Vérifieur, Prouveur) \leq \frac{\mathsf{c}_0}{2} \times (t_{\mathtt{in}} - t_{\mathtt{out}} - t_{\mathtt{proc}}).$$

Enfin, le protocole se termine par une phase de vérification durant laquelle le vérifieur peut estimer la distance du prouveur grâce à la formule précédente, ainsi qu'échanger des messages avec ce dernier pour, par exemple, assurer son authentification. Aucun temps n'est mesuré durant la première et la troisième phase.

Scénarios d'attaque

Afin d'analyser la sécurité des protocoles délimiteurs de distance, il est important de préciser la propriété de sécurité souhaitée. En effet, le scénario d'attaque par relai, bien que le plus simple à mettre en place, n'est pas le seul que doit prévenir ces protocoles. Par exemple, ils doivent également prévenir une personne malhonnête de s'authentifier auprès d'un vérifier, si ces deux deux sont distants.

La littérature distingue habituellement quatre scénarios d'attaques :

- Distance fraud : un prouveur malhonnête essaie de s'authentifier auprès d'un vérifieur honnête éloigné (voir Figure 3a).

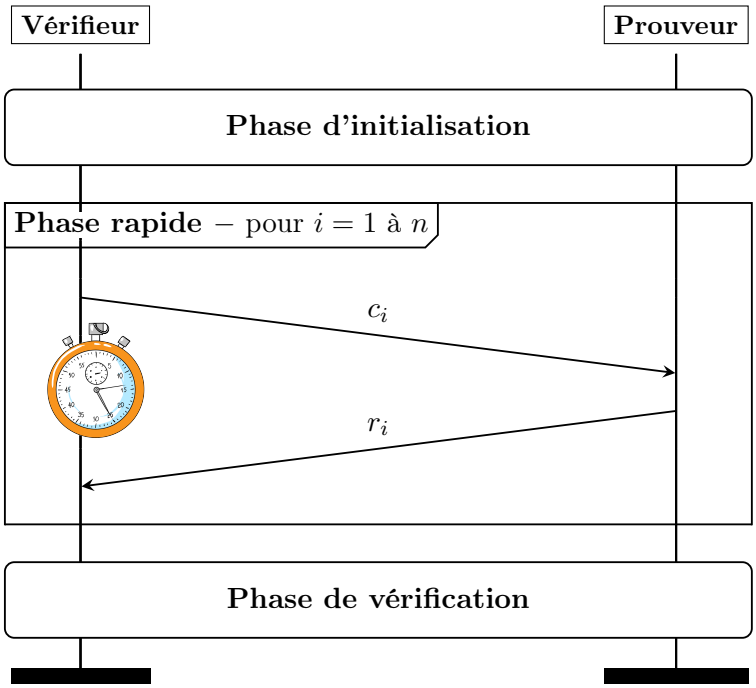


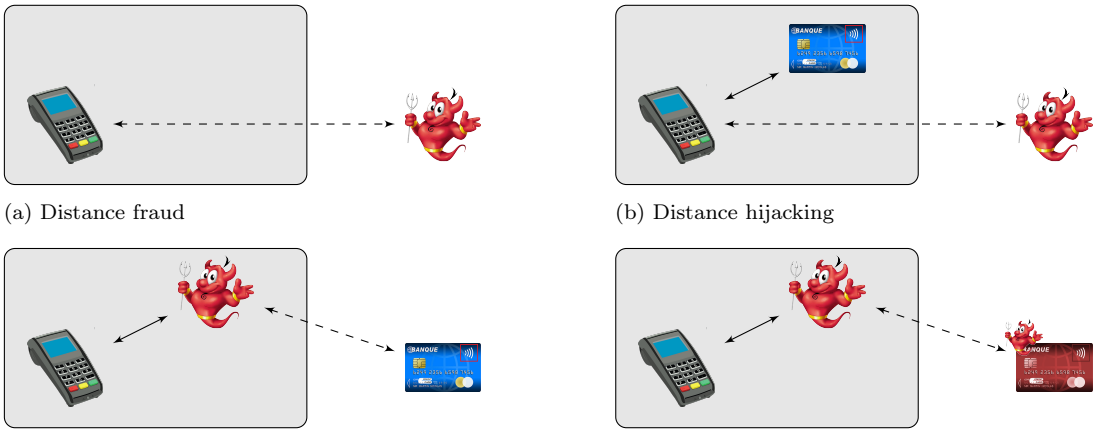
Figure 2: Structure usuelle d'un protocole délimiteur de distance.

- Distance hijacking : un prouveur malhonnête essaie de s'authentifier auprès d'un vérifieur honnête éloigné en abusant, possiblement, un agent honnête proche du vérifieur (voir Figure 3b).

- Mafia fraud (or MiM) : un attaquant situé entre un vérifieur et un prouveur, tous les deux honnêtes mais éloignés, essaie de faire authentifier le prouveur auprès du vérifieur (voir Figure 3c).

- Terrorist fraud : un prouveur malhonnête essaie de s'authentifier auprès d'un vérifieur honnête éloigné. Dans ce but, il accepte de conspirer avec d'autres attaquants, sans pour autant leur fournir une aide les permettant de mener à bien de futures attaques (voir Figure 3d).

À ce stade, il est important de noter que les scénarios sus cités ne fournissent que des exemples relativement simplistes d'attaques. Pourquoi considérer un unique attaquant dans une mafia fraud ? Pourquoi ne considérer qu'un unique agent honnête dans la proximité du vérifier dans le cas d'une distance hijacking ? Afin de prouver la sécurité des protocoles délimiteurs de distance, il est donc important de considérer ces scénarios comme des exemples permettant de dériver différentes classes d'attaque ; ces dernières contenant des scénarios mettant en jeu chacun un nombre arbitraire d'agents.



(a) Distance fraud

(b) Distance hijacking

(c) Mafia fraud

(d) Terrorist fraud

Figure 3: Les quatres scénarios d'attaques communément définis dans la littérature.

Analyses de sécurité

De nombreux travaux ont été proposés pour analyser la sécurité des protocoles délimiteurs de distance. Originellement, ces analyses avaient pour seul objectif de prouver la résistance du protocole contre des scénarios d'attaque très précis. Par exemple, en considérant un attaquant qui anticipe l'envoie des réponses durant la phase rapide en essayant de deviner la valeur des challenges. Ou encore, un attaquant commençant par jouer une phase rapide avec un agent arbitraire afin d'essayer d'en tirer profit pour mener à bien une attaque par la suite. Toutes ces analyses, bien que fournissant un premier gage de sécurité, ne procurent qu'une confiance limitée en la sécurité du protocole en raison de la spécificité des attaques qu'elles considèrent.

Plus récemment, des preuves de sécurité plus complètes ont été proposées dans des modèles calculatoires [DFKO11, ABK+11] ; ces preuves permettant d'obtenir une forte confiance en la sécurité des protocoles. En effet, ces modèles représentent très précisément les différentes entités mises en jeu au cours du protocole : les messages sont représentés par des chaines de bits, les différents rôles par des algorithmes, et les attaquants par des machines de Turing probabilistes s'exécutant en temps polynomial. Cette modélisation permet donc de considérer des scénarios d'attaque arbitraires (nombre d'agents, positions...), et d'obtenir des preuves de confiance. Une étude de cas comprenant près de vingt protocoles a récemment été menée suivant cette approche. [ABB+18].

Malheureusement, la principale limitation de cette approche est l'expertise requise pour analyser chaque protocole. En effet, chacune des analyses requiert une preuve rédigée "à la main" et pouvant receler de nombreuses subtilités. Cette limitation semblant inhérente à tout modèle calculatoire, nous développerons dans ce manuscrit différentes approches basées sur des modèles symboliques, et permettant une analyse automatique des différents protocoles délimiteurs de distance.

Vérification symbolique de protocoles

La vérification symbolique de protocoles est une technique usuelle pour analyser la sécurité des protocoles cryptographiques. En effet, sous l'hypothèse de quelques abstractions, cela permet l'utilisation de procédures automatiques de vérification.

Tout d'abord, la vérification symbolique de protocole attrait à la recherche d'*attaques logiques* ; ces attaques se basant uniquement sur la structure du protocole et des messages échangés. Les primitives cryptographiques, comme les chiffrements ou signatures, sont supposées parfaites, incassables. L'attaquant ne peut donc, par exemple, pas extraire d'information d'un chiffré dont il ne connait pas la clé de déchiffrement. Contrairement aux modèles calculatoires, les messages peuvent donc être abstraits par des termes, i.e. des symboles de fonctions appliqués à des termes atomiques. La deuxième abstraction des modèles symboliques porte sur l'attaquant. En effet, ces derniers modélisent un attaquant de type Dolev-Yao [DY83] qui est supposé omniscient et omniprésent. Il peut donc intercepter, créer, et envoyer des messages à tout moment et à tout agent impliqué dans le protocole.

Même si la vérification automatique de protocoles reste un problème difficile, indécidable pour des classes de protocoles intéressantes [MSDL99], des procédures et des outils, comme Proverif [Bla01] ou Tamarin [MSCB13], ont été développés pour tenter de résoudre ce problème. Bien que ces outils soient imparfaits (le problème est indécidable, ils peuvent donc ne pas terminer), ils ont prouvé leur efficacité pour analyser des protocoles de la vie réelle. Ils ont par exemple permis d'établir des preuves de sécurité pour le protocole TLS 1.3 Draft 18 [BBK17] ou le protocole de vote électronique Belenios [CGG19]. Ils ont également permis de détecter des vulnérabilités dans le protocole Single-Sign-On utilisé, par exemple, par Google Apps [ACC+08].

Application aux protocoles délimiteurs de distance

En essayant d'utiliser les outils existants pour analyser les protocoles délimiteurs de distance, nous pouvons immédiatement nous rendre compte que le modèle d'attaquant qu'ils implémentent, i.e. omniscient et omniprésent, est trop fort. En effet, un tel attaquant est capable de transmettre instantanément un message entre deux agents. Il est donc impossible de modéliser précisément le temps, donnée pourtant cruciale lors de la phase rapide d'un protocole délimiteur de distance.

Afin de dépasser cette limitation, quelques travaux ont été menés. Les deux premiers, proposés par Meadows *et al.* en 2007 [MPP+07] et par Basin *et al.* en 2011 [BCSS11], proposent deux modèles symboliques représentant de manière fidèle le temps ainsi que les positions des agents dans l'espace. Pour ce faire, ces deux modèles imposent une restriction physique simple : suffisamment de temps doit s'être écoulé entre l'envoi et la réception d'un message afin que ce dernier ait eu le temps d'atteindre sa destination. Cette restriction est appliquée aux messages émis par des agents honnêtes, mais également aux attaquants. Bien que ces modèles soient une première approche pour la vérification des protocoles délimiteur de distance, ils ne permettent pas une vérification automatique. Des preuves manuelles (ou assistées par l'outils de preuve Isabelle/HOL [NPW02]) restent nécessaires.

Une nouvelle approche a récemment été proposée par Chothia *et al.* [CGdR+15] et permet une analyse automatique des protocoles délimiteurs de distance. Cette dernière consiste à encoder les topologies, i.e. les positions relatives des agents dans l'espace, dans l'outils existant de vérification automatique Proverif. Cet encodage se servant astucieusement de la notion de phase que cet outil implémente. Malheureusement seules quelques topologies relativement simples (i.e. au plus deux positions différentes) peuvent être encodées. Les preuves de sécurité obtenues ont donc une portée limitée. Cette approche, bien que incomplète, sera la base de l'une des contribution présentée dans ce manuscrit.

Finalement, en parallèle des travaux présentés dans ce manuscrit, des résultats similaires ont été proposés par Mauw *et al.* [MSTPTR18, MSTPTR19]. Ces travaux étendent le modèle proposé par Basin *et al.* [BCSS11] de manière à permettre une vérification automatique des protocoles délimiteurs de distance. Tout au long de ce manuscrit nous porterons donc une attention particulière à la comparaison entre les résultats présentés et ceux introduits par ces derniers.

Contributions

Dans ce manuscrit, nous présenterons diverses approches permettant une analyse automatique des protocoles délimiteurs de distance. Pour ce faire, nous commencerons par développer un modèle symbolique permettant de représenter préciser les notions de temps et position dans l'espace. Ensuite, nous proposerons deux approches : une première considérant un nombre borné de sessions, et une seconde permettant une analyse dans un contexte non borné. L'ensemble des résultats théoriques présentés au cours de ce manuscrit, seront également implémentés et permettront l'analyse de sécurité d'un large nombre de protocoles. Au cours de ces analyses, nous porterons un intérêt particulier aux protocoles de paiement sans contact, et en particulier à deux nouveaux récemment proposés par Chothia *et al.* [CBC19] devant assurer la proximité physique des agents et ce, même dans des scénarios où le terminal de paiement est entièrement contrôlé par un agent malhonnête.

1. Un modèle symbolique avec temps et positions dans l'espace

Au cours de ce manuscrit, nous développerons un modèle symbolique permettant l'analyse des protocoles délimiteurs de distance. Ce modèle, basé sur le Pi-calcul appliqué [AF01], modélisera de manière fidèle le temps et les positions des agents au cours de l'exécution du protocole. Pour ce faire, les communications seront soumises à la contrainte physique précédemment citée, i.e. un message ne peut être reçu par un agent que si suffisamment de temps s'est écoulé depuis son émission afin qu'il puisse atteindre sa destination. Cette contrainte concernera évidemment les communications entre agents honnêtes, mais également celles impliquant des malhonnêtes. Dans le Chapitre 6 de ce manuscrit, nous étendrons ce modèle de communication afin de modéliser le caractère mobile des agents, i.e. la position d'un agent peut évoluer au cours d'une exécution du protocole. Cette extension permettra d'établir, pour la première fois, des preuves de sécurité pour des protocoles délimiteurs de distance prenant en compte le caractère mobile des participants.

Une fois le modèle de communication présenté, nous proposerons trois propriétés de sécurité qu'un protocole délimiteur de distance doit satisfaire : mafia fraud résistance, distance hijacking résistance, et terrorist fraud résistance. Chacune d'elles propose une définition formelle des différentes classes d'attaques présentées en Figure 3. Nous pouvons noter que nous ne ferons pas de distinction entre distance fraud et distance hijacking. En effet, nos définitions considérant un nombre arbitraire d'agents, tous situés à des positions arbitraires, la distance fraud n'apparait que comme un cas particulier de distance hijacking. Concernant la mafia fraud résistance, nous étendrons cette dernière dans le cadre des protocoles de paiement sans contact (voir Chapitre 6) afin de prouver la sécurité des protocoles dans le cadre de terminaux de paiement malicieux.

2. Vérification pour un nombre borné de sessions

Le première approche développée dans ce manuscrit consistera en une nouvelle procédure de vérification permettant l'analyse des protocoles délimiteurs de distance dans le cadre d'un nombre borné de sessions. Cette procédure étend celle déjà existante et implémentée dans l'outil Akiss [CCCK16]. Si cette dernière a été initialement développée pour vérifier des propriétés d'équivalence, nous l'adapterons pour des propriétés d'atteignabilité, comme le sont distance hijacking, mafia, et terrorist fraud résistance.

Nous établirons formellement sa correction et sa complétude. Si la correction de la procédure originelle permettra de déduire aisément celle de la nouvelle, la complétude sera plus complexe à établir. En effet, la contrainte physique imposée aux communications rendront incomplètes certaines optimisations présentées dans la procédure originelle. Ces dernières étant cruciales pour la terminaison, et donc l'utilisation en pratique, de la procédure, nous développerons des solutions théoriques afin de palier à cette limitation.

Finalement, cette nouvelle procédure sera appliquée dans le but d'analyser divers protocoles délimiteurs de distance. Nous présenterons et détaillerons donc différents aspects liés à son implémentations, avant de présenter l'étude de cas qui aura été menée.

3. Vérification pour un nombre non borné de sessions

Concernant la l'analyse des protocoles délimiteurs de distance dans le contexte d'un nombre non borné de sessions, nous développerons, dans ce manuscrit, deux approches différentes : la première établissant des résultats de réduction, la second étendant, dans un modèle plus riche, un résultat de causalité obtenu par Mauw *et al.* [MSTPTR18]. Ces deux approches permettant l'utilisation d'outils existants comme Proverif [Bla01] ou Tamarin [MSCB13].

Résultats de réduction

Cette première approche consiste à simplifier les analyses grâce à différents résultats de réductions. Tout d'abord, nous prouverons qu'il n'est pas nécessaire de considérer un nombre infini de topologies pour chaque classe d'attaque, comment mentionné précédemment. En effet, pour chacune d'elles, nous montrerons qu'il existe une unique topologie permettant de capturer l'ensemble des attaques existantes. Ces dernières, présentées en

Figure 4 [3], ne mettant en jeu que (au plus) quatre agents à des position pré-définies. Ensuite, concernant la terrorist fraud resistance, nous montrerons que, sous certaines hypothèses, il existe une stratégie de conspiration optimale pour mener à bien une telle attaque. L'analyse de sécurité pourra donc se focaliser sur cette unique stratégie.

L'ensemble de ces résultats de réduction permettra de mener à bien une large étude de cas de plus de 25 protocoles. En effet, les topologies réduites pourront être encodées dans un outil de vérification automatique existant, Proverif, au travers de la notion de *phase* proposée par cet outil. Cet encodage sera inspiré de [CGdR+15] mais, dans une volonté de rigueur, nous prouverons formellement la correction de ce dernier. Finalement, les résultats obtenus permettront de confirmer des résultats déjà connus, mais aussi d'établir de nouvelles preuves de sécurité. Ils permettront également de détecter des vulnérabilités encore inconnues.

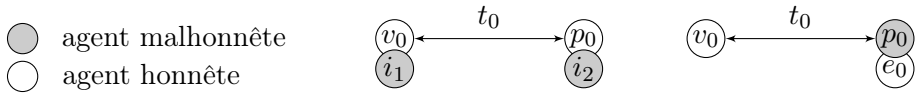


Figure 4: Topologies réduites où t_0 représente le seuil de proximité acceptée (gauche : topologie pour mafia fraud et terrorist fraud resistance, droite : topologie pour distance hiacking reistance).

Résultat de causalité

Comme nous le verrons au cours de ce manuscrit, la première approche, basée sur les résultats de réduction, ne permettra pas l'analyse des deux nouveaux protocoles de paiement sans contact récemment proposés par Chothia *et al.* [CBC19]. En effet, le modèle sous-jacent se révèlera trop restrictif ; tout comme l'ensemble des résultats existants dans la littérature.

Afin de dépasser cette limitation, nous étendrons le modèle symbolique précédemment introduit : nous permettrons l'échange de messages horodatés et étendrons la définition de mafia fraud résistance dans le cadre de vérifieurs malhonnêtes. Nous étendrons ensuite le résultat de causalité obtenu par Mauw *et al.* [MSTPTR18] à ce nouveau modèle. Ce résultat établit une équivalence entre la propriété de sécurité souhaitée pour les protocoles délimiteurs de distance (et donc exprimée dans un modèle avec temps et positions) et une propriété seulement basée sur l'ordre des actions exécutées au cours du protocole (et donc pouvant être exprimée dans une modèle sans temps et positions).

Cette nouvelle propriété, faisant abstraction du temps et des positions des agents, pourra alors être vérifiée par l'intermédiaire d'outils de vérification automatique existants comme Proverif [Bla01] ou Tamarin [MSCB13]. Cela nous permettra de présenter la première preuve formelle de sécurité pour ces deux nouveau protocoles.

L'ensemble des contributions présentées au cours de ce manuscrit ont été publiées dans des conférences internationales du domaine [DDW18, DD19, DDW19, BCDD20].

[3] Suivant les définitions présentées en Chapitre 2, nous obtiendrons la même topologie réduite pour mafia fraud et terrorist fraud resistance.

Contents

1 Introduction

Companies continuously look for new features that may improve the user experience for their customers. The development of contactless technologies like Wi-Fi, Radio Frequency Identification (RFID) or Near Field Communication (NFC) since the beginning of the 21st century has led to the birth of many innovations that are now used every day. A non-exhaustive list of such innovations includes e-passports, keyless entry systems, in-store self-checkout, transport-ticketing, tap-to-pay transactions... and many other applications. This list should keep growing in the next years thanks to the recent introduction of NFC chips in cars, smartphones and watches which provides even larger areas for innovations.

Looking at tap-to-pay transactions only, recent reports said that the global contactless market size was valued at USD 1.06 trillion in 2019 [1]. According to Visa, nearly half of all face-to-face Visa transactions around the world (excluding the USA) occur with a tap, i.e. contactlessly [2]. When considering Europe only, this rate grows up to two-thirds and is expected to keep on increasing in the next years. Recently, due to the COVID-19 crisis, Mastercard reported a 40% growth in contactless transactions worldwide in the first quarter of 2020 [3].

This quick adoption by sellers and customers is explained by the convenience and efficiency it brings. Typing a PIN code is fastidious, requires to remember it, and takes more time than a simple tap. With the tap-to-pay feature, transactions become easier and quicker. Moreover, the rise of tap-to-pay compatible checkout terminal enables people to get rid of their usual credit/debit card and wallet to solely use their smart watch or smartphone (which include an NFC interface today) for everyday purchases.

However, all these advantages for contactless technologies should not conceal new threats that are coming up. An important security concern of all these applications is to ensure the physical proximity of the tag and the reader. Indeed, a malicious person should not be able to abuse a remote and honest person to pay a bill on his behalf. While contact-based devices prevent such a scenario "by design", it becomes possible with contactless ones. Indeed, practical attacks have demonstrated that the

[1] https://www.grandviewresearch.com/industry-analysis/contactless-payments-market
[2] https://usa.visa.com/visa-everywhere/blog/bdp/2019/05/13/tap-to-pay-1557714409720.html
[3] https://mastercardcontentexchange.com/newsroom/press-releases/2020/april/mastercard-study-shows-consumers-globally-make-the-move-to-contactless-payments-for-everyday-purchases-seeking-touch-free-payment-experiences/

1

short range of the High Frequencies used by e.g. RFID of NFC tags are not sufficient to enforce physical proximity of the reader and the card involved in a transaction. They have been shown to be practical many times against keyless entry systems embedded in recent cars [FDC11] and any NFC/RFID card compliant with the ISO/IEC 14443 standard [Han05, FHMM10, SC13]. When looking at tap-to-pay transactions, practical attacks have been conducted too [CGdR+15]. On our side, we managed to perform quite easily such an attack against Mastercard and Visa credit cards controlling two "malicious" smartphones only. Messages were intercepted using an Android app relying on the NFC chip onboard the smartphones and relayed through Wi-Fi. Thanks to this simple setting, we were able to do payments with a gap of several meters between the card and the reader.

All of these attacks consist in man-in-the-middle (MiM) scenarios as described in Figure 1.1. An attacker controls two malicious devices: a rogue reader and a rogue tag. He then relays, via a very fast channel of communication (e.g. the Wi-Fi), all the messages sent by the two honest parties, i.e. the honest card and the honest terminal. Even though the two devices are distant, the reader authenticates the tag at the end of the attack. This kind of scenarios is also known as the Chess Grandmaster problem [Con76] in which a little girl defeats a chess grandmaster by playing two simultaneous games, each against a chess grandmaster, and replaying moves performed on a board to the other one. Note that, when looking at these attacks, the attacker may be simply relay messages, or be more active, e.g. making some computations to forge new messages. The contactless applications are thus subject to simple relay attacks, but also, more advanced distance-based frauds.

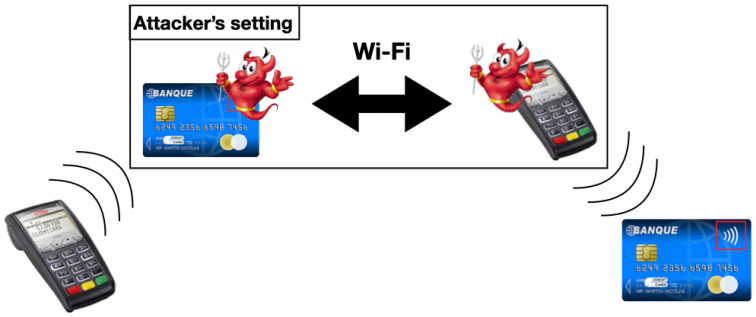


Figure 1.1: Relay attack scenario against payment protocols.

1.1. Distance-bounding protocols

In order to mitigate such attacks (and others), some security protocols, called *distance-bounding protocols*, have been designed. As usual, they rely on cryptographic primitives (e.g. encryption, signature, hash function) to ensure standard properties. When looking at the payment application, these properties are at least the confidentiality and the integrity of the data, and the mutual authentication of the card and the reader. In ad-

dition, distance-bounding protocols pretend to enforce the physical proximity of the two devices, i.e. whenever a reader authenticates a card, then both must be close. Since the card must prove its proximity to the reader, these two parties are respectively named a prover and a verifier.

1.1.1. A common design

In 1993, Brands and Chaum [BC93] proposed the first distance-bounding protocol. Since then, some surveys [BGL15, ABB+18] reported that more than 40 distance-bounding protocols have been proposed with few specificities each. However, we noteworthy observe that they all follow the same structure, presented in Figure 1.2. A distance-bounding protocol commonly starts by an initialisation phase in which the prover and the verifier exchange data to prepare the rapid phase that follows. To estimate its distance to the prover, the verifier measures the elapsed times during multiple challenge/response exchanges. Indeed, assuming the speed of communication c_0 is constant and known by the verifier, the distance between both parties can be estimated relying on the formula:

$$\mathsf{Dist}(V,P) \leq \frac{\mathsf{c}_0}{2} \times (t_{\mathtt{in}} - t_{\mathtt{out}} - t_{\mathtt{proc}})$$

where $t_{\mathtt{out}}$ is the time when the verifier sends the challenge, $t_{\mathtt{in}}$ the time when it receives the response, and $t_{\mathtt{proc}}$ the processing time needed to compute the answer on the prover side. A consequence is that the closer the prover is, the smaller the measurement will be. Finally, distance-bounding protocols may require a last phase in which the two parties exchange some extra data.

An example of such a protocol is the SPADE protocol [BGG+16] presented in Figure 1.3. It relies on an asymmetric encryption scheme, a signature scheme, and a pseudo-random function as cryptographic primitives. The prover initiates the protocol by sending a ciphertext that contains a freshly generated value n_P and a signature (of this value). The verifier replies by sending two fresh values m_V and n_V that are used to generate two ephemeral shared secrets H^0 and H^1. Then, the critical (or rapid) phase may start: the verifier sends a bit $c_i \in \{0,1\}$ and the prover replies by sending the i^{th} bit of H^{c_i}. This simple exchange that applies during the rapid phase allows to reduce the processing time $t_{\mathtt{proc}}$ as much as possible in order to get a tighter estimation of the distance. Finally, a transcript made of all the data exchanged along the protocol is sent by the prover. The verifier checks the transcript and, depending on the answers he received in the rapid phase and the times he measured, he decides, or not, to authenticate the prover.

1.1.2. Attack scenarios

The main goal of these protocols is to ensure the physical proximity of the participants. Regarding the literature, the different attack scenarios that may apply on distance-bounding protocols have been gathered into three main classes: mafia fraud [DGB87], terrorist fraud [DGB87], and distance fraud [Des88]. Recently, Cremers *et al.* [CRSC12] have discovered a new class of attacks called distance hijacking attacks. Nowadays, the security analyses are thus performed w.r.t. to these four classes of attacks. Each of them differs from the honesty and the locations of the main participants:

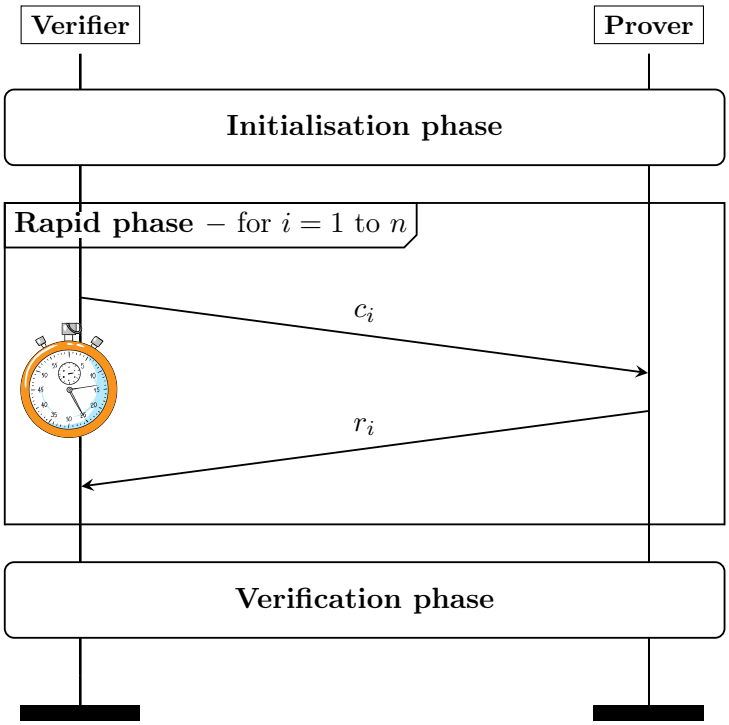


Figure 1.2: Common structure of a distance-bounding protocol.

- Distance fraud: a far-away and isolated dishonest prover tries to be authenticated by an honest verifier (see Figure 1.4a).
- Distance hijacking: a far-away dishonest prover tries to abuse honest parties in order to be authenticated by an honest verifier (see Figure 1.4b).
- Mafia fraud (or MiM): an attacker located between an honest verifier and an honest far-away prover tries to make the verifier authenticate the prover (see Figure 1.4c).
- Terrorist fraud: a far-away dishonest prover accepts to collude with an attacker to be authenticated once by an honest verifier. However the collusion should not give any advantage to the attacker for future attacks (see Figure 1.4d).

The mafia fraud, or MiM attack, is the class that encompasses the aforementioned relay attacks that applies against tap-to-pay transactions or keyless entry systems embedded in cars. The terrorist fraud is an advanced scenario in which a prover accepts to collude with the attacker once. However, this collusion should not enable the attacker to mount future attacks. By consequence, in such a scenario, the dishonest prover will not accept, for instance, to reveal his long-term keys. Instead, he will typically output session identifiers or ephemeral keys (e.g. H^0 and H^1 in the SPADE protocol) to give to

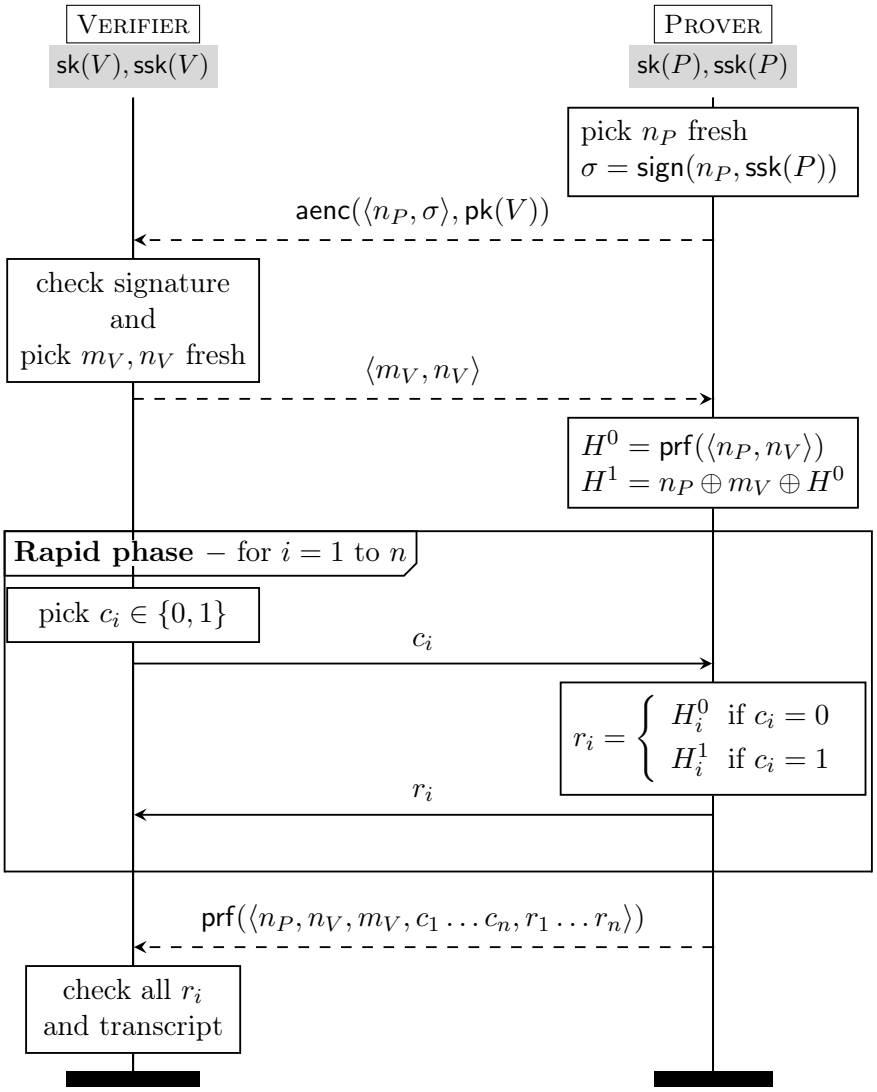


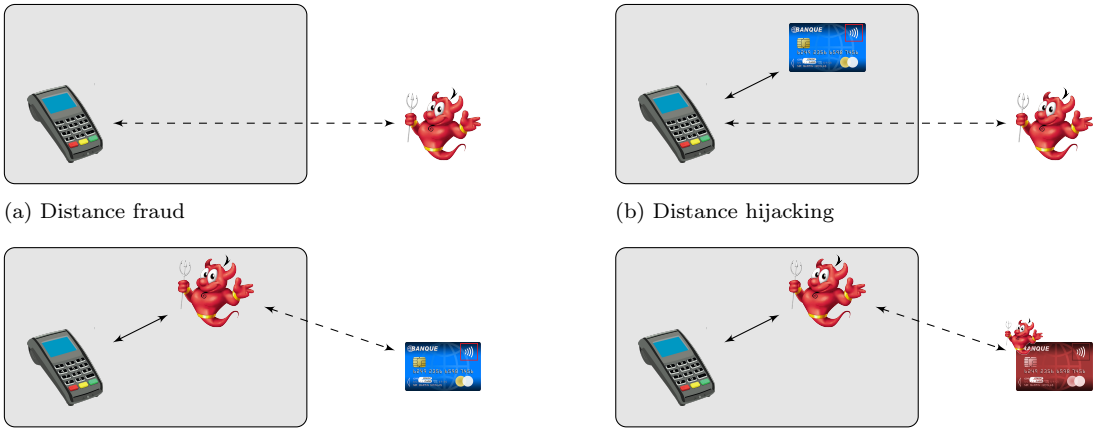
Figure 1.3: SPADE protocol [BGG+16].

his accomplice material that will not be reusable in other sessions. The two remaining classes of attacks, distance fraud and distance hijacking, are very similar. They consist in scenarios in which an attacker tries to be authenticated while being located far-away from the verifier. In case of distance fraud, this attacker is alone, whereas he can abuse honest parties in distance hijacking. Hence, distance hijacking attacks encompass distance frauds.

An important remark about these definitions is that they are not really restrictive about the topologies on which an attack may apply, e.g. the number of agents that are involved beside the three main participants is not restrained. By consequence, for

1

each class of attacks, one may imagine an infinite number of different topologies that match the requirements. Similarly, the definitions do no limit the number of sessions that each agent is willing to execute. Designing a protocol which effectively ensures physical proximity is therefore difficult.



(a) Distance fraud

(b) Distance hijacking

(c) Mafia fraud

(d) Terrorist fraud

Figure 1.4: The four usual classes of attacks when analysing distance-bounding protocols.

1.1.3. Security analysis

Due to their critical applications, e.g. payment protocols, proving the correctness of distance-bounding protocols is requested. For a long time, these security analyses consisted in analysing the protocols against well-known attacks only, and prove their inapplicability. These analyses were mainly focused on mafia fraud considering the simple pre-ask, post-ask and early-reply attack strategies. These last define the behaviour of the attacker: in the pre-ask strategy, he performs a rapid phase with the remote prover in advance and then tries to re-use the answers he received to pass the rapid phase on the verifier side in the pre-ask strategy. In the post-ask strategy, the attacker first executes the rapid phase with the verifier and then with the remote prover. Finally the early-reply strategy applies for distance fraud and let the attacker anticipate the outputs of the challenges, by sending random answers in advance. All these analyses provide a limited confidence in the correctness of the protocols since they focus on rather specific attacks only.

More recently, computational models have been designed to formally verify distance-bounding protocols in a more general setting. In these models, messages are represented by bitstrings, an attacker is any probabilistic polynomial time algorithm, and an attack applies if there exists an attacker with a non-negligible probability of success. These features are accurate enough to precisely analyse distance-bounding protocols and obtain strong security guarantees. However, this great expressiveness is also the main drawback of this approach: in order to take the numerous specificities of each protocol into account, various models have been proposed, which makes the comparison of the protocols diffi-

cult. In 2011, Dürholz *et al.* [DFKO11] proposed a framework with the aim to establish relations between frauds. They established that the frauds are mainly independent from one another. In the same time, Avoine *et al.* [ABK+11] proposed another framework with the aim to standardise the analyses of distance-bounding protocols and make the comparisons possible. Recently, numerous protocols have been analysed in this framework [ABB+18]. Unfortunately, these two works required tedious hand-written proofs for each of them. Automation appears to be a very difficult task in these models.

Another approach for protocol verification is symbolic verification. At the price of few abstractions (e.g. messages by terms) it allows for an automatic verification.

1.2. Symbolic verification

Symbolic verification of cryptographic protocol is a well-known approach to allow for automation. Efficient tools exist and have shown their usefulness to prove the (in)security of real-world protocols.

1.2.1. State-of-the-art

Most of the attacks that apply on security protocols do not require to break any cryptographic primitive. Instead they exploit weaknesses in the flow of the protocol. Symbolic verification aims at proving the security of the protocols looking for such logical attacks. A common abstraction of such models is to abstract cryptographic primitives by symbols of functions representing a perfect cryptography (e.g. an encryption primitive is assumed not to leak any information regarding the plaintexts to anyone who does not know the correct decryption key). Messages exchanged during a protocol are then represented by terms that are built over the symbols of function. Another common abstraction applies on the attacker model: symbolic models commonly assume a Dolev-Yao [DY83] attacker which is able to intercept, forge and send messages at any time of an execution. It models an omniscient and omnipresent attacker.

Even considering these two abstractions,the automatic verification of cryptographic protocols remains a difficult problem. In most cases, proving security properties for an expressive enough class of protocols appears as an undecidable problem [MSDL99]. Even though, procedures and tools, e.g. Proverif [Bla01] or Tamarin [MSCB13], have been designed to tackle this problem. They proved their efficiency and usefulness to analyse real-world protocols. Some famous examples are the proofs of security for TLS 1.3 Draft 18 [BBK17], or for a specific version of the e-voting protocol Belenios [CGG19]. They have also proved their utility to detect flaws in largely deployed protocols like the Single-Sign-On protocol used by e.g. Google Apps [ACC+08].

Some decidability results have been proposed for the verification of cryptographic protocols in restricted setting. For example, when bounding the number of sessions, the problem has been proved NP-complete [RT03] and tools implementing decision procedures have been proposed, e.g. Akiss [CCCK16] or Deepsec [CKR18].

1

1.2.2. Regarding distance-bounding protocols

Unfortunately, because of the omniscient and omnipresent attacker, the existing models and tools relying on symbolic models cannot faithfully model time and locations. Therefore, they are not suitable to analyse distance-bounding protocols. To overcome this limitation, few models have been proposed [MPP+07, MBK10, BCSS11, LSLD15, CGdR+15]. They all develop symbolic models in which messages take time to travel from one location to another.

In 2007, Meadows *et al.* [MPP+07] proposed the first symbolic model to analyse distance-bounding protocols which relies on an authentication logic that faithfully features locations and time. Thanks to this logic, authors proved the security of a new protocol they proposed (see Figure 1.5) when the answer to the challenge is either $F(n_V, n_P, P) = \langle n_V, P, n_P \rangle$, or $F(n_V, n_P, P) = \langle n_V, n_P \oplus P \rangle$, or $F(n_V, n_P, P) = \langle n_V \oplus \mathtt{f}(P, n_P) \rangle$. Sadly, their framework is quite complex and does not allow for automation; all of these security proofs are done manually.

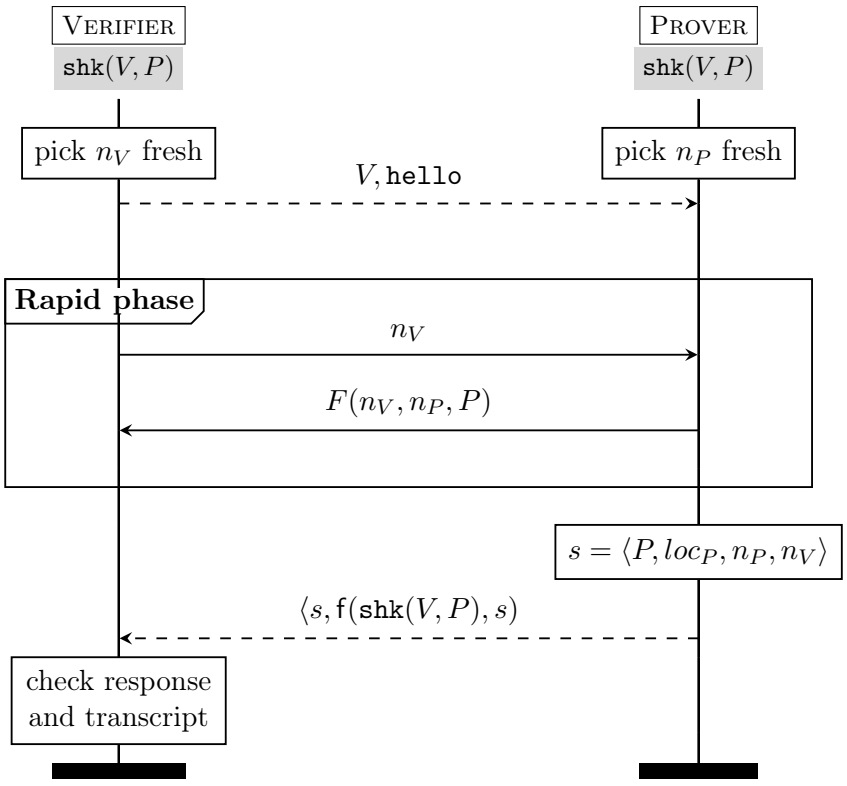


Figure 1.5: Meadows *et al.* protocol [MPP+07].

In 2011, Basin *et al.* [BCSS11] proposed another symbolic framework to analyse distance-bounding protocols. It extends an existing model based on multiset rewriting rules with time and locations in a general setting. The protocol descriptions and the deduction capabilities of the attackers remain unchanged; the unique difference with the usual model lies in the network rule: a message m can be received at time t_R by an agent B if, and only if, it has been sent by an agent A soon enough to let the message travel from A's location to B's, i.e., at time t_S such that $t_R \geq t_S + \mathsf{Dist}(A, B)$. Besides, the authors propose a formal definition of physical proximity. They consider the following security property:

$$\forall \mathsf{tr} \in Tr(Proto),\ \mathsf{claim}(V, P, d) \Rightarrow d \leq \mathsf{Dist}(V, P)$$

where $Tr(Proto)$ is the set of all the possible traces of executions of the protocol under study, and $\mathsf{claim}(V, P, d)$ is an event triggered by the verifier V when it authenticates the prover P with a proximity threshold d. Typically, $d = \frac{t_{\mathtt{in}} - t_{\mathtt{out}}}{2}$, where $t_{\mathtt{in}}$ and $t_{\mathtt{out}}$ are the two timestamps that identify the beginning and the end of the challenge response exchange.

This property faithfully models physical restrictions and has been used to analyse distance-bounding protocols. Instead of relying on hand-written proofs, authors encoded their model into the theorem-proving assistant Isabelle/HOL [NPW02] in order to provide a partially automated framework. Indeed, many lemmas are generic enough to be protocol-independent and thus re-usable across different analyses. The approach followed by the authors across this paper demonstrates a noteworthy effort to provide a rigorous framework to analyse distance-bounding protocols. Unfortunately, the automation remains incomplete since protocol-dependent lemmas must be manually defined by the end-user (typically between 8 and 20 in their case studies).

In 2015, Chothia *et al.* [CGdR+15] aimed at verifying a new payment protocol, named PaySafe, designed to ensure physical proximity in contactless transactions. To this aim, they developed an approach based on the well-known Proverif tool [Bla01] that allows for an automatic verification. They managed to encode in the Proverif tool rather simple topologies that involve only two locations: the reader's/verifier's location and a remote location (typically for the honest prover when considering relay attacks). This encoding relies on the notion of phases provided by Proverif: three phases are defined, one for each step of the protocol, following the structure of distance-bounding protocols presented in Figure 1.2. Agents at the verifier's location are allowed to act during any phase, but remote agents cannot act during the rapid phase (intuitively they are too far to respond to the challenge).

The main limitations of this approach are two-fold: first, instead of taking the infinite number of topologies that match the requirements for each class of attacks into account, they focus on a unique and rather simple one they encode in Proverif. Then, they do not formally justify the encoding of the simple topologies they propose using phases. Despite that, this is an appealing approach and some of the results presented in this manuscript take some inspiration from this work.

Meanwhile we established the results presented in this manuscript to allow for an automatic verification of distance-bounding protocols, Mauw *et al.* extended the Basin *et al.*'s model to cope with a complete automation [MSTPTR18, MSTPTR19]. As a consequence, in addition to precisely comparing our model and framework to Chothia *et al.*'s [CGdR$^+$15, CdRS18], we will compare to Mauw *et al.*'s too. These discussions will be presented in Chapters 2 and 5.

1.3. Contributions

In this manuscript, we tackle the issue of the automatic verification of distance-bounding protocols. To this end, we will develop symbolic models, procedures, and reduction results. In the spirit of [CLC03, CDD12], the latter will allow to get rid of specific difficulties of distance-bounding protocol verification (topologies and collusion behaviours), and leverage existing tools. Finally we provide automatic frameworks to analyse distance-bounding protocols. We applied them and managed to prove the (in)security of numerous distance-bounding protocols. In particular, we applied our results to analyse payment protocols.

In this manuscript, the contributions are four-fold:

(1) We develop symbolic models to analyse distance-bounding protocols. Based on the applied pi-calculus [AF01], it allows to faithfully model the physical constraints by modelling agent locations and time. In Chapter 6, we even present the first calculus which models mobility, i.e. in which agents can move during an execution, that is suitable to analyse distance-bounding protocols. About the physical constraints, we model that a message takes time to travel from one location to another. Finally, we propose formal definitions for each class of attacks, formalising the informal ones presented in Section 1.1.2 by considering an arbitrary number of agents, each located at arbitrary locations. Note that, we do not make any difference between distance-fraud and distance-hijacking attack since we aim at verifying distance-bounding protocols considering an arbitrary number of agents. When looking at payment protocols, we consider a new class of attacks involving malicious readers [CBC19] and accordingly extend an existing security property [MSTPTR18].

(2) We extend the underlying procedure of the Akiss tool [CCCK16] to take time and locations into account. This procedure addresses the automatic verification of distance-bounding protocols w.r.t. reachability properties when considering a bounded number of sessions. The soundness and the completeness of the procedure have been formally proved. Finally, it has been implemented on top of the Akiss tool and successfully applied to prove the (in)security of some distance-bounding protocols.

(3) We propose two reduction results in the spirit of [CDD12] to reduce the number of topologies that must be considered when analysing a distance-bounding protocol, and to reduce the number of collusion behaviours that must be considered when considering terrorist frauds. Precisely, we reduce from infinitely many to only one the number of topologies that must be taken into account for each class of attacks, and, under some hypotheses, we prove the existence of a most-general collusion behaviours the analyses can focus on.

(4) Finally, thanks to the previous contributions, we propose a comprehensive case studies analysis (more than 25 protocols). It relies either on our newly implemented procedure for bounded number of sessions, or the Proverif tool to verify the security properties we proposed considering an unbounded number of sessions.

All these contributions have been published in [DDW18, DD19, DDW19, BCDD20].

1.4. Outline

This manuscript is made of 2 parts and 6 chapters. The first part deals with the symbolic verification of standard distance-bounding protocols, i.e. protocols following the global shape presented before. We propose two approaches: one which consists in designing and implementing a new procedure for a bounded number of sessions, and one relying on reductions results that allows to leverage existing tools. The second part focuses on the verification of two novel EMV-payment protocols that embed specific features which make the previous results not suitable.

Part I: Verification of standard distance-bounding protocols

Chapter 2 presents a generic model that will be used across Part I. It extends the applied pi-calculus to take time and locations in consideration in order to define the three security properties we will be interested in when analysing distance-bounding protocols: mafia fraud resistance, terrorist fraud resistance, and distance hijacking resistance. This calculus has been designed to remain as general as possible and allows to model a wide class of protocols.

Chapter 3 presents the theoretical development of a procedure that will be applied to decide reachability properties (e.g. mafia fraud, terrorist fraud, and distance hijacking) on timed protocols considering a bounded number of sessions. In a bounded setting, protocols can be described by a finite set of traces that correspond to the possible interleavings of the roles. In this chapter, we prove the procedure sound and complete when considering such traces. Its termination will be demonstrated in practice across case studies presented in Chapter 5. Since the procedure builds on the underlying procedure of the Akiss tool [CCCK16], in Chapter 5 we will also discuss how our procedure has been implemented on top of it.

Chapter 4 presents two reduction results that allow to consider only two simple topologies when analysing distance-bounding protocols. Moreover, under some hypothesis, it reduces the number of collusion behaviours that must be considered when looking at terrorist frauds: there exists a most general behaviour that subsumes the others. These two reduction results will be crucial to ease the automatic verification of distance-bounding protocols.

Chapter 5 applies the results presented in the two previous chapters to perform a comprehensive case studies analysis. It discusses few implementations aspects, and bridges the gap between the theoretical model and the practical one, implemented behind the tools. The efficiency of both approaches is proved in practice, and, besides agreeing on (in)security proofs of already studied protocols, it establishes the first proof of (in)security for others.

Part II: Analysis of two novel EMV-payment protocols

Chapter 6 presents a symbolic model that fits the specificities of EMV-payment protocols. In particular it extends the security property presented by Mauw *et al.* [MSTPTR18] to prevent relay attacks (i.e. mafia frauds) in the context of mobile agents and malicious readers/verifiers. We prove this timed security-property equivalent to a causality-based property which solely relies on the order of the actions during an execution. This last property can thus be checked by any existing tool. We prove the correctness of the two novel EMV-payment protocols proposed in [CBC19], relying on the Proverif tool.

Part I

Verification of standard distance-bounding protocols

A symbolic model with time and locations 2

In this chapter, we describe the symbolic model we will use along this thesis. Similarly to the standard applied pi-calculus [AF01], our model relies on a term and a process algebra to define what messages and protocols are. However, all these notions need to be extended to faithfully model time and agent locations. In particular, a message must take time to travel from one location to another, and agents must be able to perform time checks. We finish this chapter by defining the three security properties we will consider when analysing a distance-bounding protocol: mafia fraud, terrorist fraud and distance hijacking attack resistance.

Contents

2.1. Messages

As usual in the symbolic setting, we model messages through a term algebra. Following the approach developed in [Bla01], we consider both an equational theory and a reduction relation to represent the properties of the cryptographic primitives. In addition, we consider time expressions to handle time checks performed during an execution of a protocol.

2.1.1. Term algebra

We consider two infinite and disjoint sets of *names*: $\mathcal{N}$ is the set of basic names, which are used to represent keys and nonces, whereas $\mathcal{A}$ is the set of agent names, i.e. names which represent the agent identities. We denote $\mathbb{R}_+$ the set of non-negative real numbers and Σ_0 an infinite set of public constants, i.e., known by the attacker. Finally, we consider three infinite sets of variables: $\mathcal{X}$ refers to unknown parts of messages, $\mathcal{W}$ contains the variables used to store messages learnt by the attacker, and time variables belong to $\mathcal{Z}$. During an execution variables from $\mathcal{Z}$ should only be instantiated by elements in $\mathbb{R}_+$.

To model cryptographic primitives we assume a signature Σ, i.e., a set of function symbols together with their arity. These elements are split into *constructor* and *destructor* symbols, i.e., $\Sigma = \Sigma_c \cup \Sigma_d$. Constants are considered as constructor function symbols of arity 0. We note $\Sigma^+ = \Sigma \cup \Sigma_0$ and $\Sigma_c^+ = \Sigma_c \cup \Sigma_0$. Given a signature $\mathcal{F}$, and a set of atomic data A, we denote by $\mathcal{T}(\mathcal{F}, A)$ the set of terms built from atomic data A by applying function symbols in $\mathcal{F}$. A constructor term is a term in $\mathcal{T}(\Sigma_c^+, \mathcal{N} \cup \mathcal{A} \cup \mathbb{R}_+ \cup \mathcal{X} \cup \mathcal{Z} \cup \mathcal{W})$. We note $vars(u)$ the set of message, time and frame variables occurring in a term u. A *message* is a ground constructor term u, i.e., $vars(u) = \emptyset$.

To model attacker capabilities, the signature Σ is split into public and private symbols Σ_{pub} and Σ_{priv}. We note $\Sigma_{\mathsf{pub}}^+ = \Sigma_{\mathsf{pub}} \cup \Sigma_0$. An attacker is able to make some computations to deduce new messages applying public function symbols on top of terms he already knows and that are available through variables in $\mathcal{W}$. Formally, a computation done by the attacker is a term in $\mathcal{T}(\Sigma_{\mathsf{pub}}^+, \mathcal{W} \cup \mathbb{R}_+)$. For technical reasons we distinguish two kinds of computations: *timed-recipes* which are terms in $\mathcal{T}(\Sigma_{\mathsf{pub}}^+, \mathcal{W} \cup \mathbb{R}_+)$, and *recipes* which are terms in $\mathcal{T}(\Sigma_{\mathsf{pub}}^+, \mathcal{W})$.

A substitution $\sigma = \{x_1 \to u_1, \ldots, x_n \to u_n\}$ is a mapping from variables $x_1, \ldots, x_n \in \mathcal{X} \cup \mathcal{W} \cup \mathcal{Z}$ to terms $u_1, \ldots, u_n$. We note $dom(\sigma) = \{x_1, \ldots, x_n\}$ its domain and $img(\sigma) = \{u_1, \ldots, u_n\}$ its image. The application of a substitution σ to a term u is noted $u\sigma$ and consists in replacing all the occurrences of x_i in u by u_i (for $1 \leq i \leq n$).

Given terms $u_1, \ldots, u_n$, we say that a substitution σ is a *unifier* for $u_1, \ldots, u_n$ if $u_1\sigma = \ldots = u_n\sigma$. We note (when it exists) $\mathsf{mgu}(u_1, \ldots, u_n)$, the *most general unifier* of $u_1, \ldots, u_n$, i.e. the substitution θ such that for all unifiers σ of $u_1, \ldots, u_n$, there exists a substitution τ such that $\sigma = \theta\tau$.

We define as usual, the subterms, noted $st(u)$, and the positions of a term u. We note $u|_p$ the subterm at position p in u and $u[v]_p$ the term u in which the subterm at position p has been replaced by v.

Example 2.1. *To model the SPADE protocol presented in Chapter 1, we consider the following signature:* $\Sigma = \{\mathsf{prf}/1, \mathsf{pk}/1, \mathsf{sk}/1, \mathsf{aenc}/2, \mathsf{adec}/2, \mathsf{sign}/2, \mathsf{check}/2, \mathsf{spk}/1, \mathsf{ssk}/1, \langle\ ,\ \rangle/2, \mathsf{proj}_1/1, \mathsf{proj}_2/1, \oplus/2, \mathbb{0}/0, \mathsf{answer}/3, \mathsf{eq}/2, \mathsf{ok}/0\}$. *The* prf *symbol models a pseudo-random function. The symbols* aenc, adec, pk *and* sk *model an asymmetric encryption scheme together with a decryption algorithm and the corresponding public/private keys. The symbols* sign, check, spk *and* ssk *model a signature scheme. The symbols* sign *and* ssk *are used to sign a message using a private key while* check *and* spk *are used to verify a signature using the corresponding public key. The symbols* $\langle\ ,\ \rangle$, proj_1 *and* proj_2 *model pairs and projections. The exclusive-or operator is modelled by the symbol* $\oplus$ *and we use the symbol* $\mathbb{0}$ *as the identity element. Since we cannot model messages at the bit-level, the symbol* answer *is used to represent in an abstract way the response provided by the prover during the rapid phase. Finally, we consider the symbols* eq *and* ok *to check message equalities.*

Amongst all the symbols in Σ, *we consider* adec, check, proj_1, proj_2 *and* eq *as destructors. Only* sk *and* ssk, *that model private keys associated to the agent name given in argument, belong to* Σ_{priv}. *Note that an attacker will not be able to derive his own material alone, it will be provided through an initial knowledge (see Section 2.3.1).*

Remark 2.1. *For readability purposes, we may use, in the rest of this manuscript the following notations:* $\langle t_1, \ldots, t_n\rangle = \langle t_1, \langle t_2, \ldots, \langle t_{n-1}, t_n\rangle\rangle\rangle$ *for the concatenation of* n *elements, and* $\pi_{j,n}(\langle t_1, \ldots, t_n\rangle) = \mathsf{proj}_1 \circ \mathsf{proj}_2^{j-1}(\langle t_1, \ldots, t_n\rangle)$ *the* j*-th projection on a* n*-uplet, with* $j \in \{1, \ldots, n-1\}$, *while, for* $j = n$, *we define* $\pi_{j,n} = \mathsf{proj}_2^n(\langle t_1, \ldots, t_n\rangle)$.

2.1.2. Equational theory

The algebraic properties of the constructor symbols are modelled through an *equational theory*. It consists of a finite set $\mathtt{E}$ of equations of the form $u = v$ where $u, v \in \mathcal{T}(\Sigma_c, \mathcal{X})$. It induces an equivalence relation $=_{\mathtt{E}}$ over constructor terms. Formally $=_{\mathtt{E}}$ is the least congruence on constructor terms, which contains all the equations $u = v$ in $\mathtt{E}$ and closed under substitutions. We assume that $\mathtt{E}$ models a non-trivial theory, i.e. there exists u and v such that $u \neq_{\mathtt{E}} v$.

Example 2.2. *The equational theory derived from the following set* $\mathtt{E}_{\mathsf{xor}}$ *reflects the algebraic properties, in particular the associativity and commutativity, of the exclusive-or operator.*

$$(x \oplus y) \oplus z = x \oplus (y \oplus z) \qquad x \oplus y = y \oplus x \qquad x \oplus \mathbb{0} = x \qquad x \oplus x = \mathbb{0}.$$

2.1.3. Rewriting system

The meaning of destructor symbols is provided through a set $\mathcal{R}$ of rewriting rules. Formally, a rule is of the form $\mathsf{g}(t_1, \ldots, t_n) \to t$ where $\mathsf{g} \in \Sigma_d$ and $t, t_1, \ldots, t_n \in \mathcal{T}(\Sigma_c, \mathcal{X})$. A term u can be rewritten in v if there exists a position p in u, and a rewriting rule $\mathsf{g}(t_1, \ldots, t_n) \to t$ such that $u|_p =_{\mathtt{E}} \mathsf{g}(t_1, \ldots, t_n)\theta$ for some substitution θ, and $v = u[t\theta]_p$. Moreover we assume that $t\theta, t_1\theta, \ldots, t_n\theta$ are constructor terms.

As usual, we only consider sets of rules $\mathcal{R}$ that yield to a *convergent* rewriting system (modulo $\mathtt{E}$):

- $\mathcal{R}$ is terminating, i.e. there is no infinite sequence $u \to u_1 \to u_2 \to \dots$
- $\mathcal{R}$ is confluent, i.e. for all terms u, u_1, u_2 such that $u \to^* u_1$ and $u \to^* u_2$, there exist v_1, v_2 such that $u_1 \to^* v_1$ and $u_2 \to^* v_2$ and $v_1 =_{\mathtt{E}} v_2$, where $\to^*$ denotes the transitive closure of $\to$.

Given a term u we note $u{\downarrow}$ its normal form. By convention, we note $u{\downarrow} = \bot$ when u does not reduce to a constructor term.

Example 2.3. *Continuing Examples 2.1, the property of the destructor symbols is given through $\mathcal{R}_{ex}$, the set made of the following rules:*

$$\begin{array}{ll} \mathsf{adec}(\mathsf{aenc}(x, \mathsf{pk}(y)), \mathsf{sk}(y)) \to x & \mathsf{proj}_1(\langle x, y\rangle) \to x \\ \mathsf{check}(\mathsf{sign}(x, \mathsf{ssk}(y)), \mathsf{spk}(y)) \to x & \mathsf{proj}_2(\langle x, y\rangle) \to y \\ \mathsf{eq}(x, x) \to \mathsf{ok}. & \end{array}$$

These rules exactly model the expected behaviours of an asymmetric encryption scheme, a signature or a pairing function. They allow to decrypt a ciphertext applying the symbol adec *together with the correct secret key, check the validity of a signature and retrieve the signed message applying the symbol* check *with the correct public key, and extract the first (resp. second) component of a pair applying the symbol* proj_1 *(resp.* proj_2*). Finally, the semantics given to the symbol* eq *allows to reduce a term* $\mathsf{eq}(u, v)$ *to* ok *as soon as $u{\downarrow}$ and $v{\downarrow}$ are constructor terms and $u{\downarrow} =_{\mathtt{E}} v{\downarrow}$. Note that this model assumes a perfect cryptography, i.e. given an encrypted message an attacker cannot retrieve any bit of information of the plaintext. He can neither forge a valid signature without the whole and correct private key.*

Remark 2.2. *Considering both an equational theory and a rewriting system provides a rich framework to model primitives. Indeed each of them allows to model specific behaviours. For example, an equational theory allows to faithfully model the exclusive-or operator including its associativity and commutativity properties, which is not possible through a convergent rewriting system. On the other side, a rewriting system allows to model primitives that may fail. For example, considering the rewriting rule presented in the previous example, the term* $\mathsf{adec}(\mathsf{ok}, \mathsf{pk}(id))$ *(for some $id \in \mathcal{A}$) is not a message (because it is not a constructor term) and thus cannot be sent over the network; the computation fails. On the contrary, if the encryption scheme is modelled by the equation* $\mathsf{adec}(\mathsf{aenc}(x, \mathsf{pk}(y)), \mathsf{sk}(y)) = x$ *considering both* aenc *and* adec *as constructor symbols, then* $\mathsf{adec}(\mathsf{ok}, \mathsf{pk}(id))$ *is a message and can be sent/received over the network. It then models a decryption scheme which never fails; it always returns a term which may appear as a message of the protocol.*

Depending on the encryption scheme you consider, the first (with a rewriting rule) or the second (with an equation) solution may be preferable.

2.1.4. Timing constraints

To model time, we use non-negative real numbers, i.e. $\mathbb{R}_+$, and the specific set of variables $\mathcal{Z}$. Time expressions are inductively constructed by applying arithmetic symbols (e.g., $+$, $\times$, $-\ldots$) to time expressions starting with the initial set $\mathbb{R}_+ \cup \mathcal{Z}$. A timing constraint is then of the form $t_1 \sim t_2$ with $\sim \in \{<, \leq, =\}$ and t_1 and t_2 two time expressions.

Our theoretical framework does not constrain the set of arithmetic symbols we use as soon as there exists a procedure to decide whether a finite set of timing constraints is satisfiable or not. This point will be discussed in Chapter 5 when we will present the implementations of the procedures we proposed.

Example 2.4. *Timing constraints are often used to model the check performed by the verifier during the rapid phase. Typically, for the SPADE protocol presented in Figure 1.3, the constraint has the form $z_2 - z_1 < t_d$ with $z_1, z_2 \in \mathcal{Z}$ and $t_d \in \mathbb{R}_+$. Indeed, z_1 and z_2 will be instantiated by the global times at which the verifier initiates the rapid phase by sending his challenge and the one at which he stops it receiving the response. The time t_d corresponds to the bound defining his proximity.*

2.2. Protocols

Protocols describe how messages should be exchanged between participants. Similarly to the applied pi-calculus [AF01], they are modelled by a process algebra accompanied with an operational semantics.

2.2.1. Process algebra

Processes are generated by the following grammar:

$$\begin{array}{rcl} P & := & 0 \\ & | & \mathtt{new}\ n.P \\ & | & \mathtt{in}^z(x).P \\ & | & \mathtt{out}^z(u).P \\ & | & \mathtt{let}_{\mathtt{mess}}\ x = v\ \mathtt{in}\ P \\ & | & \mathtt{let}_{\mathtt{time}}\ z = v\ \mathtt{in}\ P \\ & | & \mathtt{if}_{\mathtt{time}}\ t_1 \sim t_2\ \mathtt{then}\ P \\ & | & \mathsf{end}(u_1, u_2) \end{array}$$

with $x \in \mathcal{X}$, $z \in \mathcal{Z}$, $u, u_1, u_2 \in \mathcal{T}(\Sigma_c^+, \mathcal{N} \cup \mathcal{A} \cup \mathbb{R}_+ \cup \mathcal{X})$, $v \in \mathcal{T}(\Sigma^+, \mathcal{N} \cup \mathcal{A} \cup \mathbb{R}_+ \cup \mathcal{X})$ and $t_1 \sim t_2$ a timing constraint.

All these commands model the behaviour of a participant. The command 0 is the null process, meaning that no action is pending. The command $\mathtt{new}\ n$ allows to pick random values (unknown from the attacker). The $\mathtt{in}^z(x)$ and $\mathtt{out}^z(u)$ commands model communications, respectively the input and the output of a message. The variable z will store the current global time of the system when the command is executed. Whenever this time is useless, i.e. does not occur in the remaining process P, the variable z may

be omitted. The `let` commands allows to perform local computations. The $\texttt{let}_{\texttt{mess}}$ command succeeds and continues by executing P in which x has been substituted by v if v reduces in a constructor term. Otherwise, the process is blocked. Similarly, the $\texttt{let}_{\texttt{time}}$ command succeeds if v reduces to a time, i.e., a non-negative real number. Timing constraints are checked using the $\texttt{if}_{\texttt{time}}$ command: if the constraint is satisfied then the process P is executed, otherwise the process is blocked. Finally the command $\mathsf{end}(u_1, u_2)$, which can only occur at the end of a process, will be used to express the security properties.

Remark 2.3. *We do not consider conditionals for messages. Indeed the symbol* eq *defined in Example 2.3 allows to perform equality tests using the* $\texttt{let}_{\texttt{mess}}$ *command: given two terms* u, v*, the term* $\mathsf{eq}(u, v)$ *reduces to a constructor term,* ok*, and let the process progress if, and only if,* $u{\downarrow}$ *and* $v{\downarrow}$ *reduce to constructor terms and* $u{\downarrow} =_{\mathtt{E}} v{\downarrow}$*. This models an equality testing command. However, it is important to note that it does not allow to model else branches (i.e. dis-equality testing).*

Remark 2.4. *Process algebras based on the applied pi-calculus often define two extra commands: the parallel composition and the replication. The first models that two processes can run in parallel while the second models that a given process can be executed an arbitrary number of times. It allows for a verification with an unbounded number of sessions. These two commands are deliberately omitted in our syntax: they will be inherent to our model following the definition of configuration presented in Section 2.2.2.*

As usual, we denote $bv(P)$ the set of bound variables in P. These are variables introduced by an input, a `let` command, or an annotation over an input/output. Similarly we note $bn(P)$ the set of bound names in P, i.e., those introduced by a `new` command. On the opposite, we note $fv(P)$ the set of free variables, and $fn(P)$ the set of free names occurring in P. These are variables or names that are not bound.

We consider parametrised processes, denoted $P(\mathsf{x}_0, \ldots, \mathsf{x}_n)$, where $\mathsf{x}_0, \ldots, \mathsf{x}_n$ are specific variables that belong to the set $\mathcal{X}_{\mathcal{A}}$ disjoint from $\mathcal{X}$, $\mathcal{W}$ and $\mathcal{Z}$. They will be instantiated by agent names and, in particular, x_0 will be the agent who executes the process. Distance-bounding protocols are two-party protocols between a verifier and a prover. A *distance-bounding (DB) protocol* $\mathcal{P}$ is thus a pair $(\mathtt{V}(\mathsf{x}_0, \mathsf{x}_1), \mathtt{P}(\mathsf{x}_0, \mathsf{x}_1))$ such that:

- $fn(R) = \emptyset$; and
- $fv(R) \subseteq \{\mathsf{x}_0, \mathsf{x}_1\}$; and
- R does not contain any command $\mathsf{end}(u_i, u_j)$

for $R = \mathtt{V}(\mathsf{x}_0, \mathsf{x}_1)$ or $\mathtt{P}(\mathsf{x}_0, \mathsf{x}_1)$. The parametrised process $\mathtt{V}(\mathsf{x}_0, \mathsf{x}_1)$ is called the *verifier role* while $\mathtt{P}(\mathsf{x}_0, \mathsf{x}_1)$ is called the *prover role*. We note $\mathtt{V}_{\mathsf{end}}(\mathsf{x}_0, \mathsf{x}_1)$ the parametrised process $\mathtt{V}(\mathsf{x}_0, \mathsf{x}_1)$ ending with the command $\mathsf{end}(\mathsf{x}_0, \mathsf{x}_1)$ instead of the null process 0. This specific process will used to identify the targeted session of the verifier role used to define the security properties

Example 2.5. *As a running example, we consider the SPADE protocol presented in Figure 1.3. For modelling purposes, few abstractions must be done: first, the rapid phase relying on bit exchanges is collapsed into a single challenge/response exchange in which the challenge is modelled by a fresh name unknown from the attacker. As a consequence, we can no longer define the responses as precisely as in Figure 1.3. We thus abstract the answer by the uninterpreted symbol of function* answer *taking the challenge and the two pre-computed values H^0 and H^1 as argument. The abstracted protocol is presented in Figure 2.1 and the two roles are described as follows.*

On the first hand, the prover role is quite simple. Since there is no time consideration, it only involves standard commands from the applied pi-calculus.

$$
\begin{array}{rcl}
\mathtt{P}(\mathsf{x}_0, \mathsf{x}_1) & := & \mathtt{new}\ n_P. \\
& & \mathtt{out}(\mathsf{aenc}(\langle n_P, \mathsf{sign}(n_P, \mathsf{ssk}(\mathsf{x}_0))\rangle, \mathsf{pk}(\mathsf{x}_1))). \\
& & \mathtt{in}(x_1). \\
& & \mathtt{let}_{\mathtt{mess}}\ x_m = \mathsf{proj}_1(x_1)\ \mathtt{in} \\
& & \mathtt{let}_{\mathtt{mess}}\ x_n = \mathsf{proj}_2(x_1)\ \mathtt{in} \\
& & \mathtt{let}_{\mathtt{mess}}\ x_H^0 = \mathsf{prf}(\langle n_P, x_n\rangle)\ \mathtt{in} \\
& & \mathtt{let}_{\mathtt{mess}}\ x_H^1 = n_P \oplus x_m \oplus x_H^0\ \mathtt{in} \\
& & \mathtt{in}(x_2). \\
& & \mathtt{out}(\mathsf{answer}(x_2, x_H^0, x_H^1)). \\
& & \mathtt{out}(\mathsf{prf}(\langle n_P, x_n, x_m, x_2, \mathsf{answer}(x_2, x_H^0, x_H^1)\rangle))). \\
& & 0
\end{array}
$$

On the other hand, the verifier role must take time into account to model the rapid phase. It uses specific features of our model: the output and the input delimiting the rapid phase are annotated with time variables z_1 and z_2. Moreover, a timing constraint is used to check whether the time elapsed between these two actions is less than the admissible threshold $2 \times t_0$. If not, the prover is assumed remote and the verifier stops.

$$
\begin{array}{rcl}
\mathtt{V}(\mathsf{x}_0, \mathsf{x}_1) & := & \mathtt{in}(x_1). \\
& & \mathtt{let}_{\mathtt{mess}}\ y_1 = \mathsf{adec}(x_1, \mathsf{sk}(\mathsf{x}_0))\ \mathtt{in} \\
& & \mathtt{let}_{\mathtt{mess}}\ y^1_{\mathsf{check}} = \mathsf{eq}(\mathsf{proj}_1(y_1), \mathsf{check}(\mathsf{proj}_2(y_1), \mathsf{spk}(\mathsf{x}_1)))\ \mathtt{in} \\
& & \mathtt{new}\ mV.\ \mathtt{new}\ n_V. \\
& & \mathtt{out}(\langle m_V, n_V\rangle). \\
& & \mathtt{new}\ c. \\
& & \mathtt{out}^{z_1}(c). \\
& & \mathtt{in}^{z_2}(x_2). \\
& & \mathtt{if}_{\mathtt{time}}\ z_2 - z_1 < 2 \times t_0\ \mathtt{then} \\
& & \mathtt{in}(x_3). \\
& & \mathtt{let}_{\mathtt{mess}}\ x_H^0 = \mathsf{prf}(\langle \mathsf{proj}_1(y_1), n_V\rangle)\ \mathtt{in} \\
& & \mathtt{let}_{\mathtt{mess}}\ x_H^1 = \mathsf{proj}_1(y_1) \oplus m_V \oplus x_H^0\ \mathtt{in} \\
& & \mathtt{let}_{\mathtt{mess}}\ y^2_{\mathsf{check}} = \mathsf{eq}(x_2, \mathsf{answer}(c, x_H^0, x_H^1))\ \mathtt{in} \\
& & \mathtt{let}_{\mathtt{mess}}\ y^3_{\mathsf{check}} = \mathsf{eq}(x_3, \mathsf{prf}(\langle \mathsf{proj}_1(y_1), n_V, m_V, c, x_2\rangle))\ \mathtt{in} \\
& & 0
\end{array}
$$

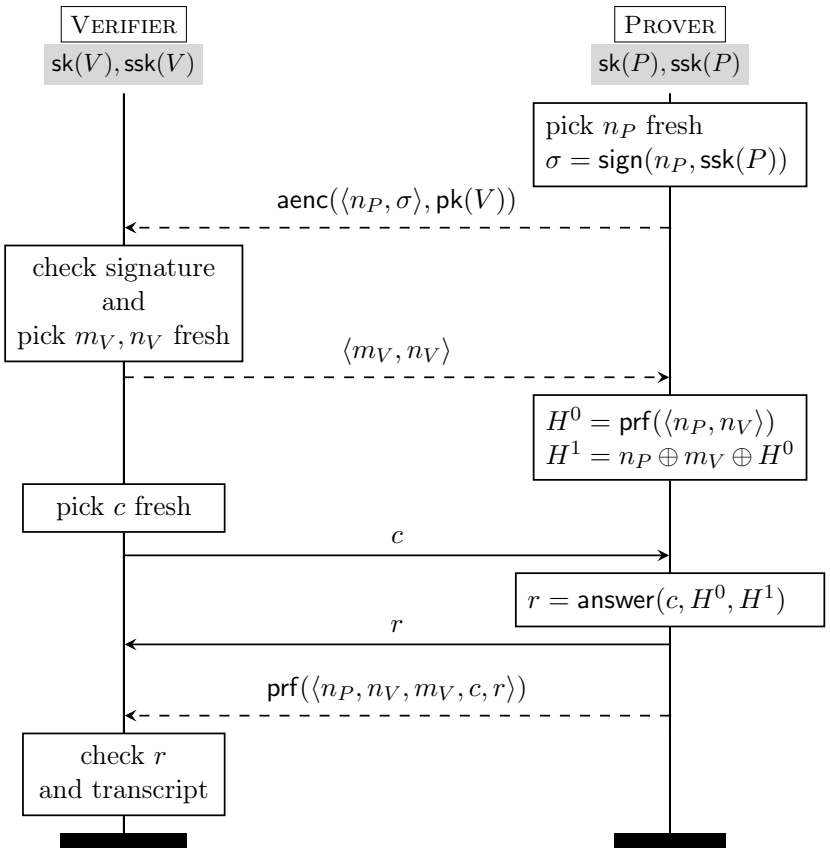


Figure 2.1: Abstracted version of the SPADE protocol.

2.2.2. Semantics

We provide an operational semantics that enables to simulate executions of the protocol. It is defined by a labelled transition system over configurations that is parametrised by a topology. This last models that agents are located in the space and messages take time to travel from one to another.

Definition 2.1. *A topology is a tuple* $\mathcal{T}_0 = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}_0, v_0, p_0)$ *such that:*

- $\mathcal{A}_0 \subseteq \mathcal{A}$ *is the finite set of agents composing the system;*
- $\mathcal{M}_0 \subseteq \mathcal{A}_0$ *is the subset of agents that are malicious;*
- $\mathsf{Loc}_0 : \mathcal{A}_0 \to \mathbb{R}^3$ *is a mapping defining the position of each agent in space;*
- $v_0, p_0 \in \mathcal{A}_0$ *are two agent names that represent respectively the verifier and the prover w.r.t. whom the security analysis is performed.*

For sake of simplicity, we assume that all the messages travel at the same speed, e.g. the speed-of-light, denoted c. The distance between two agents is then defined by the

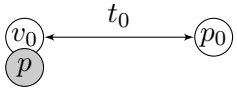
time a message takes to travel from one to the other, i.e.:

$$\mathsf{Dist}_{\mathcal{T}_0}(a,b) = \frac{\|\mathsf{Loc}_0(a) - \mathsf{Loc}_0(b)\|}{\mathsf{c}} \quad \text{for all } a, b \in \mathcal{A}_0$$

where $\|\cdot\|$ denotes the usual Euclidean norm.

Remark 2.5. *Unlike the classical Dolev-Yao attacker [DY83], our model does not assume a unique omniscient and omnipresent attacker. All the agents, including dishonest ones, are subject to the physical constraints which means that they cannot instantaneously receive messages previously sent over the network. They can neither forge messages that can be instantaneously received by the other agents. Enough time must have elapsed to let the message reach its destination. Considering several dishonest agents at different locations is therefore meaningful in our model.*

Example 2.6. *Given a threshold $t_0 \in \mathbb{R}_+$, we define $\mathcal{T}_{\mathsf{s}}^{t_0}$ (depicted on the right) the simple topology composed of three agents $\{v_0, p_0, p\}$ among whom only p is malicious. The relative distance between agents being of greater importance than their precise location, we prefer to describe the topologies focusing on this point. The topology $\mathcal{T}_{\mathsf{s}}^{t_0}$ is thus defined by:*



$$\mathit{Dist}_{\mathcal{T}_{\mathsf{s}}^{t_0}}(v_0, p) = 0 \text{ and } \mathit{Dist}_{\mathcal{T}_{\mathsf{s}}^{t_0}}(v_0, p_0) = t_0.$$

The two agents v_0 and p share the same location, and can thus instantaneously exchange messages, while p_0 is at distance t_0 from them, i.e. a message takes at least a time t_0 to reach p_0 from v_0 or p (and conversely).

The configurations manipulated by our operational semantics aim at describing the network at a given point of an execution. The actions that remain to be executed by the agents belong to a multiset of extended processes of the form $\lfloor P \rfloor_a$, meaning that process P remains to be executed by agent a. The messages already sent over the network are stored into a frame, extended to keep track of when a message has been sent and by whom. Finally, a configuration contains the global time of the system. Formally, given a topology $\mathcal{T}_0 = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}_0, v_0, p_0)$, a configuration over $\mathcal{T}_0$ is as follows.

Definition 2.2. *A configuration is a tuple $(\mathcal{P}; \Phi; t)$ such that:*

- *$\mathcal{P}$ is a multiset of extended processes $\lfloor P \rfloor_a$ such that $a \in \mathcal{A}_0$;*
- *$\Phi = \{\mathsf{w}_1 \xrightarrow{a_1,t_1} u_1, \ldots, \mathsf{w}_n \xrightarrow{a_n,t_n} u_n\}$ is an extended frame with $\mathsf{w}_i \in \mathcal{W}$, $a_i \in \mathcal{A}_0$, $t_i \in \mathbb{R}_+$ and u_i is a message, for $i \in \{1, \ldots n\}$;*
- *$t \in \mathbb{R}_+$ is the global time of the system.*

An initial frame is a frame such that $t_i = 0$ $(1 \leq i \leq n)$. We write $\lfloor \Phi \rfloor_a^t$ for the restriction of Φ to the agent a at time t, i.e.:

$$\lfloor \Phi \rfloor_a^t = \{\mathsf{w}_i \xrightarrow{a_i,t_i} u_i \mid (\mathsf{w}_i \xrightarrow{a_i,t_i} u_i) \in \Phi \text{ and } a_i = a \text{ and } t_i \leq t\}.$$

Example 2.7. *Let us consider the topology $\mathcal{T}_{\mathsf{s}}^{t_0}$ defined in Example 2.6. Given a DB protocol $(\mathtt{V}(\mathsf{x}_0,\mathsf{x}_1),\mathtt{P}(\mathsf{x}_0,\mathsf{x}_1))$ we can define the configuration $\mathcal{K}_0 = (\mathcal{P}_0;\Phi_0;0)$ such that:*

- $\mathcal{P}_0 = \{\lfloor \mathtt{V}_{\mathsf{end}}(v_0,p_0)\rfloor_{v_0} \uplus \lfloor \mathtt{P}(p_0,v_0)\rfloor_{p_0}\}$ *where* $\mathtt{V}_{\mathsf{end}}(v_0,p_0)$ *corresponds to the process* $\mathtt{V}(v_0,p_0)$ *in which the null process* 0 *has been replaced by the special command* $\mathsf{end}(v_0,p_0)$*; and*
- $\Phi_0 = \{\mathsf{w}_1 \xrightarrow{p,0} v_0,\ \mathsf{w}_2 \xrightarrow{p,0} p_0,\ \mathsf{w}_3 \xrightarrow{p,0} p,\ \mathsf{w}_4 \xrightarrow{p,0} \mathsf{sk}(p),\ \mathsf{w}_5 \xrightarrow{p,0} \mathsf{ssk}(p)\}$.

This configuration models a scenario in which the agents v_0 and p_0 try to execute a session of the protocol together. The agent v_0 acts as a verifier while p_0 acts as a prover. The special command $\mathsf{end}(v_0,p_0)$ *has been added in the verifier role to emphasise the end of the session in case of success. Moreover, since the agent p is malicious the frame Φ_0 contains his initial knowledge, i.e. the identities of the agents involved in the configuration as well as his private keys.*

Given a topology $\mathcal{T}_0 = (\mathcal{A}_0,\mathcal{M}_0,\mathsf{Loc}_0,v_0,p_0)$, the operational semantics is given through the transition system presented in Figure 2.2. All the rules are as expected and formally define the informal behaviour of each command presented in Section 2.2.1. The IN rule is the most complex one and deserves some comments. It allows an agent a to receive a message u that has necessarily been forged by another agent b early enough, i.e. at a time $t_b \leq t - \mathsf{Dist}_{\mathcal{T}_0}(b,a)$. Note that, as usual in the applied pi-calculus, dishonest agents are able to forge new messages using messages they already know. However, since they are subject to the physical constraints in our model, they do not know the whole frame: a message outputted by an agent c is known by b at time t_b if, and only if, it belongs to $\lfloor\Phi\rfloor_c^{t_b - \mathsf{Dist}_{\mathcal{T}_0}(c,b)}$.

We sometimes simply write $\rightarrow_{\mathcal{T}_0}$ instead of $\xrightarrow{a,\alpha}_{\mathcal{T}_0}$. The relation $\rightarrow_{\mathcal{T}_0}^*$ is the reflexive and transitive closure of $\rightarrow_{\mathcal{T}_0}$, and we often write $\xrightarrow{\mathsf{tr}}_{\mathcal{T}_0}$ to emphasise the sequence of labels tr that has been used during this execution.

Example 2.8. *Let us continue Example 2.7. However, instead of considering $\mathcal{K}_0$ as defined, we define $\mathcal{K}_0'$ a slightly different configuration: the process $\mathtt{P}(p_0,v_0)$ is replaced by $\mathtt{P}(p_0,p)$. This models a scenario in which the prover p_0 tries to execute a session with the malicious agent p acting as a verifier. This configuration will allow us to illustrate the mafia fraud that applies on the SPADE protocol.*

Let's start the execution as follows:

$$\mathcal{K}_0' \xrightarrow{p_0,\tau}_{\mathcal{T}_{\mathsf{s}}^{t_0}} \xrightarrow{p_0,\mathtt{out}(\mathsf{aenc}(\langle n_P',\mathsf{sign}(n_P',\mathsf{ssk}(p_0))\rangle,\mathsf{pk}(p)))}_{\mathcal{T}_{\mathsf{s}}^{t_0}} \xrightarrow{v_0,\mathtt{in}(\mathsf{aenc}(\langle n_P',\mathsf{sign}(n_P',\mathsf{ssk}(p_0))\rangle,\mathsf{pk}(v_0)))}_{\mathcal{T}_{\mathsf{s}}^{t_0}} \mathcal{K}_1$$

where the configuration $\mathcal{K}_1$ is defined by:

$$\mathcal{K}_1 = (\lfloor V_1\rfloor_{v_0} \uplus \lfloor P_1\rfloor_{p_0};\Phi_0 \cup \{\mathsf{w}_6 \xrightarrow{p_0,0} \mathsf{aenc}(\langle n_P',\mathsf{sign}(n_P',\mathsf{ssk}(p_0))\rangle,\mathsf{pk}(p))\};t_0)$$

$$\text{TIM} \quad (\mathcal{P};\Phi;t) \longrightarrow_{\mathcal{T}_0} (\mathcal{P};\Phi;t+\delta) \qquad \text{with } \delta \geq 0$$

$$\text{NEW} \quad (\lfloor \mathtt{new}\ n.P \rfloor_a \uplus \mathcal{P};\Phi;t) \xrightarrow{a,\tau}_{\mathcal{T}_0} (\lfloor P\{n \mapsto n'\} \rfloor_a \uplus \mathcal{P};\Phi;t) \quad \text{with } n' \in \mathcal{N} \text{ fresh}$$

$$\text{LET\_M} \quad (\lfloor \mathtt{let}_{\mathtt{mess}}\ x = v\ \mathtt{in}\ P \rfloor_a \uplus \mathcal{P};\Phi;t) \xrightarrow{a,\tau}_{\mathcal{T}_0} (\lfloor P\{x \to v{\downarrow}\} \rfloor_a \uplus \mathcal{P};\Phi;t) \quad \text{when } v{\downarrow} \neq \bot$$

$$\text{LET\_T} \quad (\lfloor \mathtt{let}_{\mathtt{time}}\ z = v\ \mathtt{in}\ P \rfloor_a \uplus \mathcal{P};\Phi;t) \xrightarrow{a,\tau}_{\mathcal{T}_0} (\lfloor P\{z \to v{\downarrow}\} \rfloor_a \uplus \mathcal{P};\Phi;t) \quad \text{when } v{\downarrow} \in \mathbb{R}_+$$

$$\text{IF} \quad (\lfloor \mathtt{if}_{\mathtt{time}}\ t_1 \sim t_2\ \mathtt{then}\ P \rfloor_a \uplus \mathcal{P};\Phi;t) \xrightarrow{a,\tau}_{\mathcal{T}_0} (\lfloor P \rfloor_a \uplus \mathcal{P};\Phi;t) \quad \text{if } t_1 \sim t_2 \text{ is true}$$

$$\text{OUT} \quad (\lfloor \mathtt{out}^z(u).P \rfloor_a \uplus \mathcal{P};\Phi;t) \xrightarrow{a,\mathtt{out}(u)}_{\mathcal{T}_0} (\lfloor P' \rfloor_a \uplus \mathcal{P};\Phi \uplus \{\mathsf{w} \xrightarrow{a,t} u\};t) \quad \text{with } \mathsf{w} \in \mathcal{W} \text{ fresh and } P' = P\{z \to t\}$$

$$\text{IN} \quad (\lfloor \mathtt{in}^z(x).P \rfloor_a \uplus \mathcal{P};\Phi;t) \xrightarrow{a,\mathtt{in}(u)}_{\mathcal{T}_0} (\lfloor P' \rfloor_a \uplus \mathcal{P};\Phi;t) \quad \text{with } P' = P\{z \to t\}\{x \to u\}$$

when $u \neq \bot$ and there exist $b \in \mathcal{A}_0$ and $t_b \in \mathbb{R}$ such that $t_b \leq t - \mathsf{Dist}_{\mathcal{T}_0}(b,a)$ and:

- if $b \in \mathcal{A}_0 \setminus \mathcal{M}_0$ then $u \in img(\lfloor \Phi \rfloor_b^{t_b}) \cup \mathbb{R}_+$;
- if $b \in \mathcal{M}_0$ then $u = R\Phi{\downarrow}$ for some recipe R such that for all $\mathsf{w} \in vars(R)$ there exists $c \in \mathcal{A}_0$ such that $\mathsf{w} \in dom(\lfloor \Phi \rfloor_c^{t_b - \mathsf{Dist}_{\mathcal{T}_0}(c,b)})$.

Figure 2.2: Semantics of our calculus

with V_1 the process $\mathtt{V}_{\mathsf{end}}(v_0,p_0)$ without the first input and in which x_1 has been instantiated by $u = \mathsf{aenc}(\langle n'_P, \mathsf{sign}(n'_P, \mathsf{ssk}(p_0))\rangle, \mathsf{pk}(p))$, and P_1 the process $\mathtt{P}(p_0,v_0)$ without the two first commands (i.e. the nonce generation and the output). In the execution above, the first transition is the nonce n_P generation executed by p_0 (note that this commands replaces n_P by a fresh name n'_P in the remaining process). After p_0's output, the next transition is a TIM rule in order to let enough time for the message to reach the attacker p. Then, the final transition models the input, by v_0, of a message constructed by p based on the message sent by p_0. The recipe used to build the message is $R = \mathsf{aenc}(\mathsf{adec}(\mathsf{w}_6,\mathsf{w}_4),\mathsf{pk}(\mathsf{w}_1))$.

2.3. Security properties

Distance-bounding protocols are supposed to resist against mafia frauds, distance hijacking attacks and terrorist frauds. As explained in Chapter 1, they consist in scenarios in which an illegitimate prover, or attacker, tries to be authenticated by a verifier. The three security properties (one for each class of attacks) can therefore be formally expressed

as a reachability property relying on the symbol of function end presented in Example 2.1: does there exist a topology and an execution that reaches the command $\mathsf{end}(v_0, p_0)$ for an illegitimate prover p_0?

2

2.3.1. Valid initial configuration

Depending on the protocol and the class of attacks under study, the set of initial configurations that must be considered varies. Informally, a *valid initial configuration* starts at time $t = 0$ and must contain a multiset of processes made of instances of the roles of the protocols and an initial frame that contains (at least) the initial knowledge of the dishonest agents. Moreover, we assume that there is a unique instance of the verifier role that contains the special command $\mathsf{end}(\mathsf{x}_0, \mathsf{x}_1)$ used to encode the security properties.

For sake of simplicity, we assume that the initial knowledge of an agent is uniform and given through a template, i.e. a set of terms in $\mathcal{T}(\Sigma_c^+, \mathcal{X}_\mathcal{A})$. Formally, given a template $\mathcal{I}_0$ and a set of agent names $\mathcal{A}_0$, the initial knowledge of an agent $a \in \mathcal{A}$ is defined by:

$$\mathsf{Knows}(\mathcal{I}_0, a, \mathcal{A}_0) = \left\{ (u_0\{\mathsf{x}_0 \mapsto a\})\sigma \text{ ground} \;\middle|\; \begin{array}{l} u_0 \in \mathcal{I}_0 \text{ and } \sigma \text{ a substitution} \\ \text{such that } img(\sigma) \subseteq \mathcal{A}_0 \end{array} \right\}.$$

Given a topology $\mathcal{T}_0 = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}_0, v_0, p_0)$ we note $\Phi_{\mathcal{I}_0}^{\mathcal{T}_0}$ the initial frame corresponding to the topology, i.e. such that $\lfloor img(\Phi_{\mathcal{I}_0}^{\mathcal{T}_0}) \rfloor_a^0 = \mathsf{Knows}(\mathcal{I}_0, a, \mathcal{A}_0)$ when $a \in \mathcal{M}_0$, and $\lfloor img(\Phi_{\mathcal{I}_0}^{\mathcal{T}_0}) \rfloor_a^0 = \emptyset$ otherwise.

Example 2.9. *When looking at the the SPADE protocol presented in Figure 2.1, the initial knowledge of a dishonest agent a should at least contain his private keys, i.e.,* $\mathsf{sk}(a)$ *and* $\mathsf{ssk}(a)$*. Another information that a dishonest agent should know is the names of the agents that are involved in the topology under study. In case of* $\mathcal{T}_\mathsf{s}^{t_0}$ *presented in Example 2.6 and then involved in Example 2.8, the dishonest agents must know the names* v_0*,* p_0 *and* p*.*

All this knowledge we informally described may be given to malicious agent through the template $\mathcal{I}_0 = \{\mathsf{x}_1, \mathsf{sk}(\mathsf{x}_0), \mathsf{ssk}(\mathsf{x}_0)\}$*. Indeed,* x_1 *subsumes all the names of the agents that belongs to the topology and* $\mathsf{sk}(\mathsf{x}_0)$ *and* $\mathsf{ssk}(\mathsf{x}_0)$ *provide the private keys of the malicious agent.*

We are now able to define the set of *valid initial configurations* that will be considered when looking for an attack. First, we assume that the multiset of extended processes only contains instances of the roles of the protocols. Then, we assume that the initial frame contains at least the initial knowledge of the dishonest agents that appear in the topology, i.e., $\Phi_{\mathcal{I}_0}^{\mathcal{T}}$. Finally, we assume that an execution starts with a global time set to 0. Such a configuration is said a valid initial configuration for a given protocol $(\mathtt{V}(\mathsf{x}_0, \mathsf{x}_1), \mathtt{P}(\mathsf{x}_0, \mathsf{x}_1))$ w.r.t. a topology $\mathcal{T}$, an initial frame Φ_0, and a template $\mathcal{I}_0$. When clear from the context we will omit the template $\mathcal{I}_0$.

Definition 2.3. *Let* $(\mathtt{V}, (\mathsf{x}_0, \mathsf{x}_1), \mathtt{P}(\mathsf{x}_0, \mathsf{x}_1)$ *be a DB-protocol,* $\mathcal{T}_0 = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}_0, v_0, p_0)$ *be a topology,* Φ_0 *be an initial frame, and* $\mathcal{I}_0$ *be a template. The tuple* $\mathcal{K} = (\mathcal{P}; \Phi; t)$ *is a* valid initial configuration *for* $(\mathtt{V}(\mathsf{x}_0, \mathsf{x}_1), \mathtt{P}(\mathsf{x}_0, \mathsf{x}_1))$ *w.r.t.* $\mathcal{T}_0$*,* Φ_0*, and* $\mathcal{I}_0$ *if:*

- $\mathcal{P} = \lfloor \mathtt{V}_{\mathsf{end}}(v_0, p_0) \rfloor_{v_0} \uplus \mathcal{P}'$, *and for each* $\lfloor P' \rfloor_{a'} \in \mathcal{P}'$, *we have that* $a' \in \mathcal{A}_0$, *and either* $P' = \mathtt{V}(a', b')$ *or* $P' = \mathtt{P}(a', b')$ *for some* $b' \in \mathcal{A}_0$; *and*
- $\Phi = \Phi_0 \cup \Phi_{\mathcal{I}_0}^{\mathcal{T}_0}$; *and*
- $t = 0$.

2.3.2. Mafia fraud

A mafia fraud consists in a scenario where an attacker tries to convince a verifier that an honest prover is close to him even if he is actually far away. The set of topologies representing such scenarios is denoted by $\mathcal{C}_{\mathsf{MF}}^{t_0}$ and contains any topology $\mathcal{T} = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}_0, v_0, p_0)$ such that $v_0, p_0 \in \mathcal{A}_0 \setminus \mathcal{M}_0$ and $\mathsf{Dist}_{\mathcal{T}}(v_0, p_0) \geq t_0$ where t_0 is the t-proximity threshold.

Definition 2.4. *Let* $\mathcal{I}_0$ *be a template and* $(\mathtt{V}, \mathtt{P})$ *be a DB protocol. We say that* $(\mathtt{V}, \mathtt{P})$ admits a mafia fraud *w.r.t.* t_0*-proximity if there exist* $\mathcal{T} \in \mathcal{C}_{\mathsf{MF}}^{t_0}$, *and a valid initial configuration* $\mathcal{K}_0$ *for* $(\mathtt{V}, \mathtt{P})$ *w.r.t.* $\mathcal{T}$ *and* $\emptyset$ *such that:*

$$\mathcal{K}_0 \rightarrow_{\mathcal{T}}^{*} (\lfloor \mathsf{end}(v_0, p_0) \rfloor_{v_0} \uplus \mathcal{P}; \Phi; t)$$

with $\mathcal{T} = (\mathcal{A}_0, \mathcal{M}_0, \mathit{Loc}_0, v_0, p_0)$.

Example 2.10. *We pursue our running example and the execution presented in Example 2.8. First, we can note that* $\mathcal{T}_{\mathsf{s}}^{t_0} \in \mathcal{C}_{\mathsf{MF}}^{t_0}$ *and* $\mathcal{K}_0'$ *is a valid initial configuration for* $(\mathtt{V}(\mathsf{x}_0, \mathsf{x}_1), \mathtt{P}(\mathsf{x}_0, \mathsf{x}_1))$ *w.r.t.* $\mathcal{T}_{\mathsf{s}}^{t_0}$ *and* $\emptyset$ *considering the template* $\mathcal{I}_0 = \{\mathsf{x}_1, \mathsf{sk}(\mathsf{x}_0), \mathsf{ssk}(\mathsf{x}_0)\}$ *presented in Example 2.9. Then we can continue the execution presented in Example 2.8 in order to reach the configuration:*

$$\mathcal{K} = (\lfloor \mathsf{end}(v_0, p_0) \rfloor_{v_0} \uplus \lfloor P_1 \rfloor_{p_0}; \Phi; t_0).$$

The witness trace of a mafia fraud against the SPADE protocol is as follows:

$$\begin{aligned}
\mathcal{K}_0 \xrightarrow{p_0,\tau}_{\mathcal{T}_{\mathsf{s}}^{t_0}} & \xrightarrow{p_0,\mathtt{out}(\mathsf{aenc}(\langle n_P', \mathsf{sign}(n_P', \mathsf{ssk}(p_0))\rangle, \mathsf{pk}(p)))}_{\mathcal{T}_{\mathsf{s}}^{t_0}} \\
& \xrightarrow{v_0,\mathtt{in}(\mathsf{aenc}(\langle n_P', \mathsf{sign}(n_P', \mathsf{ssk}(p_0))\rangle, \mathsf{pk}(v_0)))}_{\mathcal{T}_{\mathsf{s}}^{t_0}} \mathcal{K}_1 \\
\xrightarrow{v_0,\tau}_{\mathcal{T}_{\mathsf{s}}^{t_0}} & \xrightarrow{v_0,\tau}_{\mathcal{T}_{\mathsf{s}}^{t_0}} \xrightarrow{v_0,\tau}_{\mathcal{T}_{\mathsf{s}}^{t_0}} \xrightarrow{v_0,\tau}_{\mathcal{T}_{\mathsf{s}}^{t_0}} \xrightarrow{v_0,\mathtt{out}(\langle m_V', n_V'\rangle)}_{\mathcal{T}_{\mathsf{s}}^{t_0}} \xrightarrow{v_0,\tau}_{\mathcal{T}_{\mathsf{s}}^{t_0}} \\
\xrightarrow{v_0,\mathtt{out}(c')}_{\mathcal{T}_{\mathsf{s}}^{t_0}} & \xrightarrow{v_0,\mathtt{in}(\mathsf{answer}(c', \mathsf{prf}(\langle n_P', n_V'\rangle), n_P' \oplus m_V' \oplus \mathsf{prf}(\langle n_P, n_V'\rangle)))}_{\mathcal{T}_{\mathsf{s}}^{t_0}} \xrightarrow{v_0,\tau}_{\mathcal{T}_{\mathsf{s}}^{t_0}} \\
& \xrightarrow{v_0,\mathtt{in}(\mathsf{prf}(\langle n_P', n_V', m_V', c', \mathsf{answer}(c', \mathsf{prf}(\langle n_P', n_V'\rangle), n_P \oplus m_V' \oplus \mathsf{prf}(\langle n_P', n_V'\rangle))))))}_{\mathcal{T}_{\mathsf{s}}^{t_0}} \\
\xrightarrow{v_0,\tau}_{\mathcal{T}_{\mathsf{s}}^{t_0}} & \xrightarrow{v_0,\tau}_{\mathcal{T}_{\mathsf{s}}^{t_0}} \xrightarrow{v_0,\tau}_{\mathcal{T}_{\mathsf{s}}^{t_0}} \xrightarrow{v_0,\tau}_{\mathcal{T}_{\mathsf{s}}^{t_0}} \mathcal{K} = (\lfloor \mathsf{end}(v_0, p_0) \rfloor_{v_0} \uplus \mathcal{P}; \Phi; t_0)
\end{aligned}$$

Except the first output (done by p_0*), all the actions are performed by either the verifier* v_0 *or the attacker* p *impersonating the prover* p_0. *However only those performed by the verifier are visible in the trace. The two first rows have been presented in Example 2.8. Let us focus on the following. The third row starts by two* τ *actions corresponding to the tests*

performed by the verifier on the previously received message (i.e. decryption and signature check). The three following actions are the name generation corresponding to m_V and n_V and the output of the pair. Finally, third row finishes by the name generation of a challenge c'. The fourth row is the rapid phase: it starts by the output of this challenge cs, followed by the input of the response and the time check. When executing the input, the current frame is

$$\Phi = \Phi_0 \cup \{\mathsf{w}_6 \xrightarrow{p_0,0} \mathsf{aenc}(\langle n'_P, \mathsf{sign}(n'_P, \mathsf{ssk}(p_0))\rangle, \mathsf{pk}(p)), \mathsf{w}_7 \xrightarrow{v_0,t_0} \langle m'_V, n'_V\rangle,\ \mathsf{w}_8 \xrightarrow{v_0,t_0} c'\ \}$$

and remains unchanged until the end of the execution. The response is forged by the attacker using the recipe:

$$R_2 = \mathsf{answer}(\mathsf{w}_8, \mathsf{prf}(R', \mathsf{proj}_2(\mathsf{w}_7)), R' \oplus \mathsf{proj}_1(\mathsf{w}_7) \oplus \mathsf{prf}(R', \mathsf{proj}_2(\mathsf{w}_7))),$$

with $R' = \mathsf{proj}_1(\mathsf{adec}(\mathsf{w}_6, \mathsf{w}_4))$. On the fifth row, the input is also forged by the malicious agent p using the recipe:

$$R_3 = \mathsf{prf}(R', \mathsf{proj}_2(\mathsf{w}_7), \mathsf{proj}_1(\mathsf{w}_7), \mathsf{w}_8, R_2).$$

Finally, the sixth row consists of the final checks performed by the verifier to verify the correctness of the response to the challenge and the transcript of the session. The resulting configuration $\mathcal{K}$ is thus made of the $\mathsf{end}(v_0, p_0)$ process executed by the verifier (plus the remaining process executed by the prover p_0).

2.3.3. Terrorist fraud

In some aspects, a terrorist fraud may be considered as a strong version of a mafia fraud. In this scenario, a remote prover accepts to collude with an attacker located in the vicinity of a verifier to be authenticated once. However, this help must not give an advantage to the attacker to mount future attacks, like impersonations.

Modelling such a scenario is trickier than mafia frauds because the far away prover is neither honest (e.g. he may not follow the protocol rules) nor fully dishonest (he does not accept to reveal his longterm keys which would eventually lead to an impersonation attack). To model these scenarios, first, we will consider all the possible behaviours for this semi-dishonest prover that allow the attacker to authenticate once, considering the simple topology, $\mathcal{T}_{\mathsf{s}}^{t_0}$, described in Example 2.6 . Then, to be terrorist fraud resistant, we have to check that any of these behaviours will allow the attacker to re-authenticate later on. This means that the prover cannot reveal enough information to let the attacker authenticate once without running the risk of being impersonated afterwards.

Definition 2.5. *Let $(\mathtt{V}, \mathtt{P})$ be a DB protocol and $t_0 \in \mathbb{R}_+$ be a threshold. A* semi-dishonest prover *for $(\mathtt{V}, \mathtt{P})$ w.r.t. t_0 is a process $P_{\mathtt{sd}}$ together with an initial frame $\Phi_{\mathtt{sd}}$ such that:*

$$(\{\lfloor \mathtt{V}_{\mathsf{end}}(v_0, p_0)\rfloor_{v_0}\ ;\ \lfloor P_{\mathtt{sd}}\rfloor_{p_0}\}; \emptyset; 0) \to^*_{\mathcal{T}_{\mathsf{s}}^{t_0}} (\{\lfloor \mathsf{end}(v_0, p_0)\rfloor_{v_0}\ ;\ \lfloor 0\rfloor_{p_0}\}; \Phi; t)$$

for some t, and Φ such that Φ and $\Phi_{\mathtt{sd}}$ coincide up to their timestamps.

Example 2.11. *A semi-dishonest prover for the SPADE protocol is as follows:*

$$\begin{aligned}
\mathtt{P} \;:=\; & \mathtt{new}\ n_P.\\
& \mathtt{out}(\mathsf{aenc}(\langle n_P, \mathsf{sign}(n_P, \mathsf{ssk}(p_0))\rangle, \mathsf{pk}(v_0))).\\
& \mathtt{in}(x_1).\\
& \mathtt{let}_{\mathtt{mess}}\ x_m = \mathsf{proj}_1(x_1)\ \mathtt{in}\\
& \mathtt{let}_{\mathtt{mess}}\ x_n = \mathsf{proj}_2(x_1)\ \mathtt{in}\\
& \mathtt{let}_{\mathtt{mess}}\ x_H^0 = \mathsf{prf}(\langle n_P, x_n\rangle)\ \mathtt{in}\\
& \mathtt{let}_{\mathtt{mess}}\ x_H^1 = n_P \oplus x_m \oplus x_H^0\ \mathtt{in}\\
& \mathtt{out}(\langle x_H^0, x_H^1\rangle).\\
& \mathtt{in}(x_2).\\
& \mathtt{out}(\mathsf{answer}(x_2, x_H^0, x_H^1)).\\
& \mathtt{out}(\mathsf{prf}(\langle n_P, x_n, x_m, x_2, \mathsf{answer}(x_2, x_H^0, x_H^1)\rangle)).\\
& 0
\end{aligned}$$

It consists to the prover role in which an output has been introduced just before the rapid phase: the semi-dishonest prover outputs all the material required to forge the response to the challenge, i.e. H^0 and H^1. An initial frame corresponding to this semi-dishonest prover is, up to a renaming of freshly generated names, the following:

$$\begin{aligned}
\Phi_{\mathtt{sd}} = \; & \Big\{ \mathsf{w}_1 \xrightarrow{p_0,0} \mathsf{aenc}(\langle n_P, \mathsf{sign}(n_P, \mathsf{ssk}(p_0))\rangle, \mathsf{pk}(v_0)),\ \mathsf{w}_2 \xrightarrow{v_0,0} \langle m_V, n_V\rangle,\\
& \mathsf{w}_3 \xrightarrow{p_0,0} \langle H^0, H^1\rangle,\ \mathsf{w}_4 \xrightarrow{v_0,0} c,\ \mathsf{w}_5 \xrightarrow{p_0,0} \mathsf{answer}(c, H^0, H^1),\\
& \mathsf{w}_6 \xrightarrow{p_0,0} \mathsf{prf}(\langle n_P, n_V, m_V, c, \mathsf{answer}(c, H^0, H^1)\rangle)\Big\}
\end{aligned}$$

with $H^0 = \mathsf{prf}(\langle n_P, n_V\rangle)$ and $H^1 = n_P \oplus m_V \oplus H^0$.

We are now able to define our notion of terrorist fraud resistance. Intuitively, if the dishonest prover gives to his accomplice enough information to pass authentication once, then the latter will be able to authenticate again without any prover's help.

Definition 2.6. *Let $\mathcal{I}_0$ be a template, $(\mathtt{V}, \mathtt{P})$ be a DB-protocol and $t_0 \in \mathbb{R}_+$ be a threshold. We say that $(\mathtt{V}, \mathtt{P})$ is* terrorist fraud resistant *w.r.t. t_0-proximity if for all semi-dishonest prover $P_{\mathtt{sd}}$ w.r.t. t_0 with frame $\Phi_{\mathtt{sd}}$, there exist a topology $\mathcal{T} \in \mathcal{C}_{\mathsf{MF}}^{t_0}$ and a valid initial configuration $\mathcal{K}$ for $(\mathtt{V}, \mathtt{P})$ w.r.t. $\mathcal{T}$ and $\Phi_{\mathtt{sd}}$ such that:*

$$\mathcal{K} \to_{\mathcal{T}}^{*} (\, \lfloor \mathsf{end}(v_0, p_0) \rfloor_{v_0} \uplus \mathcal{P}; \Phi; t)$$

where $\mathcal{T} = (\mathcal{A}_0, \mathcal{M}_0, \mathit{Loc}_0, v_0, p_0)$.

We can note that unlike standard reachability properties, exhibiting a trace of execution is not sufficient to prove the existence of a terrorist fraud. Indeed, it requires to exhibit a semi-dishonest prover and check that for all the possible executions, none of them leads to a re-authentication. Unfortunately, exhibiting a trace is not sufficient to prove terrorist fraud resistance neither: the re-authentication must be possible for all the semi-dishonest provers. Both directions require a complex analysis that must explore an

infinite set: either the set of traces or the set of semi-dishonest provers.

The unique situation in which terrorist fraud analysis is simple is when the protocol already suffers from a mafia fraud. The following proposition applies.

Proposition 2.1. *Let $\mathcal{I}_0$ be a template and $(\mathtt{V},\mathtt{P})$ be a DB-protocol. If $(\mathtt{V},\mathtt{P})$ admits a mafia fraud then it is terrorist fraud resistant (w.r.t. t_0-proximity).*

Indeed, an execution that witnesses the existence of a mafia fraud can be leveraged as a witness of re-authentication for any semi-dishonest prover. By definition of a mafia fraud, the execution starts with a configuration $\mathcal{K}$ that is a valid initial configuration for $(\mathtt{V},\mathtt{P})$ w.r.t. $\mathcal{T}$ and $\emptyset$. It can be extended as a valid initial configuration for $(\mathtt{V},\mathtt{P})$ w.r.t. $\mathcal{T}$ and $\Phi_{\mathtt{sd}}$ by adding elements in the initial frame. This does not alter the trace since it simply increases the initial knowledge of the attackers.

Remark 2.6. *Regarding the literature, this implication is debatable. Indeed, a contradictory implication is sometimes stated [CdRS18, ABK+11]: if a protocol admits a mafia fraud then it admits a terrorist fraud. We first recall the informal definition of terrorist fraud given in Chapter 1 to understand such a gap:*

> *A terrorist fraud is a scenario in which a far-away dishonest prover accepts to collude with an attacker to be authenticated once by an honest verifier, but without giving any advantage to the attacker for future attacks.*

There are two ways to formally define it:

1. *consider all the dishonest provers that do not give any advantage to the attacker for future (i.e. do not enable a re-authentication); and check whether they can be authenticated once.*

2. *consider all the dishonest provers that can be authenticated once; and check whether they give an advantage to the attacker for future attacks.*

The first definition is appealing since it expresses terrorist fraud as a unique reachability issue: is authentication possible? The implication "mafia fraud implies terrorist fraud" is quite immediate: if there is a mafia fraud then authentication is possible. However, it remains to define the set of "dishonest provers that do not give any advantage to the attacker" and this appears to be a difficult task, especially in symbolic models.

The second definition is more in line with ours. It provides a more complex security property but it does not elude the difficulty of defining the "advantage" by pushing it inside the definition of admissible collusion.

Hopefully, it seems that these two definitions match when analysing protocol that are mafia fraud resistant. Therefore, to bridge the gap we could only define terrorist frauds for protocols that are already mafia fraud resistant. However, for sake of simplicity, we decided to let the two properties independent and simply argue that analysing a protocol w.r.t. terrorist fraud is meaningless when it already suffers from a mafia fraud.

Example 2.12. *As presented in Example 2.10, the SPADE protocol is vulnerable to a mafia fraud. Following the previous remark we can thus immediately conclude that it is terrorist fraud resistant.*

However, in [Ger18], authors proposed a fixed version of the SPADE protocol to prevent mafia frauds which consists in adding the identity of the verifier inside the first signature, i.e. the first message becomes $\mathsf{aenc}(\langle n_P, \sigma\rangle, \mathsf{pk}(v_0))$ *with* $\sigma = \mathsf{sign}(\langle n_P, v_0\rangle, \mathsf{sk}(p_0))$. *We will see in Chapter 5 that this fix effectively avoid mafia frauds and does not alter the terrorist fraud resistance of the protocol, i.e. for all the semi dishonest prover there exists a trace of re-authentication. Let us describe this trace when considering the semi-dishonest prover presented in Example 2.11 (adapted for the fix): the attacker replays the first message sent by* p_0 *and stored through* w_1 *in the frame* $\Phi_{\mathtt{sd}}$. *Then he is able to answer all the messages requested by* v_0 *since he can deduce* n_P *from the frame:*

$$n_P = R\Phi_{\mathtt{sd}}\downarrow \text{ with } R = \mathsf{proj}_1(\mathsf{w}_3) \oplus \mathsf{proj}_2(\mathsf{w}_3) \oplus \mathsf{proj}_1(\mathsf{w}_2).$$

2.3.4. Distance hijacking attacks

In addition to mafia and terrorist frauds, a distance-bounding protocol should also resist against distance fraud and distance hijacking. They consist in scenarios in which a dishonest prover tries to be authenticated by a remove verifier. In a distance fraud, this prover is let alone while he can abuse other honest parties, e.g. located in the vicinity of the verifier, when looking for distance hijacking attacks. Since these two classes of attacks are very similar, we decided to gather them into a single class we will call *distance hijacking attacks*.

More formally, the set of topologies representing such scenarios is noted $\mathcal{C}_{\mathsf{DH}}^{t_0}$ and contains any topology $\mathcal{T} = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}_0, v_0, p_0)$ such that $v_0 \in \mathcal{A}_0 \setminus \mathcal{M}_0$, and $p_0 \in \mathcal{M}_0$, and for every $a \in \mathcal{M}_0$, $\mathsf{Dist}_{\mathcal{T}}(a, v_0) \geq t_0$ where t_0 is the threshold delimiting the vicinity of v_0. This definition encompasses the special case in which nobody is located in the vicinity of the verifier, a.k.a. a distance fraud.

Definition 2.7. *Let* $\mathcal{I}_0$ *be a template,* $(\mathtt{V}, \mathtt{P})$ *be a DB-protocol, and* $t_0 \in \mathbb{R}_+$ *be a threshold. We say that* $(\mathtt{V}, \mathtt{P})$ *admits a* distance hijacking attack *w.r.t.* t_0*-proximity if there exist* $\mathcal{T} \in \mathcal{C}_{\mathsf{DH}}^{t_0}$, *and a valid initial configuration* $\mathcal{K}$ *for* $(\mathtt{V}, \mathtt{P})$ *w.r.t.* $\mathcal{T}$ *and* $\emptyset$ *such that:*

$$\mathcal{K} \to_{\mathcal{T}}^* (\, \lfloor \mathsf{end}(v_0, p_0) \rfloor_{v_0} \uplus \mathcal{P}; \Phi; t)$$

with $\mathcal{T} = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}_0, v_0, p_0)$.

Example 2.13. *The SPADE protocol has been proved secure w.r.t. distance hijacking attacks [BGG+16]. However, the underlying model was preventing the attacker to act as a verifier. By relaxing this constraint, the protocol becomes vulnerable to a distance hijacking attack, presented in Figure 2.3. The attacker acts as verifier and we assume that an honest prover initiates a session with him. He is then able to reuse the nonce* n_P *sent by the prover to initiate a session with the verifier* v_0. *The rest of the protocol will then be executed between the verifier* v_0 *and the honest prover* p *(who thinks he is talking*

with the verifier p_0).

Formally, this attack is caught by our definition considering the simple topology $\mathcal{T} = (\{v_0, p_0, p\}, \{p_0\}, \mathsf{Loc}_0, v_0, p_0)$ *with* $\mathsf{Dist}_{\mathcal{T}}(v_0, p_0) = t_0$ *and* $\mathsf{Dist}_{\mathcal{T}}(v_0, p) = 0$*, and the initial configuration* $\mathcal{K}_0 = (\lfloor \mathtt{V}_{\mathsf{end}}(v_0, p_0) \rfloor_{v_0} \uplus \lfloor \mathtt{P}(p, p_0) \rfloor_p ; \Phi_0; 0)$ *where* $\Phi_0 = \{\mathsf{w}_1 \xrightarrow{p_0,0} v_0,\ \mathsf{w}_2 \xrightarrow{p_0,0} p_0,\ \mathsf{w}_3 \xrightarrow{p_0,0} p,\ \mathsf{w}_4 \xrightarrow{p_0,0} \mathsf{sk}(p_0),\ \mathsf{w}_5 \xrightarrow{p_0,0} \mathsf{ssk}(p_0)\}$*. The trace of the attack is then as follows:*

$$
\begin{aligned}
\mathcal{K}_0 \quad & \xrightarrow{p_0,\tau}_{\mathcal{T}} \xrightarrow{p,\mathtt{out}(\mathsf{aenc}(\langle n_P, \mathsf{sign}(n_P, \mathsf{ssk}(p))\rangle, \mathsf{pk}(p_0)))}_{\mathcal{T}} \\
& \xrightarrow{}_{\mathcal{T}} \xrightarrow{v_0,\mathtt{in}(\mathsf{aenc}(\langle n_P, \mathsf{sign}(n_P, \mathsf{ssk}(p_0))\rangle, \mathsf{pk}(v_0)))}_{\mathcal{T}} \\
& \xrightarrow{v_0,\tau}_{\mathcal{T}} \xrightarrow{v_0,\tau}_{\mathcal{T}} \xrightarrow{v_0,\tau}_{\mathcal{T}} \xrightarrow{v_0,\tau}_{\mathcal{T}} \xrightarrow{v_0,\mathtt{out}(\langle m_V, n_V\rangle)}_{\mathcal{T}} \\
& \xrightarrow{p,\mathtt{in}(\langle m_V, n_V\rangle)}_{\mathcal{T}} \xrightarrow{p,\tau}_{\mathcal{T}} \xrightarrow{p,\tau}_{\mathcal{T}} \xrightarrow{p,\tau}_{\mathcal{T}} \xrightarrow{p,\tau}_{\mathcal{T}} \\
& \xrightarrow{v_0,\tau}_{\mathcal{T}} \xrightarrow{v_0,\mathtt{out}(c)}_{\mathcal{T}} \xrightarrow{p,\mathtt{in}(c)}_{\mathcal{T}} \xrightarrow{p,\mathtt{out}(\mathsf{answer}(c,\mathsf{prf}(\langle n_P, n_V\rangle), n_P \oplus m_V \oplus \mathsf{prf}(\langle n_P, n_V\rangle)))}_{\mathcal{T}} \\
& \xrightarrow{v_0,\mathtt{in}(\mathsf{answer}(c,\mathsf{prf}(\langle n_P, n_V\rangle), n_P \oplus m_V \oplus \mathsf{prf}(\langle n_P, n_V\rangle)))}_{\mathcal{T}} \xrightarrow{v_0,\tau}_{\mathcal{T}} \\
& \xrightarrow{p,\mathtt{out}(\mathsf{prf}(\langle n_P, n_V, m_V, c, \mathsf{answer}(c_V, \mathsf{prf}(\langle n_P, n_V\rangle), n_P \oplus m_V \oplus \mathsf{prf}(\langle n_P, n_V\rangle))))}_{\mathcal{T}} \\
& \xrightarrow{v_0,\mathtt{in}(\mathsf{prf}(\langle n_P, n_V, m_V, c, \mathsf{answer}(c, \mathsf{prf}(\langle n_P, n_V\rangle), n_P \oplus m_V \oplus \mathsf{prf}(\langle n_P, n_V\rangle))))}_{\mathcal{T}} \\
& \xrightarrow{v_0,\tau}_{\mathcal{T}} \xrightarrow{v_0,\tau}_{\mathcal{T}} \xrightarrow{v_0,\tau}_{\mathcal{T}} \xrightarrow{v_0,\tau}_{\mathcal{T}} \mathcal{K}
\end{aligned}
$$

with $\mathcal{K} = (\lfloor \mathsf{end}(v_0, p_0) \rfloor_{v_0} \uplus \lfloor 0 \rfloor_p ; \Phi; 2 \times t_0)$.

The first message received by v_0 has been forged by the attacker p_0 applying the recipe:

$$R = \mathsf{aenc}(\langle R_{n_P}, \mathsf{sign}(R_{n_P}, \mathsf{w}_5)\rangle, \mathsf{pk}(\mathsf{w}_1))$$

where $R_{n_P} = \mathsf{proj}_1(\mathsf{adec}(\mathsf{w}_6, \mathsf{w}_4))$ *is the recipe used to extract the nonce n_P. All the following inputs are filled using the output executed just before. One may note that the TIM rule is executed once with delay* $\delta = 2 \times t_0$ *to let enough time for the first output to travel back and forth from v_0 to p_0. Note that in our model, the computation performed by p does not take any time. The TIM rule is then no longer executed afterwards since v_0 and p are at the same place.*

2.4. Comparison with existing models

We elaborate now on two recent symbolic models, developed for analysing distance-bounding protocols by Chothia *et al.* [CGdR$^+$15, CdRS18] and Mauw *et al.* [MSTPTR18, MSTPTR19]. In particular, we will compare the security properties.

2.4.1. Chothia *et al.*'s model

This model, firstly published in 2015, has been the basis of our work. Hence, our definitions of mafia frauds and distance hijacking attacks are in line with those proposed in [CGdR$^+$15]. The unique difference is about the quantification over the topologies: Chothia *et al.* define the security properties w.r.t. to rather simple topologies made of, at most, 4 agents, without providing any justification. The reduction results, we will

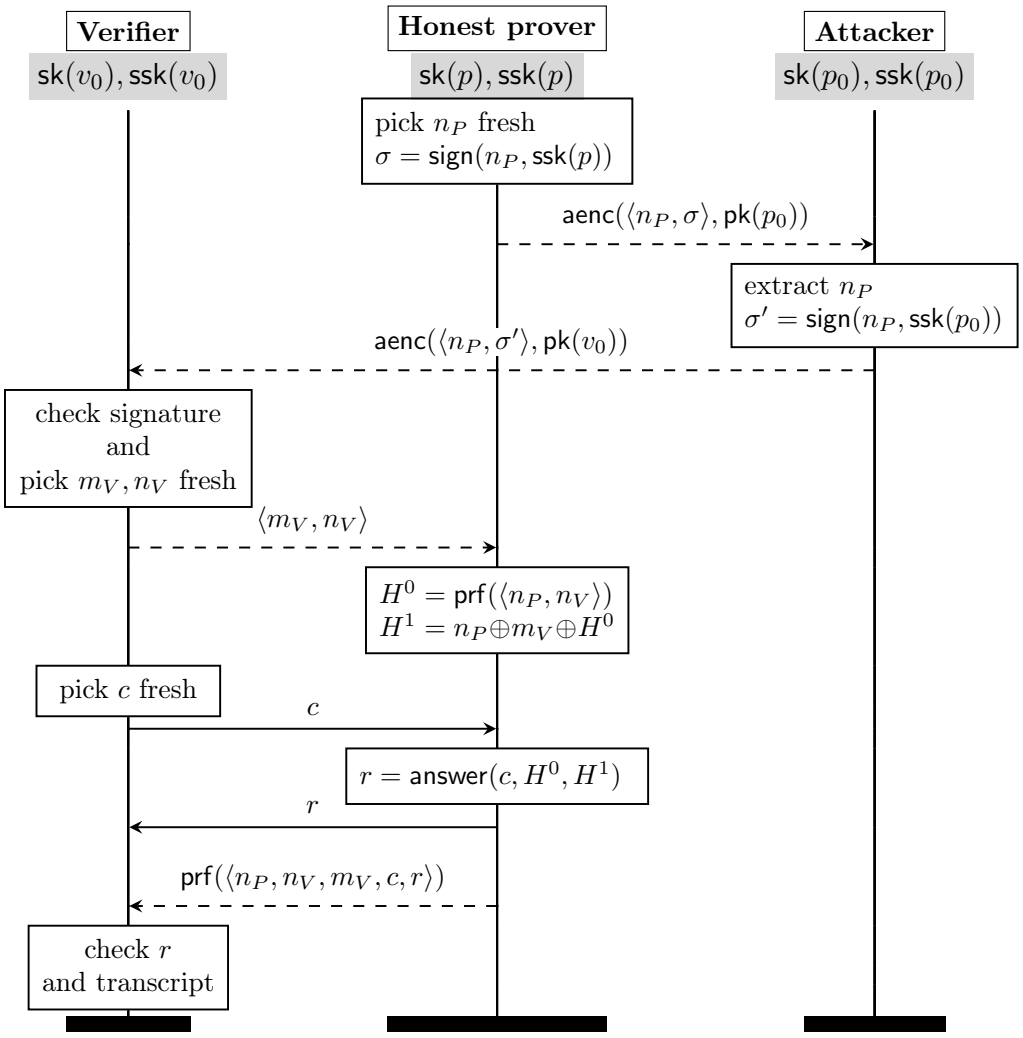


Figure 2.3: Distance hijacking attack against the SPADE protocol

present in Chapter 4, will provide such a justification and also make clear the assumptions that the protocols must satisfy to apply these reduction results.

Now, regarding terrorist frauds, the definition proposed in [CdRS18] differs with ours. Indeed, while we are following the second definition of terrorist fraud mentioned in Remark 2.6, they are following the first one. Remember it consists in considering any oracle process that colludes with dishonest agents as soon as it does not give any advantage for future attacks, and check whether the oracle can be authenticated once.

The security property thus involves what is called a *terrorist prover* which performs any operation on behalf of the attacker. It can for example encrypt, decrypt, sign... any value the attacker wishes, but never revealing its secrets. Even if this notion of terrorist

prover is appealing, because suitable for automation, they do not explain how to write such a process and we do think that it may be a difficult task. In the following we present two examples that illustrate such difficulties.

Example 2.14. *Consider a protocol such that the prover role relies on a secret k and a hash function* $\mathtt{f}$*. Given an input u sent by his accomplice, a legitimate behaviour of the terrorist prover may be to reveal the hash value of the input data together with his secret key, i.e.* $\mathtt{f}(\langle k, u\rangle)$*. Indeed such a message might help his accomplice but without revealing any information about k. Formally, the terrorist prover should contain the oracle:* $\mathtt{P}_1 = \mathtt{in}(x).\mathtt{out}(\mathtt{f}(\langle k, x\rangle))$*. In the same spirit, we could argue that the oracle* $\mathtt{P}_2 = \mathtt{in}(x).\mathtt{out}(\mathtt{f}(\langle x, k\rangle))$ *is also useful, and perhaps also the oracle* $\mathtt{P}_3 = \mathtt{in}(x_1).\mathtt{in}(x_2).\mathtt{out}(\mathtt{f}(\langle x_1, \langle k, x_2\rangle\rangle))$*, etc. Iterating such a reasoning, it is unclear how to write a finite terrorist prover that will provide all the valuable help his accomplice may need.*

Another issue comes up when considering equational theories modelling operators with algebraic properties, like the exclusive-or. It seems difficult (perhaps even impossible) to be sure that the terrorist prover we consider will not reveal secrets (possibly indirectly).

Example 2.15. *Consider an equational theory made of three public symbols of function* $\mathtt{g}$*,* $\mathtt{f}_1$ *and* $\mathtt{f}_2$ *such that* $\mathtt{g}(\mathtt{f}_1(x, y), \mathtt{f}_2(x, y)) = y$*. Following the idea developed in [CdRS18], the terrorist prover should contain the two oracles* $\mathtt{P}_1 = \mathtt{in}(x).\mathtt{out}(\mathtt{f}_1(x, k))$ *and* $\mathtt{P}_2 = \mathtt{in}(x).\mathtt{out}(\mathtt{f}_2(\langle x, k\rangle))$*. Even though these two oracles are individually legitimate, put together an attacker is able to get* $\mathtt{f}_1(m, k)$ *and* $\mathtt{f}_2(m, k)$ *for some message m and thus retrieve the secret key k. This example clearly shows that it is not obvious to describe in a syntactic way the help the terrorist prover is willing to provide (even for rather simple theories).*

2.4.2. Mauw *et al.*'s model

Meanwhile we were developing our framework [DDW18, DDW19], Mauw *et al.* proposed a concurrent model based on a multiset rewriting system.

Regarding terrorist fraud, their security property is completely in line with ours. Indeed, they seem (almost) equivalent, up to a different convention naming (i.e. *valid extension* instead of semi-dishonest prover). They also consider two-step scenarios in which a semi-dishonest prover tries to authenticate once and then check whether it can be re-authenticated later on. However, while we are considering a unique and rather simple topology $\mathcal{T}_{\mathsf{s}}^{t_0}$ for the first authentication, they consider any topology $\mathcal{T} \in \mathcal{C}_{\mathsf{MF}}^{t_0}$. This change is the unique difference between our two definitions of terrorist frauds.

Regarding mafia fraud and distance hijacking, we can note a difference. Indeed, instead of modelling these two classes of attacks separately, they define a unique property that gathers both, named *secure distance-bounding*. It claims that:

> A protocol is *distance-bounding secure* if for all trace of execution tr that contains the event $\mathsf{claim}(V, P, x, y)$ for an honest agent V, then there are two actions (t_x, α_x) and (t_y, α_y) in tr which correspond to x and y and such that $(t_y - t_x) \leq 2 \cdot \mathsf{Dist}(V, P')$ with $P \approx P'$.

The notation $P \approx P'$ allows to replace agent P by any dishonest agent if P is dishonest. Indeed, in this case, it can share all its credentials with an accomplice who can then impersonate it. The verifier can thus only estimate its distance to the closest malicious agent. It is important to note that this security property gathers both mafia frauds and distance-hijacking attacks since it applies for scenarios in which the prover P may be honest or dishonest.

According to the authors, this property can be restrained to focus on mafia fraud only by assuming the prover P honest. Relying on these two properties they are then able to say:

- a protocol admits a mafia fraud if *secure distance-bounding* does not hold when restricted to an honest prover P (no restriction about other participants); and
- a protocol admits a distance hijacking attack if *secure distance-bounding* holds when restricted to an honest prover P but does not otherwise.

Unfortunately, this seems to introduce a hierarchy between mafia fraud and distance hijacking: a protocol shall first resist against mafia fraud, and, if so, shall then resist again distance hijacking. Their framework does not allow to prove the security of protocols that are distance hijacking resistant but suffer from a mafia fraud. Even if we never encountered such protocols in the literature, we do think that they may be interesting in practice if the infrastructure in which the distance-bounding protocol is implemented enforces, by design, that malicious agent cannot enter in the verifier's proximity, e.g. relying on CCTV.

A bounded number of sessions 3

In this chapter, we propose a procedure to verify the three security properties, i.e. mafia fraud, terrorist fraud and distance hijacking, introduced in Chapter 2. As a first approach the procedure will apply in a bounded setting, i.e. considering a bounded number of sessions. It builds on an existing verification procedure originally proposed by Chadha et al. *[CCCK16], and implemented in the verification tool Akiss. In this chapter, we focus on the theoretical development that allows to prove the soundness and the correctness of our new procedure. Its implementation and its application to a number of protocols will be discussed in Chapter 5. In particular, we do not prove its termination (it was already a long and tedious work for the original procedure, more than 20 pages). Instead, we will demonstrate that our procedure terminates in practice through our case studies presented in Chapter 5.*

Contents

3.1. Preliminaries

Our procedure builds on the Akiss procedure and requires an extra assumption on the model, already presented in Chapter 2. In this section, we first give a brief overview of the existing procedure (and implemented in the verification tool Akiss). Then, we formally introduce the new assumption, i.e. the finite variant property. Finally we define the notion of traces on which the procedure is going to apply. Indeed, considering a bounded number of sessions, a protocol can be entirely described by a finite set of traces that represent all the possible interleavings of the processes under study.

3.1.1. Akiss in a nutshell

The procedure we are going to extend in this chapter has been proposed by Chadha *et al.* [CCCK16] and implemented in the verification tool Akiss. It has been originally designed to verify equivalence-based properties on protocols which do not feature time. Thus, this tool takes as input two protocols described through a variant of the applied pi-calculus, and returns whether the equivalence holds or not. Since the procedure tackles the issue of verifying security properties for a bounded number of sessions, some simplifications can be done w.r.t. the standard applied-pi calculus: `new` commands are removed and the procedure is theoretically proved correct considering symbolic traces of executions, instead of processes. Indeed, w.l.o.g. we can assume that all the bound names occurring in the sessions are set all different in advance: therefore they do not need to be freshened during an execution. Moreover, since there is a finite number of sessions, all the possible interleavings of actions can be pre-computed so that the core procedure applies on the finite set of symbolic traces describing the protocol.

Regarding the Akiss procedure, it consists in three steps: first, given a symbolic trace, it computes a finite set of Horn clauses, named *seed statements*, that represents all the possible executions of this trace. Then, it applies a saturation procedure that generates a finite set of canonical Horn clauses, called *solved statements*, that still covers all the possible executions and enjoys some nice properties. Finally, it checks whether the two symbolic traces are equivalent or not by comparing their sets of solved statements. If the last step is useless when verifying reachability properties we are going to build on the two first ones, paying attention that the computed sets of statement remain complete, even with time and locations considerations. The soundness of the procedure will be provided by a new third step that checks all the timing constraints that must be satisfied during an execution.

The procedure we propose is thus composed of three steps:

- generate a finite set of seed statements from the specification of the protocol;
- apply a saturation procedure on the seed statements in order to construct a finite set of solved statements;
- check whether the runs obtained by saturation are executable in our timed semantics by solving the timing constraints.

The two first steps remains the same as the Akiss procedure even if we will have to pay attention that our finite representations of the possible traces remain complete, even under time and locations considerations. The third step is new. Since the two first steps build on the Akiss tool, they do not take care of the executability of the generated witnesses of attacks in our timed semantics (it solely ensure their executability in an untimed one). In the last step of our procedure, we therefore derive the set of timing constraints that each witness of attack must satisfy to be executed, and decide whether a solution exists or not.

3.1.2. Finite variant property

The procedure we are going to define builds on the Akiss tool which makes some assumptions on the cryptographic primitives occurring in the protocols. Indeed, it handles convergent rewriting systems that satisfy the finite variant property [CLD05].

Definition 3.1. *Given a convergent rewriting system represented by a set of rules $\mathcal{R}$, we say that $\mathcal{R}$ satisfies the finite variant property if, given a sequence of terms $t_1, \ldots, t_n$, there exists a finite set of substitutions denoted $\mathsf{variants}^{\mathcal{R}}(t_1, \ldots, t_n)$ such that for any substitution ω, there exist $\sigma \in \mathsf{variants}^{\mathcal{R}}(t_1, \ldots, t_n)$ and τ such that:*

$$t_1\omega\downarrow, \ldots, t_n\omega\downarrow = t_1\sigma\downarrow\tau, \ldots, t_n\sigma\downarrow\tau.$$

The set of variants represents a pre-computation that matches all the normal forms a term may be reduced to, without further applications of a rewriting rule. We note $\mathsf{variants_C}^{\mathcal{R}}(t_1, \ldots, t_n)$ the restriction of $\mathsf{variants}^{\mathcal{R}}(t_1, \ldots, t_n)$ to substitutions σ such that $t_1\sigma\downarrow, \ldots t_n\sigma\downarrow$ are constructor terms.

Example 3.1. *Continuing Example 2.3 we have that $\mathcal{R}_{ex}$ satisfies the finite variant property. Given $t_1 = \mathsf{adec}(x, \mathsf{sk}(v_0))$ and $t_2 = \mathsf{eq}(\mathsf{proj}_1(t_1), \mathsf{check}(\mathsf{proj}_2(t_1), \mathsf{spk}(p_0)))$ we have that (x, t_1, t_2) admits $\{id, \sigma_1, \sigma_2, \sigma_3\}$ as a finite set of variants with id the identity substitution, $\sigma_1 = \{x \to \mathsf{aenc}(y, \mathsf{pk}(v_0))\}$, $\sigma_2 = \{x \to \mathsf{aenc}(\langle y_1, y_2\rangle, \mathsf{pk}(v_0))\}$, and $\sigma_3 = \{x \to \mathsf{aenc}(\langle y_1, \mathsf{sign}(y_1, \mathsf{ssk}(p_0))\rangle, \mathsf{pk}(v_0))\}$. Amongst $\{id, \sigma_1, \sigma_2, \sigma_3\}$, only σ_3 belongs to $\mathsf{variants}_{\mathsf{C}\mathcal{R}_{ex}}(x, t_1, t_2)$.*

In our model we were considering both an equational theory $\mathtt{E}$ and a rewriting system $\mathcal{R}$. In this chapter, we assume that $\mathtt{E}$ can be represented by a convergent rewriting system, denoted $\mathcal{R}(\mathtt{E})$. Moreover, $\mathcal{R}(\mathtt{E}) \cup \mathcal{R}$ must generate a convergent rewriting system that satisfies the finite variant property. In the following, we note $u\Downarrow$ the normal form of a term and $\mathsf{variants_C}(u_1, \ldots, u_n) = \mathsf{variants_C}^{\mathcal{R}(\mathtt{E})\cup\mathcal{R}}$ the given set of variants in this extended rewriting system. In this chapter, without further details, we always consider terms w.r.t. this extended rewriting system.

Proposition 3.1. *Given a set of atomic data A and a term $u \in \mathcal{T}(\Sigma^+, A)$ we have:*

$$u\Downarrow =_{\mathtt{E}} u\downarrow.$$

Proof. The proof is done by induction on the number k of rewriting rules that are applied to compute $u{\Downarrow}$. If $k = 0$ then $u{\Downarrow} = u$ and thus $u{\downarrow} = u$ because $\mathcal{R} \subseteq \mathcal{R}(\mathtt{E}) \cup \mathcal{R}$. Otherwise, we have that $u = u_0 \Rightarrow u_1 \Rightarrow \ldots \Rightarrow u_n = u{\Downarrow}$. We perform a case analysis on the first rule:

- The rule belongs to $\mathcal{R}$: the same rule applies and we obtain that $u_0{\downarrow} = u_1{\downarrow}$. By induction hypothesis we have $u_1{\Downarrow} =_{\mathtt{E}} u_1{\downarrow}$, and thus $u_0{\Downarrow} = u_1{\Downarrow} =_{\mathtt{E}} u_1{\downarrow} = u_0{\downarrow}$.
- The rule belongs to $\mathcal{R}(\mathtt{E})$: by definition of $\mathcal{R}(\mathtt{E})$ we have that $u_0 =_{\mathtt{E}} u_1$. Applying the induction hypothesis on u_1 we have that $u_1{\Downarrow} =_{\mathtt{E}} u_1{\downarrow}$ and thus $u_0{\Downarrow} =_{\mathtt{E}} u_1{\Downarrow} =_{\mathtt{E}} u_1{\downarrow} =_{\mathtt{E}} u_0{\downarrow}$.

□

Remark 3.1. *We may note that* $\mathtt{E}_{\oplus}$*, the equational theory modelling the exclusive-or operator and presented in Example 2.2, cannot be represented by a convergent rewriting system. More generally, no equational theory that models an associative and commutative operator can be translated into a convergent rewriting system.*

3.1.3. Symbolic traces

Verifying protocols considering a bounded number of sessions enables to make some simplifications. Similarly to Akiss' procedure, instead of considering multiset of processes as defined in Chapter 2, our procedure will rely on symbolic traces. Indeed, because we consider a bounded number of sessions, protocols can be entirely described by a finite set of traces obtained by computing all the possible interleavings of the processes under study. Formally, a *symbolic trace* is a finite sequence (possibly empty and denoted by ϵ in this case) of pairs, i.e. $T = (a_1, \alpha_1).\ldots.(a_n, \alpha_n)$ where $a_i \in \mathcal{A}$ and α_i is a command of the form:

$$\mathtt{in}^z(x) \quad \mathtt{out}^z(u) \quad \mathtt{let}_{\mathtt{mess}}\ y = v \quad \mathtt{let}_{\mathtt{time}}\ z = v \quad \mathtt{if}_{\mathtt{time}}\ t_1 \sim t_2 \quad \mathtt{end}(u_1, u_2)$$

with $x, y \in \mathcal{X}$, $z \in \mathcal{Z}$, $u, u_1, u_2 \in \mathcal{T}(\Sigma_c^+, \mathcal{N} \cup \mathcal{A} \cup \mathbb{R}_+ \cup \mathcal{X})$, $v \in \mathcal{T}(\Sigma^+, \mathcal{N} \cup \mathcal{A} \cup \mathbb{R}_+ \cup \mathcal{X})$ and $t_1 \sim t_2$ a timing constraint.

These commands are exactly the same as those presented in Chapter 2. We simply remove the `new` command without loss of expressiveness: considering a bounded number of sessions we can assume that the fresh names are pre-generated since the beginning.

Even if the names can no longer be bound, we assume the usual definition of *free* and *bound* variables for inputs and outputs, i.e. x and z are bound variables in the previous commands. However, for sake of simplicity, in the proofs we assume that the $\mathtt{let}_{\mathtt{mess}}$ commands do no bind the variable y. The term on the right hand side of the command must be repeated as often as needed in the remaining commands.

Note that following this definition of symbolic trace, a variable that appears in a command executed by an agent a may be bound by a command executed by an agent b, different from a. To avoid such unrealistic situations we say that a trace is *locally closed w.r.t. a given set of variables* $\mathcal{V}$ if for any agent a, the trace obtained by considering

commands executed by agent a does not contain free variables among those in $\mathcal{V}$. One may note that the traces obtained by interleaving commands of roles, as defined in Chapter 2, are always locally closed w.r.t. $\mathcal{X}$ and $\mathcal{Z}$.

Example 3.2. *We consider the SPADE protocol presented in Chapter 1 and the roles presented in Example 2.5. To analyse this protocol w.r.t. to a simple scenario in which an agent v_0 executes one verifier session and an agent p_0 executes one prover session, we will consider the finite set of symbolic traces obtained by computing all the possible interleavings of the two roles presented in Example 2.5. To emphasise the end of the verifier session the command* $\mathsf{end}(v_0, p_0)$ *is added at its end. Moreover, as said before, since we consider only two sessions, we can assume w.l.o.g. that names are bound only once and thus the* `new` *commands are removed. An example of such a trace is the following, in which variables defined through a* $\mathtt{let}_{\mathtt{mess}}$ *command have been replaced by their corresponding terms as often as necessary:*

$$
\begin{aligned}
T_{ex} = \quad & (p_0, \mathtt{out}(\mathsf{aenc}(\langle n_P, \mathsf{sign}(n_P, \mathsf{ssk}(p_0))\rangle, \mathsf{pk}(v_0)))).\\
& (v_0, \mathtt{in}(x_1^v)).\\
& (v_0, \mathtt{let}_{\mathtt{mess}}\ y_1 = \mathsf{adec}(x_1^v, \mathsf{sk}(v_0))).\\
& (v_0, \mathtt{let}_{\mathtt{mess}}\ y^1_{\mathsf{check}} = \mathsf{eq}(\mathsf{proj}_1(y_1), \mathsf{check}(\mathsf{proj}_2(y_1), \mathsf{spk}(p_0)))).\\
& (v_0, \mathtt{out}(\langle m_V, n_V\rangle)).\\
& (p_0, \mathtt{in}(x_1^p)).\\
& (p_0, \mathtt{let}_{\mathtt{mess}}\ x_m = \mathsf{proj}_1(x_1^p)).\\
& (p_0, \mathtt{let}_{\mathtt{mess}}\ x_n = \mathsf{proj}_2(x_1^p)).\\
& (p_0, \mathtt{let}_{\mathtt{mess}}\ x^p_{H_0} = \mathsf{prf}(\langle n_P, x_n\rangle)).\\
& (p_0, \mathtt{let}_{\mathtt{mess}}\ x^p_{H_1} = n_P \oplus x_m \oplus x^p_{H_0}).\\
& (v_0, \mathtt{out}^{z_1}(c)).\\
& (p_0, \mathtt{in}(x_2^p)).\\
& (p_0, \mathtt{out}(\mathsf{answer}(x_2^p, x^p_{H_0}, x^p_{H_1}))).\\
& (v_0, \mathtt{in}^{z_2}(x_2^v)).\\
& (v_0, \mathtt{if}_{\mathtt{time}}\ z_2 - z_1 < 2 \times t_0).\\
& (p_0, \mathtt{out}(\mathsf{prf}(\langle n_P, x_n, x_m, x_2^p, \mathsf{answer}(x_2, x^p_{H_0}, x^p_{H_1})\rangle))).\\
& (v_0, \mathtt{in}(x_3^v)).\\
& (v_0, \mathtt{let}_{\mathtt{mess}}\ x^v_{H_0} = \mathsf{prf}(\langle \mathsf{proj}_1(y_1), n_V\rangle)).\\
& (v_0, \mathtt{let}_{\mathtt{mess}}\ x^v_{H_1} = \mathsf{proj}_1(y_1) \oplus m_V \oplus x^v_{H_0}).\\
& (v_0, \mathtt{let}_{\mathtt{mess}}\ y^2_{\mathsf{check}} = \mathsf{eq}(x_2^v, \mathsf{answer}(c, x^v_{H_0}, x^v_{H_1}))).\\
& (v_0, \mathtt{let}_{\mathtt{mess}}\ y^3_{\mathsf{check}} = \mathsf{eq}(x_3^v, \mathsf{prf}(\langle \mathsf{proj}_1(y_1), n_V, m_V, c, x_2^v\rangle))).\\
& (v_0, \mathsf{end}(v_0, p_0)).
\end{aligned}
$$

For sake of simplicity in the following of the development, we introduce the following notations: $u_1 = \mathsf{adec}(x_1^v, \mathsf{sk}(v_0))$, $u_m = \mathsf{proj}_1(x_1^p)$, $u_n = \mathsf{proj}_2(x_1^p)$, $u^p_{H_0} = \mathsf{prf}(\langle n_P, u_n\rangle)$, $u^p_{H_1} = n_P \oplus n_m \oplus n^p_{H_0}$, $u^v_{H_0} = \mathsf{prf}(\langle \mathsf{proj}_1(u_1), n_V\rangle)$, *and* $u^v_{H_1} = proj_1(u_1) \oplus m_V \oplus u^v_{H_0}$.

To model the initial knowledge of a dishonest agent we consider a subtrace that outputs all the messages that are necessary. This subtrace is then prepended to the traces of the protocol. Assuming that a dishonest agent must know the identities of the agents involved in the configuration and his secret keys (as presented in Example 2.10 with the template $\mathcal{I}_0 = \{\mathsf{x}_1, \mathsf{sk}(\mathsf{x}_0), \mathsf{ssk}(\mathsf{x}_0)\}$*, the subtrace may be:*

$$T_{dis} = (p, \mathtt{out}(p)).(p, \mathtt{out}(p_0)).(p, \mathtt{out}(v_0)).(p, \mathtt{out}(\mathsf{sk}(p))).(p, \mathtt{out}(\mathsf{ssk}(p))).$$

Semantics

The semantics of the commands remains the same as the one presented in Chapter 2. We simply adapt the notion of configuration replacing the multiset of extended processes by a symbolic trace, i.e. $\mathcal{K} = (T; \Phi; t)$. In addition, we define new labels to ease the theoretical development: instead of using τ labels when executing $\mathtt{let}_{\mathtt{mess}}$ or $\mathtt{let}_{\mathtt{time}}$ commands, we make the trace more expressive using $\mathtt{letM}(v\Downarrow)$ or $\mathtt{letT}(v\Downarrow)$ labels. Similarly the τ labels corresponding to timing constraints are replaced by the label $\mathtt{test}$. The semantics is reminded in Figure 3.1.

$$\text{TIM} \quad (T; \Phi; t) \longrightarrow_{\mathcal{T}_0} (T; \Phi; t+\delta) \qquad \text{with } \delta \geq 0$$

$$\text{LET\_M} \quad ((a, \mathtt{let}_{\mathtt{mess}}\ x = v).T; \Phi; t) \xrightarrow{a,\mathtt{letM}(v\Downarrow)}_{\mathcal{T}_0} (T\{x \to v\Downarrow\}; \Phi; t) \quad \text{when } v\Downarrow \neq \bot$$

$$\text{LET\_T} \quad ((a, \mathtt{let}_{\mathtt{time}}\ z = v).T; \Phi; t) \xrightarrow{a,\mathtt{letT}(v\Downarrow)}_{\mathcal{T}_0} (T\{z \to v\Downarrow\}; \Phi; t) \quad \text{when } v\Downarrow \in \mathbb{R}_+$$

$$\text{IF} \quad ((a, \mathtt{if}_{\mathtt{time}}\ t_1 \sim t_2).T; \Phi; t) \xrightarrow{a,\mathtt{test}}_{\mathcal{T}_0} (T; \Phi; t) \text{ if } t_1 \sim t_2 \text{ is true}$$

$$\text{OUT} \quad ((a, \mathtt{out}^z(u)).T; \Phi; t) \xrightarrow{a,\mathtt{out}(u\Downarrow)}_{\mathcal{T}_0} (T'; \Phi \uplus \{\mathsf{w} \xrightarrow{a,t} u\Downarrow\}; t) \quad \text{with } \mathsf{w} \in \mathcal{W} \text{ fresh and } T' = T\{z \to t\}$$

$$\text{IN} \quad ((a, \mathtt{in}^z(x)).T; \Phi; t) \xrightarrow{a,\mathtt{in}(u)}_{\mathcal{T}_0} (T'; \Phi; t)$$

when $u \neq \bot$ and there exist $b \in \mathcal{A}_0$ and $t_b \in \mathbb{R}$ such that $t_b \leq t - \mathsf{Dist}_{\mathcal{T}_0}(b, a)$ and:

- if $b \in \mathcal{A}_0 \setminus \mathcal{M}_0$ then $u \in img(\lfloor \Phi \rfloor_b^{t_b})$;
- if $b \in \mathcal{M}_0$ then $u = R\Phi\Downarrow$ for some recipe R such that for all $\mathsf{w} \in vars(R)$ there exists $c \in \mathcal{A}_0$ such that $\mathsf{w} \in dom(\lfloor \Phi \rfloor_c^{t_b - \mathsf{Dist}_{\mathcal{T}_0}(c,b)})$.

Figure 3.1: Semantics of our trace-based calculus w.r.t. a topology $\mathcal{T}_0$

Example 3.3. *Continuing Example 3.2, since the trace* T_{ex} *models a normal interleaving between the two agents* v_0 *and* p_0*, it can be fully executed starting with an empty frame when considering the trivial topology made of the two agents located at the same place, i.e.*

$\mathcal{T} = (\{v_0, p_0\}, \emptyset, \mathsf{Loc}, v_0, p_0)$ *with* $\mathsf{Loc}(v_0) = \mathsf{Loc}(p_0)$. *Since they share the same location, we do not even need to apply the TIM rule The execution starts as follows:*

$$
\begin{aligned}
(T_{ex}; \emptyset; 0) \quad & \xrightarrow{p_0, \mathtt{out}(\mathsf{aenc}(\langle n_P, \mathsf{sign}(n_P, \mathsf{ssk}(p_0))\rangle, \mathsf{pk}(v_0)))} \mathcal{T} \\
& \xrightarrow{v_0, \mathtt{in}(\mathsf{aenc}(\langle n_P, \mathsf{sign}(n_P, \mathsf{ssk}(p_0))\rangle, \mathsf{pk}(v_0)))} \mathcal{T} \\
& \xrightarrow{v_0, \mathtt{letM}(\langle n_P, \mathsf{sign}(n_P, \mathsf{ssk}(p_0))\rangle)} \mathcal{T} \xrightarrow{v_0, \mathtt{letM}(\mathsf{ok})} \mathcal{T} \xrightarrow{v_0, \mathtt{out}(\langle m_V, n_V\rangle)} \mathcal{T} \\
& \xrightarrow{p_0, \mathtt{in}(\langle m_V, n_V\rangle)} \mathcal{T} \xrightarrow{p_0, \mathtt{in}(\langle m_V, n_V\rangle)} \mathcal{T} \xrightarrow{p_0, \mathtt{letM}(m_V)} \mathcal{T} \xrightarrow{p_0, \mathtt{letM}(n_V)} \mathcal{T} \\
& \to_{\mathcal{T}} \ldots \to_{\mathcal{T}} (\epsilon; \Phi; 0)
\end{aligned}
$$

with

$$
\begin{aligned}
\Phi = \{ \quad & \mathsf{w}_1 \to \mathsf{aenc}(\langle n_P, \mathsf{sign}(n_P, \mathsf{ssk}(p_0))\rangle, \mathsf{pk}(v_0)), \\
& \mathsf{w}_2 \to \langle m_V, n_V\rangle, \mathsf{w}_3 \to c, \mathsf{w}_4 \to \mathsf{answer}(c, H^0, H^1), \\
& \mathsf{w}_5 \to \mathsf{prf}(\langle n_P, n_V, m_V, c, \mathsf{answer}(c, H^0, H^1)\rangle) \qquad \}
\end{aligned}
$$

where $H^0 = \mathsf{prf}(\langle n_P, n_V\rangle)$ *and* $H^1 = n_P \oplus m_V \oplus H^0$.

3.2. A model based on Horn clauses

The procedure we are going to present relies on an abstract modelling of traces using first-order Horn clauses, called *seed statements*. Before to describe these statements we introduce the predicates on which these statements are built on.

3.2.1. Predicates

Our predicates are built over *symbolic runs* which are finite sequences of labels possibly ending with a *run variable*, denoted y. Following the semantics presented in Figure 3.1, we have that each pair (a, α) is such that $a \in \mathcal{A}$ and α is an action of the form:

$$\mathtt{out}(u) \quad \mathtt{in}(u) \quad \mathtt{letM}(u) \quad \mathtt{letT}(u) \quad \mathtt{test}$$

with $u \in \mathcal{T}(\Sigma^+, \mathcal{N} \cup \mathcal{A} \cup \mathbb{R}_+ \cup \mathcal{X})$. By construction, excluding the run variable y, a symbolic run only contains message variables, i.e. from $\mathcal{X}$. As defined for symbolic traces, we say that a symbolic run $(a_1, \alpha_1), \ldots, (a_n, \alpha_n)$ is *locally closed* (w.r.t. $\mathcal{X}$) if whenever a variable $x \in \mathcal{X}$ occurs in an output or a `let` action α_i then there exists an input action α_j before (i.e. $j < i$) such that $a_j = a_i$ and $x \in vars(\alpha_j)$. Symbolic runs are often denoted $w, w', w_i, \ldots$ and we write $w \sqsubseteq w'$ when the sequence w is a prefix of w'. Given a symbolic run w whose sequence of outputs is $\mathtt{out}(u_1).\ldots.\mathtt{out}(u_n)$, we denote $\phi(w) = \{\mathsf{w}_1 \to u_1, \ldots, \mathsf{w}_n \to u_n\}$ its corresponding *symbolic frame*. Besides symbolic runs and frames we also consider *symbolic recipes*, i.e. recipes that may contain variables. They are terms in $\mathcal{T}(\Sigma^+_{\mathsf{pub}}, \mathcal{W} \cup \mathcal{Y})$ where $\mathcal{Y}$ is a set of recipe variables disjoint from $\mathcal{X}$, $\mathcal{W}$ and $\mathcal{Z}$. Recipe variables are denoted by capital letters, e.g. X, Y, Z...

Example 3.4. *We follow the notations introduced in Example 3.2 and provide the sym-*

bolic run that corresponds to T_{ex}:

$$\begin{aligned} w_0 = \quad & (p_0, \mathtt{out}(\mathsf{aenc}(\langle n_P, \mathsf{sign}(n_P, \mathsf{ssk}(p_0))\rangle, \mathsf{pk}(v_0)))). \\ & (v_0, \mathtt{in}(x_1^v)).(v_0, \mathtt{letM}(\mathsf{adec}(x_1^v, \mathsf{sk}(v_0)))). \\ & (v_0, \mathtt{letM}(\mathsf{eq}(\mathsf{proj}_1(u_1), \mathsf{check}(\mathsf{proj}_2(u_1), \mathsf{spk}(p_0))))).(v_0, \mathtt{out}(\langle m_V, n_V\rangle)) \\ & (p_0, \mathtt{in}(x_1^p)).(p_0, \mathtt{letM}(\mathsf{proj}_1(x_1^p))).(p_0, \mathtt{letM}(\mathsf{proj}_2(x_1^p))).\ldots \end{aligned}$$

We have that:

$$\begin{aligned} \phi(w_0) = \{ \quad & \mathsf{w}_1 \to \mathsf{aenc}(\langle n_P, \mathsf{sign}(n_P, \mathsf{ssk}(p_0))\rangle, \mathsf{pk}(v_0)), \\ & \mathsf{w}_2 \to \langle m_V, n_V\rangle, \mathsf{w}_3 \to c, \mathsf{w}_4 \to \mathsf{answer}(x_2^p, u_{H_0}^p, u_{H_1}^p), \\ & \mathsf{w}_5 \to \mathsf{prf}(\langle n_P, \mathsf{proj}_2(x_1^p), \mathsf{proj}_1(x_1^p), x_2^p, \mathsf{answer}(x_2^p, u_{H_0}^p, u_{H_1}^p)\rangle) \quad \} \end{aligned}$$

.

Horn clauses are built over two predicates expressing deduction and reachability but without taking time into account (timing constraints will be taken into account at the last step of the procedure). More formally, given a configuration $(T; \Phi; t)$, its untimed counterpart is $(T; \phi)$ where ϕ is the untimed counterpart of Φ, i.e. the frame Φ without agent and time annotations. The untimed semantics is given in Figure 3.2. Since time variables are not instantiated during a relaxed execution, in an untimed configuration $(T; \phi)$, the trace is only locally closed w.r.t. $\mathcal{X}$. Our predicates are:

- a *reachability predicate:* r_w holds when the run w is executable.
- a *deduction predicate:* $\mathsf{k}_w(R, u)$ holds if an attacker can deduce the message u applying the recipe $R \in \mathcal{T}(\Sigma^+_{\mathsf{pub}}, \mathbb{R}_+ \cup \mathcal{W})$ to the frame resulting of the execution of the run w (if this execution is possible).

Formally we have:

$$\begin{aligned} & (T_0; \phi_0) \models \mathsf{r}_{\ell_1,\ldots,\ell_n} && \text{if } (T_0; \phi_0) \overset{\ell_1\ldots\ell_n}{\rightsquigarrow} (T_n; \phi_n) \\ & (T_0; \phi_0) \models \mathsf{k}_{\ell_1,\ldots,\ell_n}(R, u) && \text{if } R\phi_n{\Downarrow} = u \text{ when } (T_0; \phi_0) \overset{\ell_1\ldots\ell_n}{\rightsquigarrow} (T_n; \phi_n) \end{aligned}$$

The semantics of these predicates is extended to first order logic formulas as usual using conjunction, disjunction, negation...

3.2.2. The seed statements

The first step of the procedure consists in computing a finite set of Horn clauses that represent all the possible executions of a symbolic trace. To do so, we consider particular Horn clauses called *statements*.

Definition 3.2. *A* statement *is a Horn clause: $H \Leftarrow \mathsf{k}_{w_1}(X_1, u_1), \ldots, \mathsf{k}_{w_n}(X_n, u_n)$ with $H \in \{\mathsf{r}_{w_0}, \mathsf{k}_{w_0}(R, u)\}$ and such that:*

- *$w_0, \ldots, w_n$ are locally closed symbolic runs;*
- *$w_i \sqsubseteq w_0$ for any $i \in \{1, \ldots, n\}$;*

$$\text{LET\_M} \quad ((a, \mathtt{let}_{\mathtt{mess}}\ x = v).T; \phi) \xrightarrow{a,\mathtt{letM}(v\Downarrow)} (T\{x \mapsto v\Downarrow\}; \phi) \quad \text{when } v\Downarrow \in \mathcal{T}(\Sigma_c^+, \mathcal{N} \cup \mathcal{A} \cup \mathbb{R}_+)$$

$$\text{LET\_T} \quad ((a, \mathtt{let}_{\mathtt{time}}\ x = v).T; \phi) \xrightarrow{a,\mathtt{letT}(v\Downarrow)} (T; \phi)$$

$$\text{TEST} \quad ((a, \mathtt{if}_{\mathtt{time}}\ t_1 \sim t_2).T; \phi) \xrightarrow{a,\mathtt{test}} (T; \phi)$$

$$\text{OUT} \quad ((a, \mathsf{out}^z(u)).T; \phi) \xrightarrow{a,\mathsf{out}(u\Downarrow)} (T; \phi \uplus \{\mathsf{w} \to u\Downarrow\}) \text{ with } \mathsf{w} \in \mathcal{W} \text{ fresh}$$

$$\text{IN} \quad ((a, \mathsf{in}^z(x)).T; \phi) \xrightarrow{a,\mathsf{in}(u)} (T\{x \to u\}; \phi) \quad \text{when } u = R\phi\Downarrow \text{ for some timed-recipe } R \text{ and } u \neq \bot.$$

Figure 3.2: Untimed semantics of symbolic traces.

- $u, u_1, \ldots, u_n$ *are terms in* $\mathcal{T}(\Sigma^+, \mathcal{A} \cup \mathcal{N} \cup \mathbb{R}_+ \cup \mathcal{X})$*;*
- $R \in \mathcal{T}(\Sigma^+, \mathcal{A} \cup \mathbb{R}_+ \cup \mathcal{W} \cup \{X_1, \ldots, X_n\}) \setminus \mathcal{Y}$*, and* $X_1, \ldots, X_n$ *are distinct variables from* $\mathcal{Y}$*.*

In addition, when $H = \mathsf{k}_{w_0}(R, u)$*, we assume that we have* $vars(u) \subseteq vars(u_1, \ldots, u_n)$*, and* $R\ (\{X_i \to u_i\} \uplus \phi(w_0))\Downarrow = u$*.*

When considering a statement we implicitly assume that all the variables are universally quantified. A statement is thus always ground. Nevertheless, we sometimes call σ a grounding substitution for a statement $H \Leftarrow (B_1, \ldots, B_n)$ when σ is grounding for each predicate $H, B_1, \ldots, B_n$. The *skeleton* of a statement f, denoted $\mathsf{skl}(f)$, is the statement where recipes are removed. We say that a skeleton is in normal form if all the terms that occur in it are in normal form and different from $\bot$. Our definition of statement is in line with the original one first proposed in [CCCK16]. In the original procedure, symbolic runs were assumed closed; we thus strengthen this hypothesis summing locally closed runs. Indeed, even if this assumption already held in the original procedure, it was not required to prove the completeness of the procedure: if no matter who binds a variable in an untimed semantics, it is meaningful when considering agents at different locations. Finally, we also state an additional invariant, $R(\{X_i \to u_i\} \uplus \phi(w_0))\Downarrow = u$, that will be useful to establish the completeness of our procedure too.

In order to define our set of seed statements, we have to fix some naming conventions. Given a trace T of the form $(a_1, \alpha_1).(a_2, \alpha_2).\ldots.(a_n, \alpha_n)$. We assume w.l.o.g. the following naming conventions:

1. if α_i is a receive action, then $\alpha_i = \mathsf{in}^{z_i}(x_i)$, and $\ell_i = (a_i, \mathsf{in}(x_i))$;
2. if α_i is a send action, then $\alpha_i = \mathsf{out}^{z_i}(u_i)$, and $\ell_i = (a_i, \mathsf{out}(u_i\Downarrow))$;
3. if α_i is a let message action, then $\alpha_i = (\mathtt{let}_{\mathtt{mess}}\ y_i = v_i)$ and $\ell_i = (a_i, \mathtt{letM}(v_i\Downarrow))$;
4. if α_i is a let time action, then $\alpha_i = (\mathtt{let}_{\mathtt{time}}\ z_i' = v_i)$ and $\ell_i = (a_i, \mathtt{letT}(v_i\Downarrow))$;

1. $\mathsf{r}_{\ell_1\tau\Downarrow\cdot\ldots\cdot\ell_n\tau\Downarrow} \Leftarrow \{\mathsf{k}_{\ell_1\tau\Downarrow\cdot\ldots\cdot\ell_{j-1}\tau\Downarrow}(X_j, x_j\tau\Downarrow)\}_{j\in\mathsf{Rcv}(n)}$
   for all $\tau \in \mathsf{variants}_{\mathsf{C}}(\ell_1, \ldots, \ell_n)$
2. $\mathsf{k}_{\ell_1\tau\Downarrow\cdot\ldots\cdot\ell_m\tau\Downarrow\cdot\mathsf{y}}(\mathsf{w}_{|\mathsf{Snd}(m)|}, u_m\tau\Downarrow) \Leftarrow \{\mathsf{k}_{\ell_1\tau\Downarrow\cdot\ldots\cdot\ell_{j-1}\tau\Downarrow}(X_j, x_j\tau\Downarrow)\}_{j\in\mathsf{Rcv}(m)}$
   for all $m \in \mathsf{Snd}(n)$;
   for all $\tau \in \mathsf{variants}_{\mathsf{C}}(\ell_1, \ldots, \ell_m)$
3. $\mathsf{k}_{\mathsf{y}}(c, c) \Leftarrow$
   for all $c \in \mathcal{C}$
4. $\mathsf{k}_{\mathsf{y}}(\mathsf{f}(Y_1, \ldots, Y_k), \mathsf{f}(y_1, \ldots, y_k)\tau\Downarrow) \Leftarrow \{\mathsf{k}_{\mathsf{y}}(Y_j, y_j\tau\Downarrow)\}_{j\in\{1,\ldots,k\}}$
   for all $\mathsf{f} \in \Sigma_{\mathsf{pub}}$ of arity k;
   for all $\tau \in \mathsf{variants}_{\mathsf{C}}(\mathsf{f}(y_1, \ldots, y_k))$.

Figure 3.3: The set $\mathsf{seed}(T, \mathcal{C})$ of seed statements.

5. if α_i is a if time action, then $\alpha_i = (\mathtt{if}_{\mathtt{time}}\ t_i \sim t'_i)$, and $\ell_i = (a_i, \mathtt{test})$.

For each $m \in \{0, \ldots, n\}$, the sets $\mathsf{Rcv}(m)$, $\mathsf{Snd}(m)$, $\mathsf{LetM}(m)$, $\mathsf{LetT}(m)$, and $\mathsf{Test}(m)$ respectively denote the set of indices of the receive, send, let message, let times and if time actions amongst $\alpha_1, \ldots, \alpha_m$. We denote by $|S|$ the cardinality of S.

Given a set $\mathcal{C} \subseteq \Sigma_0 \cup \mathbb{R}_+$, the set of *seed statements* associated to T and $\mathcal{C}$, denoted $\mathsf{seed}(T, \mathcal{C})$, is defined in Figure 3.3. If $\mathcal{C} = \Sigma_0 \cup \mathbb{R}_+$, then $\mathsf{seed}(T, \mathcal{C})$ is said to be the set of seed statements associated to T and in this case we write $\mathsf{seed}(T)$ as a shortcut for $\mathsf{seed}(T, \Sigma_0 \cup \mathbb{R}_+)$. When computing seed statements, we compute complete sets of variants that lead to constructor terms. This allows us to get rid of the rewrite system in the remainder of our procedure.

The seed statements, as defined in Figure 3.3, are in line with the definition introduced in the original procedure [CCCK16]. However, few changes might be noticed: first we no longer compute the most general unifiers of the equality tests occurring in the trace. Indeed, our model is slightly richer than theirs and allows to consider both constructor and destructor symbols. Equalities are modelled through the destructor symbol eq. Therefore, the computation of the variants, as defined in Section 3.1.2, will encompass this unification. Indeed, to reduce a term $u = \mathsf{eq}(u_1, u_2)$ onto a constructor term, the substitution τ will necessarily unify the two terms u_1, u_2. Finally, the second difference with the original definition of seed statements lies in the use of a trace variable y in statements of type 2, 3 and 4. This change appears as an optimisation in the Akiss implementation: it allows to not enumerate all the possible symbolic runs. We decided to follow this modification to stay as close to the implementation as possible.

Proposition 3.2. *Given a trace T locally closed w.r.t. $\mathcal{X}$, we have that $\mathsf{seed}(T_0)$ is a set of statements.*

Proof. We consider each type of statement separately and show that each item of Definition 3.2 is satisfied. We follow the notations introduced in Figure 3.3. First, we note that for $0 \leq m \leq n$, $\ell_1, \ldots, \ell_m$ is a symbolic run that is locally closed because the trace

T_0 is locally closed w.r.t. $\mathcal{X}$. Moreover, input actions are of the form $\mathsf{in}(x)$ with $x \in \mathcal{X}$, and thus the property of being locally closed is still satisfied by $\ell_1\tau\Downarrow \cdot \ldots \cdot \ell_m\tau\Downarrow$ since variables occurring in inputs cannot be discarded when applying the rewriting rules. The other items are immediately satisfied. □

Example 3.5. *We continue Examples 3.2 and 3.4 to illustrate the definition of the seed statements. For sake of simplicity we do not present an exhaustive enumeration of all the statements of* $\mathsf{seed}(T_{ex})$*. Instead, we provide and explain few of them:*

$$f_1 = \mathsf{k}_{w_0^1.\mathsf{y}}(\mathsf{w}_1, \mathsf{aenc}(\langle n_P, \mathsf{sign}(n_P, \mathsf{ssk}(p_0))\rangle, \mathsf{pk}(v_0))) \Leftarrow$$
$$f_2 = \mathsf{k}_{w_0^5.\mathsf{y}}(\mathsf{w}_2, \langle m_V, n_V\rangle) \Leftarrow \mathsf{k}_{w_0^1}(X_2, \mathsf{aenc}(\langle x_1^v, \mathsf{sign}(x_1^v, \mathsf{ssk}(p_0))\rangle \mathsf{pk}(v_0)))$$
$$f_3 = \mathsf{k}_{\mathsf{y}}(\mathsf{proj}_1(X), x) \Leftarrow \mathsf{k}_{\mathsf{y}}(X, \langle x, y\rangle)$$

where w_0^i *denotes the symbolic run made of the* i *first elements of* w_0*.*

The two first statements, i.e. f_1 *and* f_2*, are statements of type* 2 *that correspond to the two first outputs of the trace. The statement* f_1 *can be immediately deduced from the definition of the seed statement. Indeed, when computing the set of variants we obtain that* $\mathsf{variants}_{\mathsf{C}}(\mathsf{aenc}(\langle n_P, \mathsf{sign}(n_P, \mathsf{ssk}(p_0))\rangle, \mathsf{pk}(v_0)))$ *is reduced to the singleton identity. On the contrary, when considering the second output, the variant* σ_3 *(defined in Example 3.1) must be applied due to the two* LetM *actions occurring in* w_0^5*. Finally, the statement* f_3 *is a statement of type* 4 *which represents the capability of the attacker to retrieve the first component of a pair by applying the first projection symbol. All the remaining statements of the seed are generated in the same way.*

3.2.3. Soundness and completeness of the seed statements

In this section, we prove that the set of seed statements, $\mathsf{seed}(T)$, is a sound and complete abstraction of the possible executions of the given symbolic trace. We define $\mathcal{H}(\mathsf{seed}(T))$ an extension of $\mathsf{seed}(T)$ and show that it represents all the executions. Moreover the proof tree witnessing this fact must match with the relaxed execution we have considered. This extra condition is mandatory to establish the completeness of our procedure. Since our seed statements are similar to those presented in [CCCK16], most of the proofs are similar, thus we decided to highlight the changes.

Definition 3.3. *Given a set* K *of statements,* $\mathcal{H}(K)$ *is the smallest set of ground facts such that:*

$$\textsc{Conseq.}\ \frac{\begin{array}{ll} f = \Big(H \Leftarrow B_1, \ldots, B_n\Big) \in K & B_1\sigma \in \mathcal{H}(K),\ \ldots,\ B_n\sigma \in \mathcal{H}(K) \\ \sigma \text{ grounding for } f & \mathsf{skl}(f\sigma) \text{ in normal form} \end{array}}{H\sigma \in \mathcal{H}(K)}$$

Let $B_i = \mathsf{k}_{w_i}(X_i, u_i)$ *for* $i \in \{1, \ldots, n\}$*, and* w_0 *the symbolic run associated to* H *with* $v_1, \ldots, v_{k'}$ *the terms occurring in input in* w_0*. We say that such an instance of* Conseq matches with $\mathsf{exec} = (T; \emptyset) \overset{\ell_1,\ldots,\ell_p}{\rightsquigarrow} (S; \phi)$ *using* $R_1, \ldots, R_k$ *as input recipes if* $w_0\sigma \sqsubseteq \ell_1, \ldots, \ell_p$*, and there exist* $\hat{R}_1, \ldots, \hat{R}_{k'}$ *such that:*

- $\hat{R}_j(\{X_i \to u_i \mid 1 \leq i \leq n\} \uplus \phi(w_0))\Downarrow = v_j$ *for* $j \in \{1, \ldots, k'\}$*; and*
- $\hat{R}_j\sigma = R_i$ *for* $j \in \{1, \ldots, k'\}$.

This notion of matching is extended to a proof tree π as expected, meaning that all the instances of CONSEQ used in π satisfies the property. Given a proof tree π we note $\mathsf{nodes}(\pi)$ the set of all the facts that occur in it (i.e. the results of an application of the CONSEQ rule).

This notion of matching will be crucial to prove the completeness of the procedure. Indeed, if a statement represents an execution by the intermediate of the symbolic run that appears in its head, it completely omits the recipes that are used to trigger the inputs. This approach is safe when considering untimed executions because any recipe that forges the input message is suitable to trigger the action. However, this becomes unsafe when considering our timed semantics. Indeed, two recipes that deduce the same term may generate two different sets of constraints: one satisfiable, one not. We thus have to keep track of the original recipe to be sure that the witness trace of attack we generate will be executable in our timed semantics.

Soundness

The proof of soundness is made in two steps: first we prove that the statements in the seed are sound, and then we prove that the CONSEQ rule only generates sound statements.

Lemma 3.1. *Given a trace T that is locally closed w.r.t. $\mathcal{X}$ we have that:*

$$(T_0; \emptyset) \models g \text{ for any statement } g \in \mathsf{seed}(T_0).$$

Proof. We follow the original proof of [CCCK16] and consider each kind of statement separately. The proof is straightforward for statements of type 2, 3 and 4. We now consider a statement of type 1. Following the notations of Figure 3.3, we have: $g = (\mathsf{r}_{\ell_1\tau\Downarrow\cdot\ldots\cdot\ell_n\tau\Downarrow} \Leftarrow \{\mathsf{k}_{\ell_1\tau\Downarrow\cdot\ldots\cdot\ell_{j-1}\tau\Downarrow}(X_j, x_j\tau\Downarrow)\}_{j\in\mathsf{Rcv}(n)})$. To prove that $(T_0; \emptyset) \models g$, we assume $(T_0; \emptyset) \models \mathsf{k}_{\ell_1\tau\Downarrow\cdot\ldots\cdot\ell_{j-1}\tau\Downarrow}(X_j, x_j\tau\Downarrow)$ for any $j \in \mathsf{Rcv}(n)$ and prove by induction on p we have that for any $1 \leq p \leq n$ $(T_0; \emptyset)) \models \mathsf{r}_{\ell_1\tau\Downarrow\cdot\ldots\cdot\ell_p\tau\Downarrow}$.
Base case: We have that $p = 0$ and thus the property trivially holds.
Induction case: By induction hypothesis we have that $(T_0; \emptyset) \models \mathsf{r}_{\ell_1\tau\Downarrow\cdot\ldots\cdot\ell_p\tau\Downarrow}$ and thus:

$$(T_0; \emptyset) \overset{\ell_1\tau\Downarrow\cdot\ldots\cdot\ell_p\tau\Downarrow}{\rightsquigarrow} (T_p; \phi_p).$$

We prove that $(T_p; \phi_p) \overset{\ell_{p+1}\tau\Downarrow}{\rightsquigarrow} (T_{p+1}; \phi_{p+1})$ and thus $(T_0; \emptyset) \models \mathsf{r}_{\ell_1\tau\Downarrow\cdot\ldots\cdot\ell_p\tau\Downarrow\cdot\ell_{p+1}\tau\Downarrow}$. We consider each type of action separately. The original proof applies for all the actions except for the $\ell_{p+1}\tau\Downarrow = (a_{p+1}, \mathtt{letM}(v_{p+1}\tau\Downarrow))$ action. In this case, we know that, by definition of $\mathsf{variants}_{\mathsf{C}}(\ell_1, \ldots, \ell_n)$, the term $v_{p+1}\tau\Downarrow$ is a constructor term and thus the action can be triggered.

□

Lemma 3.2. *Let S be a set of statements such that for all $g \in S$ we have that $(T_0; \emptyset) \models g$. We have that $(T_0; \emptyset) \models g$ for any $g \in \mathcal{H}(S)$.*

The same lemma is stated and proved in the original paper [CCCK16]. Even if our definition of statement is a bit more restrictive then theirs, (e.g. locally closed assumption plus the extra invariant), the same proof applies. Indeed it only depends on the definition of the Conseq rule which remains unchanged.

We can now combine the two lemmas to establish the soundness of our seed statements.

Theorem 3.1. *Given a trace T_0 that is locally-closed w.r.t. $\mathcal{X}$ we have that:*

$$(T_0; \emptyset) \models g \text{ for any statement } g \in \mathcal{H}(\mathsf{seed}(T_0)).$$

Completeness

Actually the completeness of our procedure will be established w.r.t. a subset of recipes called *uniform recipes*. Informally, a recipe is uniform as soon as there is not two different sub-recipes deducing the same term.

Definition 3.4. *Given a frame ϕ, a recipe R is* uniform w.r.t. ϕ *if for any $R_1, R_2 \in st(R)$ such that $R_1\phi{\Downarrow} = R_2\phi{\Downarrow}$, we have that $R_1 = R_2$.*

Given a set S of statements, we say that a set $\{\pi_1, \ldots, \pi_n\}$ of proof trees in $\mathcal{H}(S)$ is uniform *if for any $\mathsf{k}_w(R_1, t)$ and $\mathsf{k}_w(R_2, t)$ that occur in $\{\pi_1, \ldots, \pi_n\}$, we have that $R_1 = R_2$.*

We are now able to prove the completeness of our seed statements following the original proof. We also explain how to prove that any execution of a trace T_0 which only involves uniform recipes has a counterpart in $\mathcal{H}(\mathsf{seed}(T_0))$ that is uniform too.

Theorem 3.2. *Let T_0 be a trace locally closed w.r.t. $\mathcal{X}$. If $\mathsf{exec} = (T_0; \emptyset) \overset{\ell_1,\ldots,\ell_p}{\rightsquigarrow} (S; \phi)$ with input recipes $R_1, \ldots, R_k$ that are uniform w.r.t. ϕ then*

1. $\mathsf{r}_{\ell_1,\ldots,\ell_p} \in \mathcal{H}(\mathsf{seed}(T_0))$*; and*
2. $\mathsf{k}_{\ell_1,\ldots,\ell_p}(R, R\phi{\Downarrow}) \in \mathcal{H}(\mathsf{seed}(T_0))$ *for all recipe R uniform w.r.t. ϕ.*

Moreover, we may assume that the proof tree witnessing these facts are uniform and match with exec *using $R_1, \ldots, R_k$ as input recipes.*

Proof. We follow the original proof but we have to show in addition that the proof trees witnessing these facts are uniform and match with exec and $R_1, \ldots, R_k$ as input recipes. We first explain how we construct a proof tree that is matching exec and then we show that it is uniform too.

When using an instance of a statement of type 3 (resp. type 4), the proof tree trivially matches with exec with input recipes $R_1, \ldots, R_k$ since the symbolic run is reduced to y. When considering an instance of a statement of type 1 (resp. type 2), we note k' the

number of input actions in the symbolic run of the head and it is sufficient to consider $X_1, \ldots, X_{k'}$ for $\hat{R}_1, \ldots, \hat{R}_{k'}$ to obtain a proof tree matching exec with $R_1, \ldots, R_k$.

Actually, the resulting proof tree is uniform. Indeed, when considering a statement of type 4 we will always have that the recipe in the head is uniform w.r.t. ϕ (either a subterm of the initial recipe R or a subterm of an input recipe) and thus the premises will respect the uniformity. In addition, we will always construct the same proof tree when considering a deduction fact corresponding to an input of the execution. □

3.3. The saturation step

The saturation procedure consists in computing a finite set of solved statements that can be used to decide whether a trace can be fully executed. We first formally define both the notion of *solved statements* and the saturation procedure. Then we establish the soundness and the completeness of this saturated set.

3.3.1. The saturation procedure

For termination purposes, the procedure manipulates a set of statements, called *knowledge base*, that meets some assumptions.

Definition 3.5. *Given a statement* $f = (H \Leftarrow B_1, \ldots, B_n)$*,*

- f *is said to be* solved *if* $B_i = \mathsf{k}_{w_i}(X_i, x_i)$ *with* $x_i \in \mathcal{X}$ *for all* $i \in \{1, \ldots, n\}$*.*
- f *is said to be* well-formed *if whenever it is solved and* $H = \mathsf{k}_w(R, u)$*, we have that* $u \notin \mathcal{X}$*.*

A set of well-formed *statements is called a* knowledge base*. If* K *is a knowledge base,* $\mathsf{solved}(K) = \{f \in K \mid f \text{ is solved }\}$*.*

The saturation procedure consists in doing resolutions, i.e. merging two statements as soon as the atom in the head of the first can be unified with an atom in the body of the second one. This merge is performed by applying the syntactic *most general unifier* of the two atoms.

To formally define the selection of the atom in the body (there may be several that match), we assume a selection function sel which returns $\bot$ when applied on a solved statements and an atom $\mathsf{k}_w(X, t)$ with $t \notin \mathcal{X}$ otherwise. Resolution must be performed on this selected atom.

$$\textsc{Res}\ \frac{\begin{array}{c} f : H \Leftarrow \mathsf{k}_w(X,t), B_1, \ldots, B_n \in K \text{ such that } \mathsf{k}_w(X,t) = \mathsf{sel}(f) \\ g : \mathsf{k}_{w'}(R',t') \Leftarrow B_{n+1}, \ldots, B_m \in \mathsf{solved}(K) \quad \sigma = \mathsf{mgu}(\mathsf{k}_w(X,t), \mathsf{k}_{w'}(R',t')) \end{array}}{h\sigma \qquad \text{where } h = \Big(H \Leftarrow B_1, \ldots, B_n, B_{n+1}, \ldots, B_m\Big)}$$

Example 3.6. *We continue Example 3.5 to illustrate the use of the resolution rule.*

The statement f_1 is solved and a resolution rule applies between f_1 and f_2, leading to the statement:

$$h_1 = \mathsf{k}_{w_0^5.\mathsf{y}}(\mathsf{w}_2, \langle m_V, n_V \rangle) \Leftarrow$$

We can continue by applying a resolution rule between f_3 and h_1 and we obtain the statement:

$$h_2 = \mathsf{k}_{w_0^5.\mathsf{y}}(\mathsf{proj}_1(\mathsf{w}_2), m_V) \Leftarrow$$

The saturation procedure keeps on applying the Res *rule, possibly considering previously generated statements like h_1 or h_2, until a fixed point is reached.*

Lemma 3.3. *Let f, g, and h as defined on the* Res *rule. If $\mathsf{skl}(h\sigma)$ is in normal form then $h\sigma$ is a statement.*

Proof. Let $H = \mathsf{k}_{w_0}(R_0, t_0)$, and $B_i = \mathsf{k}_{u_i}(X_i, t_i)$ for $i \in \{1, \ldots, m\}$. The two non trivial points to establish are the locally closed assumption and the extra invariant, i.e. $w_0\sigma$, $u_1\sigma, \ldots, u_m\sigma$ are locally closed and:

$$(R_0\sigma)(\{X_i \to t_i\sigma \mid 1 \leq i \leq m\} \uplus \phi(w_0\sigma))\Downarrow = t_0\sigma.$$

We can first note that the locally closed assumption is preserved by substitution. Then, since f and g are two statements we know that:

- $R_0(\{X \to t\} \uplus \{X_i \to t_i \mid 1 \leq i \leq n\} \uplus \phi(w_0))\Downarrow = t_0$; and
- $R'(\{X_i \to t_i \mid n+1 \leq i \leq m\} \uplus \phi(w'))\Downarrow = t'$.

Therefore, we have that:

$$\begin{aligned}
&R_0\sigma(\sigma_X \uplus \phi(w_0\sigma))\Downarrow \\
&= R_0\{X \to R'\}(\sigma_X \uplus \phi(w_0\sigma))\Downarrow \\
&= R_0(\{X \to R'(\sigma_X \uplus \phi(w_0\sigma))\} \uplus \sigma_X \uplus \phi(w_0\sigma))\Downarrow \\
&= R_0(\{X \to R'(\{X_i \to t_i \mid 1 \leq i \leq m\} \uplus \phi(w'))\sigma\} \uplus \sigma_X \uplus \phi(w_0\sigma))\Downarrow \\
&= R_0(\{X \to t'\sigma\} \uplus \sigma_X \uplus \phi(w_0\sigma))\Downarrow \\
&= (R_0(\{X \to t\} \uplus \{X_i \to t_i \mid 1 \leq i \leq m\} \uplus \phi(w_0)))\sigma\Downarrow \\
&= t_0\sigma\Downarrow = t_0\sigma
\end{aligned}$$

where $\sigma_X = \{X_i \to t_i\sigma \mid 1 \leq i \leq m\}$.

This concludes the proof. □

To avoid non-termination issues all the statements generated by the resolution rule are not automatically added into the knowledge base: they are given to an update function which decides whether it must be added or not. Indeed, some of them may be not well-formed or redundant with those that already belong to the base. When considering untimed protocols, redundancy is very common: for example there is no need to deduce the same term in more than one way. Unfortunately, this remark may be wrong when considering timed protocols. Indeed, depending on the location of each agent, two different outputs deducing the same term may be of great interest.

Example 3.7. *We consider the following trace:*

$$T = (a_1, \mathtt{out}(k)).(a_2, \mathtt{out}(k)).(b, \mathsf{in}^z(x)).(b, \mathtt{let}_{\mathtt{mess}}\ x = \mathsf{eq}(x,k)).(b, \mathtt{if}_{\mathtt{time}}\ z < 2)$$

and a topology $\mathcal{T}_0$ such that $\mathsf{Dist}_{\mathcal{T}_0}(a_1, b) = 10$ while $\mathsf{Dist}_{\mathcal{T}_0}(a_2, b) = 1$. The trace can be fully executed starting with an empty frame and a global time set to 0 by filling the input with the message outputted by a_2, i.e. w_2. Indeed, b and a_2 are close enough to pass the last time check. On the contrary, even if the recipe w_1 deduces the same term as w_2, i.e. k, the trace cannot be executed using this recipe: agent a_1 is too far to let b receive the message on time.

The original procedure of Akiss *will typically discard the statement $\mathsf{k}(\mathsf{w}_2, k) \Leftarrow$ (by replacing it with an identical statement) because it is subsumed by $\mathsf{k}(\mathsf{w}_1, k) \Leftarrow$. In our procedure, both can be useful, and thus we have to keep both of them.*

As illustrated above, the original procedure would discard too many recipes, thus to preserve the completeness in presence of time and locations, we design a new way of updating the database. Typically, we will keep more statements than the original Akiss procedure but still allow to discard some of them to help the saturation process to terminate. Indeed, the more statements the database will contain, the longer will be saturation process. In worst cases it may lead to non termination issues.

Definition 3.6. *The canonical form $f\Downarrow_c$ of a statement $f = (H \Leftarrow B_1, \ldots, B_n)$ is the statement obtained by applying the* Remove *rule given below as many times as possible.*

$$\textsc{Remove}\ \frac{H \Leftarrow \mathsf{k}_w(X,t),\ \mathsf{k}_w(Y,t),\ B_1, \ldots, B_n \qquad \text{with } X \notin vars(H)}{H \Leftarrow \mathsf{k}_w(Y,t),\ B_1, \ldots, B_n}$$

This rule consists in keeping only one representative for each deduced term in the body. Intuitively it is not necessary to use two different recipes to deduce the same term t at a given point of the execution (note that both predicates have the same work w), we can discard the one (when it exists) that does not appear in H.

The update of a knowledge base K by a statement f, noted $K \Cup \{f\}$ is defined by:

$$K \Cup \{f\} = \begin{cases} K & \text{if } \mathsf{skl}(f\Downarrow_c) \text{ is not in normal form} \\ K & \text{else if } f\Downarrow_c \text{ is not well-formed} \\ K \cup \{f\Downarrow_c\} & \text{otherwise} \end{cases}$$

The saturation procedure starts by considering an empty set of statements updated by all the seed statements. Formally, given a set S of statements (typically $\mathsf{seed}(T, \Sigma_0 \cup \mathbb{R}_+)$), we note $K_{\mathsf{init}}(S)$ the initial knowledge base, i.e. $K_{\mathsf{init}}(S) = (((\emptyset \Cup f_1) \Cup f_2) \Cup \ldots f_n)$ where $f_1, \ldots f_n$ is an enumeration of S. We sometimes write $K_{\mathsf{init}}(T)$ instead of $K_{\mathsf{init}}(\mathsf{seed}(T, \Sigma_0 \cup \mathbb{R}_+))$. The saturation procedure consists in applying the Res rule starting with $K_{\mathsf{init}}(T)$ until a fixed point is reached.

Proposition 3.3. *Given a set S of statements, $K_{\mathsf{init}}(S)$ (resp. $\mathsf{sat}(K_{\mathsf{init}}(S))$) is a knowledge base.*

Proof. Lemma 3.3 ensures that Horn clauses generated during the saturation procedure are statements as soon as their skeleton is in normal form. Actually, the update function will discard a statement when its skeleton is not in normal form, and thus, during saturation, we only consider statements. Now, it is easy to see that those statements are well-formed since the update function discards those that are not. Finally, it is quite straightforward that the canonical form of a statement is a statement too. This concludes the proof. □

3.3.2. Soundness

The soundness of the saturation procedure is quite straightforward. We first prove the soundness of the resolution rule, then the soundness of the canonicalisation and finally we rely on the soundness of the seed statements to conclude.

Lemma 3.4. *Let* $f = \Big(H \Leftarrow \mathsf{k}_w(X,t), B_1, \ldots, B_n\Big)$ *be a statement such that* $\mathsf{k}_w(X,t)$ *is the selected atom, i.e.* $\mathsf{k}_w(X,t) = \mathsf{sel}(f)$*, and* $g = \Big(\mathsf{k}_{w'}(R,t') \Leftarrow B_{n+1}, \ldots, B_m\Big)$ *be a solved statement. We note* $\sigma = \mathsf{mgu}(\mathsf{k}_w(X,t), \mathsf{k}_{w'}(R,t'))$*. Given a trace* T_0*, if* $(T_0;\emptyset) \models f$ *and* $(T_0;\emptyset) \models g$ *then* $(T_0;\emptyset) \models h$ *where:*

$$h = \Big(H \Leftarrow B_1, \ldots, B_n, B_{n+1}, \ldots, B_m\Big)\sigma.$$

Proof. Let τ be a grounding substitution for h such that $(T_0;\emptyset) \models B_i\sigma\tau$ for all $i \in \{1,\ldots,m\}$. Since $(T_0;\emptyset) \models B_i\sigma\tau$ for all $i \in \{n+1,\ldots,m\}$ and $(T;\emptyset) \models g$, we deduce that $(T_0;\emptyset) \models \mathsf{k}_{w'}(R,t')\sigma\tau$. We have that $\mathsf{k}_{w'}(R,t')\sigma\tau = \mathsf{k}_w(X,t)\sigma\tau$ by definition of σ. Therefore, since $(T_0;\emptyset) \models f$, we have that $(T_0;\emptyset) \models H\sigma\tau$ and thus we conclude that $(T_0;\emptyset) \models h$. □

Lemma 3.5. *Given a trace* T_0*, if* $(T_0;\emptyset) \models h$ *then* $(T_0;\emptyset) \models h{\Downarrow_c}$.

Proof. This result can be proved by induction on the number n of application of the rule Remove. Below, we show that the result is true for $n = 1$.

Let $f = (H \Leftarrow \mathsf{k}_w(X,t), \mathsf{k}_w(Y,t), B_1, \ldots, B_n)$ be a statement such that $X \notin vars(H)$, and $g = (H \Leftarrow \mathsf{k}_w(Y,t), B_1, \ldots, B_n)$ the statement obtained after the application of the rule Remove.

Let τ be a grounding substitution for g such that $(T_0;\emptyset) \models \mathsf{k}_w(Y,t)\tau$ and, for all $i \in \{1,\ldots,n\}$, $(T_0;\emptyset) \models B_i\tau$. Let $\tau' = \tau \cup \{X \mapsto Y\tau\}$. We have that all the antecedents of $f\tau'$ are true in $(T_0;\emptyset)$, and since $(T_0;\emptyset) \models f$ by hypothesis, we deduce that $(T_0;\emptyset) \models H\tau' = H\tau$ (remember that $X \notin vars(H)$). This allows us to conclude. □

Theorem 3.3. *Let* T_0 *be a trace locally closed w.r.t.* $\mathcal{X}$*,* $K = \mathsf{sat}(K_{\mathsf{init}}(T_0))$*. We have that* $(T_0;\emptyset) \models g$ *for any* $g \in \mathsf{solved}(K) \cup \mathcal{H}(\mathsf{solved}(K))$.

Proof. We first establish that $(T_0;\emptyset) \models g$ for any $g \in K$ by induction on the number of resolution step needed to produce g. If $g \in K_{\mathsf{init}}(T_0)$ then $g \in \mathsf{seed}(T_0)$ and thus applying

Theorem 3.1, we conclude that $(T_0;\emptyset) \models g$ (seed statements are already in canonical form). Otherwise, such a statement $g = g'\Downarrow_c$ with g' a statement obtained through the RES rule. We can thus apply the induction hypothesis on the two statements used to derive g', and rely on Lemma 3.4 and Lemma 3.5 to conclude. Finally, since $(T_0;\emptyset) \models g$ for any $g \in K$, we have that the property holds for any $g \in \mathsf{solved}(K)$.

Now, let $g \in \mathcal{H}(\mathsf{solved}(K))$. The fact that $(T_0;\emptyset) \models g$ is a direct consequence of Lemma 3.2. □

3.3.3. Completeness

Proving the completeness of the saturation procedure is more complex than proving the soundness. Indeed, to preserve the termination (in practice) we discard some redundant statements through the update function. By consequence we can not expect to be able to retrieve all the possible recipes. Hopefully, we will prove that the recipes we keep are enough to represent all the reachable configurations.

Asap recipes

We define a refinement of uniform recipes we call *asap recipes*. Informally, a recipe is asap if it allows one to deduce the term as soon as possible. This notion of asap recipes relies on two relations that order recipes.

We first define a coarse grain order solely based on the subterm relation and the data dependencies along an execution. Given an untimed execution $\mathsf{exec} = (T;\emptyset) \overset{\ell_1,\ldots,\ell_n}{\rightsquigarrow} (S;\phi)$ with input recipes $R_1,\ldots,R_k$, we define the following relations:

- $R' <^{\mathsf{sub}}_{\mathsf{exec}} R$ when R' is a strict subterm of R;
- $R <^{\mathsf{in}}_{\mathsf{exec}} \mathsf{w}$ when $\ell_i = (a, \mathtt{in}(u))$ with input recipe R and $\ell_j = (a, \mathtt{out}(u_j))$ with output recipe w for some agent a with $i < j$.

Then, $<_{\mathsf{exec}}$ is the smallest transitive relation over recipes built on $dom(\phi)$ that contains $<^{\mathsf{in}}_{\mathsf{exec}}$ and $<^{\mathsf{sub}}_{\mathsf{exec}}$. As usual, we denote $\leq_{\mathsf{exec}}$ the reflexive closure of $<_{\mathsf{exec}}$.

The fine grain order is much more precise and relies on the time at which each output has been performed. This order precisely defines which recipe is available first to deduce a term. However, since agents may have different locations, the first available recipe may be different for each agent. Given a topology $\mathcal{T}$ and a timed execution $\mathsf{exec} = (T_0;\emptyset;t_0) \xrightarrow{\ell_1,\ldots,\ell_n}_{\mathcal{T}} (S;\Phi;t)$ with $\Phi = \{\mathsf{w}_1 \xrightarrow{a_1,t_1} u_1, \ldots, \mathsf{w}_n \xrightarrow{a_n,t_n} u_n\}$, we denote by $\mathsf{agent}(\mathsf{w}_i)$ (resp. $\mathsf{time}(\mathsf{w}_i)$) the agent a_i (resp. the time t_i). The relation $<^a_{\mathsf{exec}}$ over $dom(\Phi) \times dom(\Phi)$ with $a \in \mathcal{A}$ is defined as follows: $\mathsf{w} <^a_{\mathsf{exec}} \mathsf{w}'$ when:

- either $\mathsf{time}(\mathsf{w}) + \mathsf{Dist}_{\mathcal{T}}(\mathsf{agent}(\mathsf{w}), a) < \mathsf{time}(\mathsf{w}') + \mathsf{Dist}_{\mathcal{T}}(\mathsf{agent}(\mathsf{w}'), a)$;
- or $\mathsf{time}(\mathsf{w}) + \mathsf{Dist}_{\mathcal{T}}(\mathsf{agent}(\mathsf{w}), a) = \mathsf{time}(\mathsf{w}') + \mathsf{Dist}_{\mathcal{T}}(\mathsf{agent}(\mathsf{w}'), a)$, and the output w occurs before w' in the execution exec.

This order is extended on recipes as follows: $R <^a_{\mathsf{exec}} R'$ when:

1. either $\mathsf{multi}_{\mathcal{W}}(R) <^{a}_{\mathsf{exec}} \mathsf{multi}_{\mathcal{W}}(R')$ where $\mathsf{multi}_{\mathcal{W}}(R)$ is the multiset of variables $\mathcal{W}$ occurring in R ordered using the multiset extension of $<^{a}_{\mathsf{exec}}$ on variables;

2. or $\mathsf{multi}_{\mathcal{W}}(R) = \mathsf{multi}_{\mathcal{W}}(R')$ and $|R| < |R'|$ where $|R|$ is the size (number of symbols) occurring in R;

3. or $\mathsf{multi}_{\mathcal{W}}(R) = \mathsf{multi}_{\mathcal{W}}(R')$, $|R| = |R'|$, and $|st_{\mathsf{eq}}(R)| < |st_{\mathsf{eq}}(R')|$ where $st_{\mathsf{eq}}(R) = \{(S, S') \in st(R) \times st(R) \mid S \neq S' \text{ and } S\Phi{\Downarrow} = S'\Phi{\Downarrow}\}$ is the set of pairs of distinct syntactic subterms of R that deduce the same term.

We have that $<^{a}_{\mathsf{exec}}$ is a well-founded order for any $a \in \mathcal{A}$ which is compatible with $<_{\mathsf{exec}}$, i.e. $R <_{\mathsf{exec}} R'$ implies $R <^{a}_{\mathsf{exec}} R'$ for any agent a.

Lemma 3.6. *Let* $\mathsf{exec} = (T_0; \emptyset; t_0) \xrightarrow{\ell_1,\ldots,\ell_n}_{\mathcal{T}} (S; \Phi; t_n)$ *be an execution w.r.t. a topology* $\mathcal{T} = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}_0, v_0, p_0)$ *with input recipes* $R_1, \ldots, R_k$*. Let* R *and* R' *be two recipes such that* $R <_{\mathsf{exec}} R'$*. We have that* $<^{a}_{\mathsf{exec}}$ *is a well-founded order and* $R <^{a}_{\mathsf{exec}} R'$ *for any* $a \in \mathcal{A}_0$*.*

Proof. Let $a \in \mathcal{A}_0$ and R, R' be two recipes such that $R <_{\mathsf{exec}} R'$. We prove each property separately.
$<^{a}_{\mathsf{exec}}$ is a well-founded order: Thanks to the finiteness of the execution, $<^{a}_{\mathsf{exec}}$ is a well-founded order on multiset of frame variables. Therefore, $<^{a}_{\mathsf{exec}}$ is a well-founded order by composition of well founded orders and the lexicographic order.
$R <^{a}_{\mathsf{exec}} R'$: Because $R <_{\mathsf{exec}} R'$, there exists a chain $R = R_0 <_{\mathsf{exec}} \ldots <_{\mathsf{exec}} R_n = R'$ such that each step corresponds to a step of $<^{\mathtt{in}}_{\mathsf{exec}}$ or $<^{\mathtt{sub}}_{\mathsf{exec}}$. We prove the property by induction on the length of this chain. Let us show that the property holds for one step. We distinguish two cases depending if $<^{\mathtt{in}}_{\mathsf{exec}}$ or $<^{\mathtt{sub}}_{\mathsf{exec}}$ has been applied:

- Case $R <^{\mathtt{in}}_{\mathsf{exec}} R'$. We know that $R' = \mathsf{w}$ and R is a recipe used to feed an input performed by $\mathsf{agent}(\mathsf{w})$. Observing that for all $\mathsf{w}' \in \mathit{vars}(R)$, we have that $\mathsf{time}(\mathsf{w}') + \mathsf{Dist}_{\mathcal{T}}(\mathsf{agent}(\mathsf{w}'), \mathsf{agent}(\mathsf{w})) \leq \mathsf{time}(\mathsf{w})$ we deduce that for all $\mathsf{w}' \in \mathit{vars}(R)$, we have that:

$$\begin{aligned}\mathsf{time}(\mathsf{w}') + \mathsf{Dist}_{\mathcal{T}}(\mathsf{agent}(\mathsf{w}'), \mathsf{agent}(\mathsf{w})) + \mathsf{Dist}_{\mathcal{T}}(\mathsf{agent}(\mathsf{w}), a)\\ \leq \mathsf{time}(\mathsf{w}) + \mathsf{Dist}_{\mathcal{T}}(\mathsf{agent}(\mathsf{w}), a).\end{aligned}$$

  This implies that $\mathsf{time}(\mathsf{w}') + \mathsf{Dist}_{\mathcal{T}}(\mathsf{agent}(\mathsf{w}'), a) \leq \mathsf{time}(\mathsf{w}) + \mathsf{Dist}_{\mathcal{T}}(\mathsf{agent}(\mathsf{w}), a)$ and thus $\mathsf{w}' <^{a}_{\mathsf{exec}} \mathsf{w}$. By definition of the multiset order, we have thus $R <^{a}_{\mathsf{exec}} R'$.

- Case $R <^{\mathtt{sub}}_{\mathsf{exec}} R'$. We know that R is a strict subterm of and R'. Thus, we have that $R <^{a}_{\mathsf{exec}} R'$.

□

We are now able to formally define the notion of *asap recipes*.

Definition 3.7. *Given a topology* $\mathcal{T} = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}_0, v_0, p_0)$ *and an execution* $\mathsf{exec} = (T_0; \emptyset; t_0) \xrightarrow{\ell_1,\ldots,\ell_n}_{\mathcal{T}_0} (S; \Phi; t)$*. A recipe* R *is* asap w.r.t. $a \in \mathcal{A}_0$ and exec *if:*

- *either* $R \in \mathcal{W} \cup \mathbb{R}_+$ *and* $\nexists R'$ *such that* $R' <_{\mathsf{exec}} R$ *and* $R'\Phi\Downarrow = R\Phi\Downarrow$*;*
- *or* $R = \mathsf{f}(R_1, \ldots, R_k)$ *with* $\mathsf{f} \in \Sigma$ *and there is no recipe* R' *such that* $R' <^a_{\mathsf{exec}} R$ *and* $R'\Phi\Downarrow = R\Phi\Downarrow$*.*

This definition of minimal recipes cleverly combines the orders previously defined to favour recipes that allow to deduce the term as soon as possible. However, it does not favour a complex recipe over an atomic one, i.e. a frame variable, since the former must be forged by a malicious agent while the last may be an output performed by an honest agent. Such a difference is discussed in the following example.

Example 3.8. *We consider the topology* $\mathcal{T}$ *made of two honest agents* v_0 *and* p_0 *located at the same place, i.e.* $\mathsf{Dist}_{\mathcal{T}}(v_0, p_0) = 0$*, an uninterpreted public symbol of function* f *of arity* 1*, and the simple trace made of two actions:* $T = (v_0, \mathtt{out}(\mathsf{f}(\mathsf{ok}))).(p_0, \mathtt{in}(\mathsf{f}(\mathsf{ok})))$*. We have that:*

$$\mathsf{exec} = (\mathcal{T}; \emptyset; 0) \xrightarrow{(v_0,\mathtt{out}(\mathsf{f}(\mathsf{ok}))).(p_0,\mathtt{in}(\mathsf{f}(\mathsf{ok})))}_{\mathcal{T}} (\epsilon; \{\mathsf{w}_1 \xrightarrow{v_0,0}, \mathsf{f}(\mathsf{ok})\}; 0)$$

using the input recipe $R = \mathsf{w}_1$*.*

By definition we have that $R' = \mathsf{f}(\mathsf{ok})$ *is a recipe (because* $\mathsf{ok} \in \Sigma_0$ *and* $\mathsf{f} \in \Sigma_{\mathsf{pub}}$*) such that* $R' <^{p_0}_{\mathsf{exec}} R$*. However,* exec *cannot be executed using the recipe* R' *since there is no malicious agent in the topology to forge this message. Definition 3.7 avoids such a problem by taking care of messages coming from honest agents through the coarse grain order.*

Before establishing the completeness of our saturation procedure, we prove two properties about asap recipes. First we prove the statement we made before: asap is a refinement of uniform. Then we prove that a subterm of an asap recipe is also asap..

Lemma 3.7. *Let* $\mathsf{exec} = (T_0; \emptyset; t_0) \xrightarrow{\ell_1,\ldots,\ell_n}_{\mathcal{T}_0} (S; \Phi; t_n)$ *be an execution w.r.t. a topology* $\mathcal{T}_0 = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}_0, v_0, p_0)$*. Let* $a \in \mathcal{A}_0$ *and* R *a recipe that is asap w.r.t.* a *and* exec*. We have that* R *is uniform w.r.t.* Φ*.*

Proof. To conclude, it is sufficient to show that $|st_{\mathsf{eq}}(R)| = 0$. Assume by contradiction that $|st_{\mathsf{eq}}(R)| > 0$, and consider one of the deepest subterm S of R such that $|st_{\mathsf{eq}}(S)| > 0$ i.e. there exist $R'', R' \in st(S)$ with $R'' \neq R'$ and $R''\Phi\Downarrow = R'\Phi\Downarrow$. Let p'' (resp p') be the position at which R'' (resp. R') occurs in R. We distinguish 3 cases:

1. Case $\mathsf{multi}_{\mathcal{W}}(R'') <^a_{\mathsf{exec}} \mathsf{multi}_{\mathcal{W}}(R')$. Let $\widetilde{R} = R[R'']_{p'}$. We have that $\widetilde{R}\Phi\Downarrow = R\Phi\Downarrow$ since $R''\Phi\Downarrow = R'\Phi\Downarrow$ and $\mathsf{multi}_{\mathcal{W}}(\widetilde{R}) <^a_{\mathsf{exec}} \mathsf{multi}_{\mathcal{W}}(R)$. This contradicts the fact that R is asap w.r.t. a and exec.
2. Case $\mathsf{multi}_{\mathcal{W}}(R'') = \mathsf{multi}_{\mathcal{W}}(R')$ and $|R''| < |R'|$. Let $\widetilde{R} = R[R'']_{p'}$. We have that $\widetilde{R}\Phi\Downarrow = R\Phi\Downarrow$ since $R''\Phi\Downarrow = R'\Phi\Downarrow$. Moreover, we have that $\mathsf{multi}_{\mathcal{W}}(\widetilde{R}) = \mathsf{multi}_{\mathcal{W}}(R)$, and $|\widetilde{R}| < |R|$. This contradicts the fact that R is asap w.r.t. a and exec.

3. Case $\mathsf{multi}_{\mathcal{W}}(R'') = \mathsf{multi}_{\mathcal{W}}(R')$ and $|R''| = |R'|$ and $|st_{\mathsf{eq}}(R'')| < |st_{\mathsf{eq}}(R')|$. Therefore we have that R'' and R' are two distinct strict subterms of S and $|st_{\mathsf{eq}}(R')| \geq 1$. This contradicts the choice of S.

This concludes the proof. □

Lemma 3.8. *Let* $\mathsf{exec} = (T_0; \emptyset; t_0) \xrightarrow{\ell_1,\ldots,\ell_n}_{\mathcal{T}_0} (S; \Phi; t_n)$ *be an execution w.r.t. a topology* $\mathcal{T}_0 = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}_0, v_0, p_0)$. *Let* $a \in \mathcal{A}_0$ *and* R *a recipe that is asap w.r.t.* a *and* exec. *For all* $S \in st(R)$, *we have that* S *is asap w.r.t.* a *and* exec.

Proof. Suppose by contradiction that there exists $R' \in st(R)$ such that R' is not asap w.r.t. a and exec. Let p' be the position in R such that $R|_{p'} = R'$. We distinguish two cases:

1. Case 1: $R' \in \mathcal{W} \cup \mathbb{R}_+$ and there exists R'' such that $R'' <_{\mathsf{exec}} R'$ and $R'\Phi\Downarrow = R''\Phi\Downarrow$.
2. Case 2: $R' = \mathsf{f}(R'_1, \ldots, R'_n)$ and there exists R'' such that $R'' <^a_{\mathsf{exec}} R'$ and $R'\Phi\Downarrow = R''\Phi\Downarrow$.

Note that, thanks to Lemma 3.6 applied in Case 1, we have that $R'' <^a_{\mathsf{exec}} R'$ in both cases. We distinguish 3 cases:

1. Case $\mathsf{multi}_{\mathcal{W}}(R'') <^a_{\mathsf{exec}} \mathsf{multi}_{\mathcal{W}}(R')$. Let $\widetilde{R} = R[R'']_{p'}$. We have that $\widetilde{R}\Phi\Downarrow = R\Phi\Downarrow$ and $\mathsf{multi}_{\mathcal{W}}(\widetilde{R}) <^a_{\mathsf{exec}} \mathsf{multi}_{\mathcal{W}}(R)$. This contradicts the fact that R is asap w.r.t. a and exec.
2. Case $\mathsf{multi}_{\mathcal{W}}(R'') = \mathsf{multi}_{\mathcal{W}}(R')$ and $|R''| < |R'|$. Considering $\widetilde{R} = R[R'']_{p'}$ also allows us to contradict the fact that R is asap w.r.t. a and exec.
3. Case $\mathsf{multi}_{\mathcal{W}}(R'') = \mathsf{multi}_{\mathcal{W}}(R')$ and $|R''| = |R'|$ and $|st_{\mathsf{eq}}(R'')| < |st_{\mathsf{eq}}(R')|$. This will imply that $|st_{\mathsf{eq}}(R')| > 0$ and thus implies that R' is not uniform w.r.t. Φ leading to a contradiction with the result obtained by applying Lemma 3.7.

This concludes the proof. □

Completeness

We are now able to establish the completeness of the saturation step. The proof relies on two intermediate lemmas: Lemma 3.10 the main Lemma that lifts a proof tree in $\mathcal{H}(\mathsf{seed}(T_0))$ in a proof tree in $\mathcal{H}(\mathsf{solved}(\mathsf{sat}(K_{\mathsf{init}}(T_0))))$. Its proof if done by induction on the proof tree in $\mathcal{H}(\mathsf{seed}(T_0))$ relying on Lemma 3.9 that lifts each step. Once these two lemmas are proved, the proof of the completeness becomes almost immediate relying on the completeness of the seed statements.

Lemma 3.9. *Let* $\mathsf{exec} = (T_0; \emptyset; t_0) \xrightarrow{\ell_1,\ldots,\ell_p}_{\mathcal{T}} (S; \Phi; t)$ *be an execution with input recipes* $R_1, \ldots, R_k$ *forged by* $b_1, \ldots, b_k$ *and such that each* R_j *with* $j \in \{1, \ldots, k\}$ *is uniform w.r.t.* Φ. *Let* $K = \mathsf{sat}(K_{\mathsf{init}}(T_0))$, *and* $g = (H \Leftarrow B_1, \ldots, B_n) \in K$ *be such that* u_0, *the*

underlying world of H, is locally closed. Let σ be a grounding substitution for g such that $\mathsf{skl}(g\sigma)$ *is in normal form, and g and σ match with* exec *and $R_1, \ldots, R_k$.*

Moreover, in case H is of the form $H = \mathsf{k}_{u_0}(R_H, t_H)$, we assume that $u_0\sigma = \ell_1, \ldots, \ell_{q-1}$ for some $q \in \mathsf{Rcv}(p)$ and $R_H\sigma$ is asap w.r.t. $b_{|\mathsf{Rcv}(q)|}$ and exec.

Assuming that $B_i\sigma \in \mathcal{H}(\mathsf{solved}(K))$ with a proof tree π_i matching with exec *and $R_1, \ldots, R_k$ for each $i \in \{1, \ldots, n\}$, and $\{\pi_1, \ldots, \pi_n\}$ is uniform, then we have that $H\sigma \in \mathcal{H}(\mathsf{solved}(K))$ with a proof tree π' matching with* exec *and $R_1, \ldots, R_k$, and such that* $\mathsf{nodes}(\pi') \subseteq \bigcup_{i\in\{1,\ldots,n\}} \mathsf{nodes}(\pi_i) \cup \{H\sigma\}$.

3

Proof. (Sketch of proof - see Appendix A for the full proof) The proof proceeds by induction on the sum of the sizes of the proof trees witnessing that $B_1\sigma, \ldots, B_n\sigma \in \mathcal{H}(\mathsf{solved}(K))$.

If the statement g is solved then we easily conclude. Otherwise, we prove that a RESOLUTION rule applies between g and a statement (that is solved) occurring in the proof tree of $B_i\sigma \in \mathcal{H}(\mathsf{solved}(K))$ (for some $i \in \{1 \ldots, n\}$). We note g' the resulting statement and $\sigma = \omega\tau$ with ω the most general unifier computed during the resolution. Two situations may happen:

1. $g'\Downarrow_c$ is added to the knowledge base;
2. $g'\Downarrow_c$ is not added to the knowledge base (i.e. it is discarded by the update function).

Case 1: we prove that $g'\Downarrow_c$ and τ match with exec and $R_1, \ldots, R_k$ to conclude relying on the induction hypothesis.
Case 2: since $g'\Downarrow_c$ is discarded by the update function we know that $H = \mathsf{k}_{u_0}(R_H, t_H)$ with $t_H\omega$ a variable, i.e. $t_H\omega \in \mathcal{X}$. In this case, we derive a contradiction with the fact that $R_H\omega\tau = R_H\sigma$ is asap w.r.t. $b_{|\mathsf{Rcv}(q)|}$.

This concludes this sketch of proof. □

Lemma 3.10. *Let* $\mathsf{exec} = (T_0; \emptyset; t_0) \xrightarrow{\ell_1,\ldots,\ell_p}_{\mathcal{T}} (S; \Phi; t)$ *be an execution with input recipes $R_1, \ldots, R_k$ forged by $b_1, \ldots, b_k$ and such that each R_j with $j \in \{1, \ldots, k\}$ is asap w.r.t. b_j and* exec. *Let $K = \mathsf{solved}(\mathsf{sat}(K_{\mathsf{init}}(T_0)))$, and $H \in \mathcal{H}(\mathsf{seed}(T_0))$ with a uniform proof tree π matching with* exec *and $R_1, \ldots, R_k$. Moreover, in case H is of the form $H = \mathsf{k}_{u_0}(R, t)$, we assume that $u_0 = \ell_1, \ldots, \ell_{q-1}$ for some $q \in \mathsf{Rcv}(p)$ and R is asap w.r.t. $b_{|\mathsf{Rcv}(q)|}$ and* exec.

We have that $H \in \mathcal{H}(K)$ with a proof tree π' matching with exec *and $R_1, \ldots, R_k$, and such that* $\mathsf{nodes}(\pi') \subseteq \mathsf{nodes}(\pi)$.

Proof. The proof proceeds by induction on π.
Base case: We have that there is $f \in \mathsf{seed}(T_0)$ of the form $f = (H \Leftarrow)$ and σ grounding for f such that $\mathsf{skl}(f\sigma)$ is in normal form. Moreover, if $H = \mathsf{k}_u(R_0, t_0)$ we know that $vars(t_0) = \emptyset$ (and thus t_0 is not a variable), and has been added to the knowledge base. We have that $f \in \mathsf{solved}(K) = K$. Then f and σ trivially match with exec and $R_1, \ldots, R_k$ because the underlying world of H is either a variable (statement of type 3 or 4) or does not contain any input (statement of type 1 or 2). Finally, $\mathsf{nodes}(\pi') = \mathsf{nodes}(\pi)$ because

we keep the same proof tree.

Induction step: In such a case, we know that there exists $f \in \mathsf{seed}(T_0)$ of the form $H \Leftarrow B_1, \ldots, B_n$ and σ grounding for f such that $\mathsf{skl}(f\sigma)$ is in normal form, and for all $i \in \{1, \ldots, n\}$, $B_i\sigma \in \mathcal{H}(\mathsf{seed}(T_0))$ with a proof tree π_i matching exec and $R_1, \ldots, R_k$ and since π is uniform we have that $\{\pi_1, \ldots, \pi_n\}$ is uniform.

Let us distinguish two cases depending on the type of the statement f:

- if f is a statement of type 1 or 2 then for all $i \in \{1, \ldots, n\}$, there exists $j \in \mathsf{Rcv}(p)$ such that $B_i\sigma = \mathsf{k}_{\ell_1,\ldots,\ell_{j-1}}(R_{|\mathsf{Rcv}(j)|}, u_i)$ for some term u_i. Therefore, by hypothesis we have that $R_{|\mathsf{Rcv}(j)|}$ is asap w.r.t. $b_{|\mathsf{Rcv}(j)|}$ and exec. Moreover π_i (the proof of $B_i\sigma \in \mathcal{H}(\mathsf{seed}(T_0))$) is uniform (as a subtree of π). Our induction hypothesis applies with $B_i\sigma$.

- if f is a statement of type 4 then for all $i \in \{1, \ldots, n\}$, $B_i\sigma = \mathsf{k}_{\ell_1,\ldots,\ell_{j-1}}(S_i, u_i)$ for some $j \in \mathsf{Rcv}(p)$ (by assumption) and term u_i, with S_i a strict subterm of R. Since R is asap w.r.t. $b_{|\mathsf{Rcv}(j)|}$ and exec, we deduce that S_i is asap w.r.t. $b_{|\mathsf{Rcv}(j)|}$ and exec (by Lemma 3.8). We still have that π_i (the proof of $B_i\sigma \in \mathcal{H}(\mathsf{seed}(T_0))$) is uniform (as a subtree of π). Our induction hypothesis applies with $B_i\sigma$.

Therefore in both cases, we have that for all $i \in \{1, \ldots, n\}$, $B_i\sigma \in \mathcal{H}(K)$ with a proof tree π'_i matching with exec and $R_1, \ldots, R_k$, and $\mathsf{nodes}(\pi'_i) \subseteq \mathsf{nodes}(\pi_i)$. Because $\{\pi_1, \ldots, \pi_n\}$ is uniform, we have that $\{\pi'_1, \ldots, \pi'_n\}$ is uniform too.

Since the rule Remove cannot be applied on seed statements, we can distinguish two cases:

1. f is added in the knowledge base, i.e. $f \in K$. Then, we conclude that $H\sigma \in \mathcal{H}(\mathsf{solved}(K))$ thanks to Lemma 3.9 applied on f and σ, because f and σ match with exec and $R_1, \ldots, R_k$ using recipes $\hat{R}_j = X_j$ for all $i \in \{1, \ldots, |w|\}$ where w is the underlying world of H. In addition we have that $\{\pi'_1, \ldots, \pi'_n\}$ is uniform.

2. f is not added in the knowledge base by the update function. Since the underlying run that appears in f is locally closed, we can prove that there exists i_0 such that $X_{i_0}\sigma <_{\mathsf{exec}} R_0\sigma = R$ and thus contradict that R is asap w.r.t. b_m and exec. The full reasoning is formally presented in the proof of Lemma 3.9 (see Case 2) that is available in Appendix A.

   This concludes the proof.

□

We have now all the material needed to establish the completeness of the saturation step. This is formally stated in Theorem 3.4 and the proof mainly relies on the completeness of the seed statements (Theorem 3.2) and the last lemma (Lemma 3.10).

Theorem 3.4. *Let $\mathcal{T}$ be a topology and T_0 be a trace locally closed w.r.t. $\mathcal{X}$. Let $K = \mathsf{solved}(\mathsf{sat}(K_{\mathsf{init}}(T_0)))$ the set of solved statements of the saturated knowledge base. Let $\mathsf{exec} = (T_0; \emptyset; t_0) \xrightarrow{\ell_1,\ldots,\ell_p}_{\mathcal{T}} (S; \Phi; t)$ be an execution with input recipes $R_1, \ldots, R_k$ forged by $b_1, \ldots, b_k$ and such that each R_j with $j \in \{1, \ldots, k\}$ is asap w.r.t. b_j and exec. We have that:*

- $\mathsf{r}_{\ell_1,\ldots,\ell_p} \in \mathcal{H}(K)$ *with a proof tree matching* exec *and* $R_1, \ldots, R_k$*;*
- $\mathsf{k}_{u_0}(R, R\phi\Downarrow) \in \mathcal{H}(K)$ *with a proof tree matching* exec *and* $R_1, \ldots, R_k$ *whenever* $u_0 = \ell_1, \ldots, \ell_{q-1}$ *for some* $q \in \mathsf{Rcv}(p)$ *and* R *is asap w.r.t.* $b_{|\mathsf{Rcv}(q)|}$ *and* exec*.*

Proof. First, we note that exec can be weakened in the untimed semantics such that $\mathsf{exec}' = (T_0; \emptyset) \overset{\ell_1,\ldots,\ell_p}{\rightsquigarrow} (S'; \phi)$ with input recipes $R_1, \ldots, R_k$ and ϕ the untimed counterpart of Φ. Moreover, applying Lemma 3.7 we obtain that $R_1, \ldots, R_k$ are uniform w.r.t. Φ (and ϕ). Therefore we can apply Theorem 3.2 to obtain that $\mathsf{r}_{\ell_1,\ldots,\ell_p} \in \mathcal{H}(\mathsf{seed}(T_0))$ and $\mathsf{k}_{u_0}(R, R\phi\Downarrow) \in \mathcal{H}(\mathsf{seed}(T_0))$ with proof trees that are uniform and matching with exec using $R_1, \ldots, R_k$ as input recipes. We conclude applying Lemma 3.10 to obtain that $\mathsf{r}_{\ell_1,\ldots,\ell_p} \in \mathcal{H}(K)$ and $\mathsf{k}_{u_0}(R, R\phi\Downarrow) \in \mathcal{H}(K)$ with proof trees that are still matching with exec using $R_1, \ldots, R_k$ as input recipes. □

3.4. The full procedure

In this section, we present the full procedure used to verify whether a given trace (locally closed w.r.t. $\mathcal{X}$) is executable until the end. Even if we do not prove its termination in practice, we provide some clues about it. We then prove the soundness and the completeness of the procedure.

3.4.1. Algorithm

The algorithm is presented in Algorithm 1. It takes as input the trace T_0, the initial global time t_0, and the underlying topology under study. As expected, it starts by computing the set K of solved statements applying the saturation step on the seed derived from the trace T. However, for termination purposes, it only considers the set of public names occurring in T, noted $\mathcal{C}_T$, as constants. The algorithm continues by considering all the reach statements in K. It grounds them using a bijective function that maps variables to a specific set of constants and then computes for each input $(a_i, \mathtt{in}(v_i))$, the set $\overline{L_i}$ of recipes that may generate the message v_i. Finally, it generates the set of timing constraints associated to the run under study. In case such a set is satisfiable then the trace will be declared executable. Note that the inverse substitution ρ^{-1} is applied to the recipes before checking the satisfiability of the timing constraints. This allows to replace the constants $c_1, \ldots, c_q$ introduced before by variables: some of them might be instantiated by non negative real numbers (i.e. times) to make the $\mathtt{let}_{\mathtt{time}}$ and $\mathtt{if}_{\mathtt{time}}$ commands executable. Intuitively, these constants represent any atomic data in $\Sigma_0 \cup \mathbb{R}_+$ that are all known by the attackers, and are only used to close the statement and make the search in $\mathcal{H}(K)$ possible.

Algorithm 1 Test for checking whether $(T, \emptyset, t_0)$ is executable in $\mathcal{T}_0$

procedure REACHABILITY$(T, t_0, \mathcal{T}_0)$
  $K = \mathsf{solved}(\mathsf{sat}(\mathsf{seed}(T, \mathcal{C}_T)))$
  for all $\mathsf{r}_{\ell'_1,\ldots,\ell'_n} \Leftarrow \mathsf{k}_{w_1}(X_1, x_1), \ldots, \mathsf{k}_{w_m}(X_n, x_m) \in K$ **do**
    let $c_1, \ldots, c_q$ be specific constants such that
    $\rho : vars(\ell'_1, \ldots, \ell'_n) \to \{c_1, \ldots, c_q\}$ is a bijective function
    for all $i \in \mathsf{Rcv}(n)$ **do**
      if $\ell'_i = (a_i, \mathsf{in}(v_i))$ **then**
        $\overline{L_i} = \left\{ R \mid \mathsf{k}_{\ell'_1\rho \ldots \ell'_{i-1}\rho}(R, v_i\rho) \in \mathcal{H}(K) \right\}$
      end if
    end for
    Let $\{i_1, \ldots, i_p\} = \mathsf{Rcv}(n)$ such that $i_1 < i_2 < \ldots < i_p$
    for all $L_{i_1} \times \ldots \times L_{i_p} \in \overline{L_{i_1}}\rho^{-1} \times \ldots \times \overline{L_{i_p}}\rho^{-1}$ **do**
      Let $\psi = \mathsf{Timing}(\mathcal{T}_0, (T; \emptyset; t_0), L_{i_1} \ldots L_{i_p}, v_{i_1}, \ldots, v_{i_p})$.
      if ψ satisfiable **then**
        return true
      end if
    end for
  end for
  return false
end procedure

Let us explain how the set of timing constraints is generated. Given a trace $T = (a_1, \alpha_1).(a_2, \alpha_2).\ldots.(a_n, \alpha_n)$ locally closed w.r.t. $\mathcal{X}$ and $\mathcal{Z}$, we note $\mathsf{orig}(j)$ the index of the action in the trace T that perform the j^{th} output, i.e. $\mathsf{orig}(j)$ is the minimal k such that $|\mathsf{Snd}(k)| = j$. For sake of simplicity, we also assume the notations presented in Section 3.2.2. The TIMING function takes as input the topology, the initial configuration (made of the trace T and the initial time t_0), the recipes used to feed the inputs as well as the corresponding terms. These terms may still contain variables from $\mathcal{Z}$ since they are not instantiated during the saturation step. It then generates the formula which is the conjunction of the following constraints:

1. $z_1 = t_0$, and $z_i \leq z_{i+1}$ for any $1 \leq i < n$;
2. $t_i \sim t'_i$ for any $i \in \mathsf{Test}(n)$ with $\alpha_i = (\mathtt{if}_{\mathtt{time}}\ t_i \sim t'_i)$;
3. $z'_i = v_i\{x_j \to u_j \mid j \in \mathsf{Rcv}(i)\}\Downarrow$ for any $i \in \mathtt{let}_{\mathtt{time}}(n)$;
4. For any $i \in \mathsf{Rcv}(n)$, we consider the formula:
   - $z_{\mathsf{orig}(j)} + \mathsf{Dist}_{\mathcal{T}_0}(a_{\mathsf{orig}(j)}, a_i) \leq z_i$ if $R_i = \mathsf{w}_j$;
   - otherwise, we consider:

$$\bigvee_{b \in \mathcal{M}_0} \Big(\bigwedge_{\{j \mid \mathsf{w}_j \in vars(R_i)\}} z_{\mathsf{orig}(j)} + \mathsf{Dist}_{\mathcal{T}_0}(a_{\mathsf{orig}}(j), b) \leq z_i - \mathsf{Dist}_{\mathcal{T}_0}(b, a_i) \Big).$$

The constraints given in item 1 ensure that the global time increases along the execution in which the first action is set to start at the global time t_0. We assume that a time variable z_i is associated to each pair (a_i, α_i) in the trace. Constraints given in item 2 correspond to those that already explicitly occurs in the trace through the $\mathtt{if}_{\mathtt{time}}$ commands. Constraints in item 3 ensure that the reduced term in $\mathtt{let}_{\mathtt{time}}$ commands will reduce to a time. Indeed, if not, the formula will not be satisfiable. Finally, constraints in item 4 gather all the timing constraints that must be satisfied to trigger the input rule.

The last step of the algorithm consists in checking whether the formula ψ is satisfiable. This step can be implemented relying on any constraint solving algorithm which features the kind of constraints allowed in the $\mathtt{if}_{\mathtt{time}}$ commands. All the other constraints are simple linear inequalities.

3.4.2. About termination

As previously mentioned, we do not intend to prove the termination of our procedure. Instead, we want it to terminate in practice. To do so, we can note several things.

First, the set of seed statements must be finite. The set $\mathsf{seed}(T)$ as presented in Figure 3.3 is infinite but it was proved in [CCCK16] that we can restrict ourselves to perform saturation using the finite set $\mathsf{seed}(T, \mathcal{C}_T)$ where $\mathcal{C}_T$ contains the public names and the real numbers occurring in the trace T. This is formally stated and proved below.

Lemma 3.11. *Let $\mathcal{C}_T$ be the finite set of public names and real numbers occurring in T, and $\mathcal{C}_{all} = \Sigma_0 \cup \mathbb{R}_+$. We have that:*

$$\mathsf{sat}(K_{\mathsf{init}}(\mathsf{seed}(T, \mathcal{C}_{all}))) = \mathsf{sat}(K_{\mathsf{init}}(\mathsf{seed}(T, \mathcal{C}_T))) \cup \{\mathsf{k}_{\mathsf{y}}(c, c) \Leftarrow \mid \ c \in \mathcal{C}_{all}\}.$$

Proof. We just note that the saturation step does not involve seed statements of type 3 with constants which do not occur in T, i.e. $c \in \mathcal{C}_{all} \setminus \mathcal{C}_T$. This can be formally established by showing that constants in $\mathcal{C}_{all} \setminus \mathcal{C}_T$ will never occur in statements obtained by resolution. □

Another source of non-termination is the saturation step itself. To avoid such issue the update function must discard as many statements as possible. Sadly, but as expected, the more statements are discarded, the more difficult the theoretical development to prove completeness is. The update function we have defined keeps more statements than the one defined in the original procedure. We thus did not try to prove its termination (which already required more than 20 pages in [CCCK16]). Instead we will demonstrate its termination in practice in Chapter 5.

Finally, a source of non-termination is the computation of the sets $\overline{L_i}$ that contain all the recipes R such that $\mathsf{k}_w(R, u_i)$ for the input terms u_i. Fortunately we can focus on asap recipes and rely on a backward search in the saturated knowledge base K to compute the sets. Since all the statements in K are well-formed, the recursive calls will consider strict subterms at each step and the search will eventually terminate.

3.4.3. Soundness and completeness

Theorem 3.5 formally states the soundness (item 1) and the completeness (item 2) of our procedure.

Theorem 3.5. *Let T be a trace that is locally closed w.r.t. $\mathcal{X}$ and $\mathcal{Z}$. Let $\mathcal{T}_0$ be a topology and $\mathcal{K} = (T; \emptyset; t_0)$ be a configuration built on $\mathcal{T}_0$.*

- *if* $\textsc{Reachability}(T, t_0, \mathcal{T}_0)$ *holds then* $(T; \emptyset; t) \rightarrow^*_{\mathcal{T}_0} (\epsilon; \Phi; t)$*;*
- *if* $(T; \emptyset; t_0) \rightarrow^*_{\mathcal{T}_0} (\epsilon; \Phi; t)$ *then* $\textsc{Reachability}(K, t_0, \mathcal{T}_0)$ *holds.*

The soundness part (item 1) is quite straightforward relying on Theorem 3.3 that proves the soundness of the saturation procedure and the correctness of the timing constraints solver. However, the completeness part (item 2) is more involved. Indeed, the completeness of the saturation procedure has only been established w.r.t. executions with asap recipes for inputs. We thus need to first prove that w.l.o.g. we can focus on such executions.

Lemma 3.12. *Let* $\mathsf{exec} = K_0 \xrightarrow{\ell_1,\ldots,\ell_n}_{\mathcal{T}_0} (S; \Phi; t)$ *be an execution. We may assume w.l.o.g. that* exec *involves input recipes $R_1, \ldots, R_k$ forged by agents $b_1, \ldots, b_k$ and R_i is asap w.r.t. b_i and* exec *for each $i \in \{1, \ldots, k\}$.*

Proof. (Sketch of proof)

Due to the definition of asap recipes that builds on two different orders, $<_{\mathsf{exec}}$ and $<^a_{\mathsf{exec}}$ (for $a \in \mathcal{A}_0$) the proof becomes quite technical even if the results seems rather intuitive. The full proof is presented in Appendix A. It proceeds as follows:

1. We consider the execution exec together with recipes $R_1, \ldots, R_k$ that are minimal w.r.t. $<_{\mathsf{exec}}$ or $<^{b_i}_{\mathsf{exec}}$ depending on the honesty of the agent b_i who forges the inputted message.
2. We prove by induction on the length of the trace that $R_1, \ldots, R_k$ are asap recipes w.r.t. $b_1, \ldots, b_k$. To do so we distinguish several cases depending on the action that is going to be executed, the form of the recipe and the status of the agent forging a message (in case of an input). We then conclude relying on the induction hypothesis or the minimality of the recipes $R_1, \ldots, R_k$ we consider.

□

Proof of Theorem 3.5

Proof. We can now prove the soundness and the completeness of our algorithm.

Soundness (item 1): We assume that our algorithm returns true when considering $\ell'_1 \ldots \ell'_n$ and recipes $L_{i_1}, \ldots, L_{i_n}$. Actually, soundness of our saturation procedure (Theorem 3.3) gives us that $(T; \emptyset) \xrightarrow{\ell'_1 \rho \ldots \ell'_n \rho} (\epsilon; \phi)$ (untimed semantics) using recipes $L_{i_1}, \ldots, L_{i_n}$. The formula ψ gathers all the timing constraints that have to be satisfied, in particular those

to ensure that all the material needed to perform the computation L_{i_j} is available in time. The satisfiability of ψ gives us a valuation $\rho' : \mathcal{X} \cup \mathcal{Z} \cup \mathcal{Y} \to \mathbb{R}_+$ that satisfies all the timing constraints. Let us consider the substitution $\rho' : \{c_1, \ldots, c_q\} \to \mathbb{R}_+$ which match the given valuation. Note that since $c_1, \ldots, c_q$ do not appear in T we can apply ρ' along the execution to obtain that:

$$(T; \emptyset) \xrightarrow{\ell'_1\rho' \ldots \ell'_n\rho'}_{\mathcal{T}_0} (\epsilon; \phi\rho')$$

3

Finally, since ρ' has been defined such that it matches the valuation, we have:

$$(T; \emptyset; t_0) \xrightarrow{\ell'_1\rho' \ldots \ell'_n\rho'}_{\mathcal{T}_0} (\epsilon; \phi\rho'; t).$$

This concludes the proof for the first item.

Completeness (item 2): We assume that $(T; \emptyset; t_0)$ is executable in $\mathcal{T}_0$. It follows that there exists an execution exec such that:

$$\mathsf{exec} = (T; \emptyset; t_0) \xrightarrow{\ell_1}_{\mathcal{T}_0} (T_1; \Phi_1; t_1) \xrightarrow{\ell_2}_{\mathcal{T}_0} \ldots \xrightarrow{\ell_n}_{\mathcal{T}_0} (T_n; \Phi_n; t_n) = (\epsilon; \Phi; t)$$

First, thanks to Lemma 3.12, we can assume w.l.o.g. that there exist recipes $R_1, \ldots, R_k$ forged by agents $b_1, \ldots, b_k$ that can be used as inputs in exec and such that R_j is asap w.r.t. b_j and exec for any $j \in \{1, \ldots, k\}$. Therefore, applying Theorem 3.4 we obtain that $\mathsf{r}_{\ell_1,\ldots,\ell_n} \in \mathcal{H}(\mathsf{solved}(\mathsf{sat}(\mathcal{K}_{\mathsf{init}}(T))))$ with a proof tree π matching with exec using $R_1, \ldots, R_k$ as input recipes. Applying Lemma 3.11 we have that $\mathsf{r}_{\ell_1,\ldots,\ell_n} \in \mathcal{H}(\mathsf{solved}(\mathsf{sat}(\mathcal{K}_{\mathsf{init}}(T, \mathcal{C}_T))))$ using the same proof tree.

We deduce that π ends with a solved statement

$$h = \mathsf{r}_{\ell'_1,\ldots,\ell'_n} \Leftarrow \mathsf{k}_{w_1}(Y_1, y_1), \ldots, \mathsf{k}_{w_m}(Y_m, y_m) \in K$$

instantiated with a substitution τ such that $(\ell'_1, \ldots, \ell'_n)\tau = (\ell_1, \ldots, \ell_n)$. Moreover, because π matches with exec with input recipes $R_1, \ldots, R_k$, there exist recipes $\hat{R}_1, \ldots, \hat{R}_k$ such that:

- $\hat{R}_j(\{Y_i \to y_i \mid 1 \leq i \leq m\} \uplus \phi(\ell'_1, \ldots, \ell'_n))\Downarrow = v'_j$ for $j \in \{1, \ldots, k\}$.
- $\hat{R}_j\tau = R_j$ for $j \in \{1, \ldots, k\}$

where $v'_1, \ldots, v'_k$ are the terms occurring in input in $\ell'_1, \ldots, \ell'_n$.

Note that for all grounding substitutions $\rho : \mathcal{X} \cup \mathcal{Z} \cup \mathcal{Y} \to \mathbb{R}_+$ such that $Y_i\rho = y_i\rho$ for $i \in \{1, \ldots, m\}$, we have that:

$$\mathsf{exec}_\rho = (T; \emptyset) \xrightarrow{\ell'_1\rho} \ldots \xrightarrow{\ell'_n\rho} (\epsilon; \varphi_\rho)$$

using $\hat{R}_1\rho, \ldots, \hat{R}_k\rho$ as input recipes.

Moreover, the timing constraints induced by $\hat{R}_j\rho$ are less restrictive than those induced by R_j since $\hat{R}_i\tau = R_j$. W.l.o.g. we can thus consider the substitution ρ that makes

all the $\mathtt{let}_{\mathtt{time}}$ and $\mathtt{if}_{\mathtt{time}}$ commands executable (note that it exists since the commands are executable in exec) and we have that:

$$\mathsf{exec}_{\rho}^{\mathsf{timed}} = (T;\emptyset;t_0) \xrightarrow{\ell'_1\rho}_{\mathcal{T}_0} \ldots \xrightarrow{\ell'_n\rho}_{\mathcal{T}_0} (\epsilon;\varphi_\rho;t)$$

using $\hat{R}_1\rho,\ldots,\hat{R}_k\rho$ as input recipes.

The statement h is the one that will be considered in our procedure. To conclude we have to show that our procedure will consider recipes that can fill the inputs and satisfy the timing constraints. Applying Lemma 3.12 we know that there exist recipes $\hat{S}_1,\ldots,\hat{S}_k$ and agents $b'_1,\ldots,b'_k$ such that for each $j \in \{1,\ldots,k\}$, $\hat{S}_j$ is asap w.r.t. b'_j and $\mathsf{exec}_{\rho}^{\mathsf{timed}}$ and such that $\mathsf{exec}_{\rho}^{\mathsf{timed}}$ with input recipes $\hat{S}_1,\ldots,\hat{S}_k$ forged by $b'_1,\ldots,b'_k$ is an execution. Applying Theorem 3.4 (item 2), we obtain that $\mathsf{k}_{\ell'_1\ldots,\ell'_{j-1}}(\hat{S}_j, v_j\rho) \in \mathcal{H}(\mathsf{solved}(\mathsf{sat}(\mathcal{K}_{\mathsf{init}})(T)))$ with a proof tree π. We can now replace all the occurrences of $\mathsf{k}_{\mathsf{y}}(t,t)$ with $t \in \mathbb{R}_+$ by $\mathsf{k}_{\mathsf{y}}(c,c)$ with $c \in \{c_1,\ldots,c_q\}$ such that the new proof tree deduces $v_j\rho'$ where ρ' is the bijective function considered in the algorithm. This proof tree provides the recipe R'_j that will be considered in our algorithm. Finally, the set of constraints derived from $\hat{S}_j$ is the same as the one derived from R'_j and thus the formula ϕ considered in the algorithm will be satisfiable. This concludes the proof. □

3.5. Conclusion

In this chapter we presents a procedure that addresses the issue of verifying the executability of a timed symbolic trace. We have managed to prove its correctness, i.e. its soundness and completeness, in a generic model that is very close to the one described in Chapter 2. The unique restriction is about the equational theory that must satisfy the finite variant property. Removing this assumption, there is no hope of termination in practice.

This chapter have presented the theoretical foundations of the global procedure that will be presented in Chapter 5 to analyse distance-bounding protocols w.r.t. reachability properties considering a bounded number of sessions.. In Chapter 5, we will first explain how a protocol can be represented by a finite set of symbolic traces, then we will discuss about the implementation of the procedure, and then we will apply it to the analysis of case studies. On this occasion we will discuss about its termination and efficiency in practice: these two aspects are closely related to the update function involved during the saturation step.

As reminded above, the main theoretical limitation of our procedure is the finite variant property assumption. As defined in Definition 3.1, it prevents us to consider associative and commutative (AC) symbols of function like the exclusive-OR operator. The finite variant property has been defined for AC theories [CLD05], and the Akiss procedure has been recently extended to handle the exclusive-OR [BDGK17]. We thus hope that this limitation could be removed in future works when looking at our procedure too. However, to handle this new operator, the update function of the Akiss procedure has been modified so that more statements are discarded (typically, it discards a deduction

statement as soon as the knowledge base already deduces the same term modulo AC). Hence, it will complicate the proof of completeness when considering time and locations.

An unbounded number of sessions 4

In this chapter, we address the verification of distance-bounding protocols considering an unbounded number of sessions. We establish several reduction results that remove specific sources of unboundedness, and allow one to leverage existing tools for an automatic verification (see Chapter 5). The first result reduces the number of topologies that must be considered from infinitely many to only one per class of attacks. The second one applies for terrorist frauds and defines a most general semi-dishonest prover which subsumes all the possible collusion behaviours.

Contents

4.1. Hypotheses

Verifying protocols in the context of an unbounded number of sessions is well-known to be much more complex than in a bounded setting. In order to establish our reduction results we therefore make some restrictions on our timed model. In particular we will restrict the use of the timing constraints.

4.1.1. Messages without times

The aim of this chapter is to present reduction results that enable the use of existing verification tools to analyse distance-bounding protocols. As a first observation, one may note that these protocols strongly rely on time to estimate the proximity of the agents, but they do not involve times in messages. Indeed, the proximity check is entirely performed by the verifier who does not share timing information with the prover. In this chapter, we therefore prevent from considering time in messages. Note that an agent is still able to perform time checks on his own side. An immediate consequence of this restriction is that the $\mathtt{let}_{\mathtt{time}}$ command becomes useless. In this chapter we thus consider protocols made of roles without it.

In few protocols, like MasterCard-RRP [EMV16], the time-bound for the proximity check is received by the verifier from a message sent by the prover. Indeed, regarding payment protocol, the time-bound may depend on the computation speed of the card (i.e. the prover). In this case, in order to overcome the restriction we just introduced, we will model the time-bound by a public symbol of function $\mathtt{timebound}(x)$, where x is an agent name. We add this value in the $\mathtt{end}$ event and simply verify the standard correspondence property: whenever the command $\mathtt{end}(v_0, p_0, u)$ is reached then the equality $u = \mathtt{checkBound}(\mathtt{timebound}(\mathtt{p}_0))$ holds. This modelling will be further detailed in Chapter 5 when looking at these protocols.

4.1.2. A unique clock per process

Distance-bounding protocols always follow the same structure: they mainly rely on the timing of a challenge/response exchange. To perform such a computation, agents just need to have a clock that they can reset just before sending the challenge, stop immediately after they receive the response, and compare to a given threshold. It corresponds to sub-processes of the form: $\mathtt{out}^{z_1}(u).\mathtt{in}^{z_2}(x).\mathtt{if}_{\mathtt{time}}\ z_2 - z_1 < t_g\ \mathtt{then}\ P$.

To simplify our calculus, we replace the $\mathtt{if}_{\mathtt{time}}$ command by a $\mathtt{reset}$ and a guarded input $\mathtt{in}^{<t_g}(x)$. More formally, a sub-process $\mathtt{out}^{z_1}(u).\mathtt{in}^{z_2}(x).\mathtt{if}_{\mathtt{time}}\ z_2 - z_1 < t_g\ \mathtt{then}\ P$ is translated into a sub-process $\mathtt{reset}.\mathtt{out}(u).\mathtt{in}^{<t_g}(x).P$. In a configuration, the extended processes are then annotated by the value of the local clock associated to it. We note $\lfloor P \rfloor_a^{t_a}$ to model a process P executed by an agent a with a current clock value $t_a \in \mathbb{R}_+$.

The semantic rules of these two commands are presented in Figure 4.1. The RESET rule simply sets the local clock to 0 and the IN-G rule is the IN rule with the extra constraint $t_a < t_g$, meaning that the local clock is less than the guard when triggering the input. The semantic rules of the old commands do not involve the local clock and can thus be trivially adapted. However, one may note that the local clocks of each agent must be increased when the global time elapses. This comment implies to slightly modify

$$\text{TIM} \quad (\mathcal{P};\Phi;t) \longrightarrow_{\mathcal{T}_0} (\mathsf{Shift}(\mathcal{P},\delta);\Phi;t+\delta) \qquad \text{with } \delta \geq 0$$

$$\text{RESET} \quad (\lfloor \mathtt{reset}.P \rfloor_a^{t_a} \uplus \mathcal{P};\Phi;t) \xrightarrow{a,\tau}_{\mathcal{T}_0} (\lfloor P \rfloor_a^{0} \uplus \mathcal{P};\Phi;t)$$

$$\text{IN-G} \quad (\lfloor \mathtt{in}^{<t_g}(x).P \rfloor_a^{t_a} \uplus \mathcal{P};\Phi;t) \xrightarrow{a,\mathtt{in}^{<t_g}(u)}_{\mathcal{T}_0} (\lfloor P\{x \mapsto u\} \rfloor_a^{t_a} \uplus \mathcal{P};\Phi;t)$$

when $t_a < t_g$ and there exist $b \in \mathcal{A}_0$ and $t_b \in \mathbb{R}_+$ such that $t_b \leq t - \mathsf{Dist}_{\mathcal{T}_0}(b,a)$ and:

- if $b \in \mathcal{A}_0 \setminus \mathcal{M}_0$ then $u \in img(\lfloor \Phi \rfloor_b^{t_b})$;
- if $b \in \mathcal{M}_0$ then $u = R\Phi\downarrow$ for some recipe $R \in \mathcal{T}(\sigma_{\mathsf{pub}}, \mathcal{W})$ such that for all $\mathsf{w} \in vars(R)$ there exists $c \in \mathcal{A}_0$ such that $\mathsf{w} \in dom(\lfloor \Phi \rfloor_c^{t_b - \mathsf{Dist}_{\mathcal{T}_0}(c,b)})$.

Figure 4.1: Semantics of our calculus using local clocks

the TIM rule. We consider the following notation to increase all the clocks in a raw by a given non-negative delay δ:

$$\mathsf{Shift}(\mathcal{P},\delta) = \{ \lfloor P \rfloor_a^{t_a+\delta} \mid \lfloor P \rfloor_a^{t_a} \in \mathcal{P} \}.$$

In the following of this chapter, we consider distance-bounding protocols that match this restriction about clocks, i.e. $\mathtt{if}_{\mathtt{time}}$ commands only appear in sub-processes as described. By abuse of notation we will always describe the roles and the processes through this new syntax and execute them through the new semantics. A role is thus a parametrised process generated by the following grammar:

$$\begin{array}{rcl} P & := & 0 \\ & | & \mathtt{new}\ n.P \\ & | & \mathtt{in}(x).P \\ & | & \mathtt{out}(u).P \\ & | & \mathtt{let}_{\mathtt{mess}}\ x = v\ \mathtt{in}\ P \\ & | & \mathtt{reset}.\mathtt{out}(u').\mathtt{in}^{<t_g}(x).P \\ & | & \mathsf{end}(u_1,u_2) \end{array}$$

where $x \in \mathcal{X}$, $n \in \mathcal{N}$, $u, u', u_1, u_2 \in \mathcal{T}(\Sigma_c^+, \mathcal{X} \cup \mathcal{N})$, $v \in \mathcal{T}(\Sigma^+, \mathcal{X} \cup \mathcal{N})$ and $t_g \in \mathbb{R}_+$.

Remark 4.1. *Note that in Figure 4.1, the recipes that malicious agents are able to use to forge a message no longer involve non negative real numbers in comparison to the timed recipes involved in the semantics presented in Chapter 2. This restriction is necessary to prevent messages from containing real numbers but does not restrict the expressiveness of the model itself. Indeed, since an agent only compares times that are generated by himself, times that might be introduced by an attacker can be seen as constants and thus replaced by elements in* Σ_0.

Example 4.1. *The SPADE protocol presented in Example 2.5 matches the constraints about clocks. The prover role is let unchanged but the verifier role is now described by:*

$$
\begin{array}{rcl}
\mathtt{V}(\mathsf{x}_0,\mathsf{x}_1) & := & \mathtt{in}(x_1).\\
& & \mathtt{let}_{\mathtt{mess}}\ y_1 = \mathsf{adec}(x_1,\mathsf{sk}(\mathsf{x}_0))\ \mathtt{in}\\
& & \mathtt{let}_{\mathtt{mess}}\ y^1_{\mathsf{check}} = \mathsf{eq}(\mathsf{proj}_1(y_1),\mathsf{check}(\mathsf{proj}_2(y_1),\mathsf{spk}(\mathsf{x}_1)))\ \mathtt{in}\\
& & \mathtt{new}\ m_V.\ \mathtt{new}\ n_V.\\
& & \mathtt{out}(\langle m_V,n_V\rangle).\\
& & \mathtt{new}\ c.\\
& & \mathtt{reset}.\mathtt{out}(c).\mathtt{in}^{<2\times t_0}(x_2).\\
& & \mathtt{in}(x_3).\\
& & \mathtt{let}_{\mathtt{mess}}\ x^0_H = \mathsf{prf}(\langle\mathsf{proj}_1(y_1),n_V\rangle)\ \mathtt{in}\\
& & \mathtt{let}_{\mathtt{mess}}\ x^1_H = \mathsf{proj}_1(y_1)\oplus m_V\oplus x^0_H\ \mathtt{in}\\
& & \mathtt{let}_{\mathtt{mess}}\ y^2_{\mathsf{check}} = \mathsf{eq}(x_2,\mathsf{answer}(c,x^0_H,x^1_H))\ \mathtt{in}\\
& & \mathtt{let}_{\mathtt{mess}}\ y^3_{\mathsf{check}} = \mathsf{eq}(x_3,\mathsf{prf}(\langle\mathsf{proj}_1(y_1),n_V,m_V,c,x_2\rangle))\ \mathtt{in}\ 0
\end{array}
$$

In the following, we will never take care of the initial values of the local clocks. Indeed, by construction we have that each guarded input is preceded by a `reset` command that sets to 0 the corresponding local clock. Its initial value is thus meaningless. The following proposition allows us to assume w.l.o.g. that all the clocks, i.e. the global and the local ones, are set to 0 in initial configurations. This will be useful to re-time executions given in untimed semantics without taking care of the initial times.

Proposition 4.1. *Let $\mathcal{T}_0 = (\mathcal{A}_0,\mathcal{M}_0,\mathsf{Loc},v_0,p_0)$ be a topology and $\mathcal{K} = (\mathcal{P};\Phi;t)$ be a configuration such that any guarded input in $\mathcal{P}$ is preceded by a reset. Let $\mathcal{K}' = (\mathcal{P}';\Phi\uplus\Psi';t')$ be a configuration such that $\mathcal{K}\xrightarrow{\mathsf{tr}}_{\mathcal{T}_0}\mathcal{K}'$. Let $\mathcal{K}_0 = (\mathcal{P}_0;\Phi;0)$ with $\mathcal{P}_0$ equal to $\mathcal{P}$ in which all the local clocks are set to 0.*

We have that $\mathcal{K}_0\xrightarrow{\mathsf{tr}}_{\mathcal{T}_0}\mathcal{K}'_0 = (\mathcal{P}'_0;\Phi\uplus\Psi'_0;t'_0)$ for some $\mathcal{K}'_0$ such that:

(1) $\{(P,a)\mid \lfloor P\rfloor^t_a\in\mathcal{P}'$ *for some* $t\} = \{(P,a)\mid \lfloor P\rfloor^t_a\in\mathcal{P}'_0$ *for some* $t\}$*;*

(2) $\{(\mathsf{w},a,u)\mid \mathsf{w}\xrightarrow{a,t}u\in\Psi'$ *for some* $t\} = \{(\mathsf{w},a,u)\mid \mathsf{w}\xrightarrow{a,t}u\in\Psi'_0$ *for some* $t\}$*.*

Proof. This proposition is proved establishing a strong relation between the two executions. We prove the result together with the three properties below by induction on the length of the derivation $\mathcal{K}\xrightarrow{\mathsf{tr}}_{\mathcal{T}_0}\mathcal{K}'$ where $\delta = \max(\{\mathsf{Dist}_{\mathcal{T}}(a,b)\mid a,b\in\mathcal{A}_0\}\cup\{t\})$.

(i) if $\lfloor P\rfloor^{t_a}_a\in\mathcal{P}'$ and P contains a guarded input that is not preceded by a reset then $\lfloor P\rfloor^{t_a}_a\in\mathcal{P}'_0$ (with the same value for t_a);

(ii) $t'_0 = t'-t+\delta$, i.e., the global time is just shifted by a constant;

(iii) $\Psi'_0 = \mathsf{Shift}(\Psi',\delta-t)$ with $\mathsf{Shift}(\Psi',\delta-t) = \{\mathsf{w}\xrightarrow{a,t_a+\delta-t}u\mid \mathsf{w}\xrightarrow{a,t_a}u\in\Psi'\}$, i.e., the frames are the same, up to a time shift.

Base case: In such a case, we have that $\mathcal{K}' = \mathcal{K}$, and thus $t' = t$, $\mathcal{P}' = \mathcal{P}$, $\Phi' = \Phi$, and $\Psi' = \emptyset$. Let $\mathcal{K}'_0 = (\mathsf{Shift}(\mathcal{P}, \delta); \Phi; \delta)$. We have that $\mathcal{K}_0 \rightarrow_{\mathcal{T}_0} \mathcal{K}'_0$ using the TIM rule. Note that item (i) is satisfied since by hypothesis in $\mathcal{P}'$ $(= \mathcal{P})$, any guarded input is preceded by a reset. Regarding (ii), we have that $t'_0 = \delta = t' - t + \delta$ since $t' = t$. Since $\Psi' = \emptyset$, item (iii) is also satisfied. Then items (1) and (2) are trivially satisfied.

Induction step: In such a case, we have that $\mathcal{K} \xrightarrow{\mathsf{tr}}_{\mathcal{T}_0} \mathcal{K}'' \xrightarrow{a,\alpha}_{\mathcal{T}_0} \mathcal{K}'$ with $\mathcal{K}'' = (\mathcal{P}''; \Phi''; t'')$ and $\Phi'' = \Phi \uplus \Psi''$. By induction hypothesis, we know that there exists $\mathcal{K}''_0 = (\mathcal{P}''_0; \Phi \uplus \Psi''_0; t''_0)$ such that $\mathcal{K}_0 \xrightarrow{\mathsf{tr}}_{\mathcal{T}_0} \mathcal{K}''_0$ with:

1. $\{(P, a) \mid \lfloor P \rfloor_a^t \in \mathcal{P}'' \text{ for some } t\} = \{(P, a) \mid \lfloor P \rfloor_a^t \in \mathcal{P}''_0 \text{ for some } t\}$;
2. $\{(\mathsf{w}, a, u) \mid \mathsf{w} \xrightarrow{a,t} u \in \Psi'' \text{ for some } t\} = \{(\mathsf{w}, a, u) \mid \mathsf{w} \xrightarrow{a,t} u \in \Psi''_0 \text{ for some } t\}$.

We also have that:

(i) if $\lfloor P \rfloor_a^{t_a} \in \mathcal{P}''$ and P contains a guarded input that is not preceded by a reset then $\lfloor P \rfloor_a^{t_a} \in \mathcal{P}''_0$;

(ii) $t''_0 = t'' - t + \delta$;

(iii) $\Psi''_0 = \mathsf{Shift}(\Psi'', \delta - t)$.

We consider the rule involved in $\mathcal{K}'' \xrightarrow{a,\alpha}_{\mathcal{T}_0} \mathcal{K}'$ and we show that the same rule can be applied on $\mathcal{K}''_0$, and allows one to get $\mathcal{K}'_0$ with the five properties stated above.

Case TIM rule. In such a case, we have that $\mathcal{K}' = (\mathcal{P}''; \Phi''; t'' + \delta_0)$ for some δ_0, and we apply the same rule with the delay δ_0 on $\mathcal{K}''_0$. We obtain $\mathcal{K}'_0 = (\mathcal{P}''_0; \Phi \uplus \Psi''_0; t''_0 + \delta_0)$, and we easily check that all the properties are satisfied.

Case OUT rule. In such a case, $\mathcal{K}' = (\mathcal{P}'; \Phi \uplus \Psi'' \uplus \{\mathsf{w} \xrightarrow{a,t''} u\}; t'')$ (for some $\mathcal{P}'$, w, a, and u). Relying on item 1, we apply the same rule on $\mathcal{K}''_0$, and obtain $\mathcal{K}'_0 = (\mathcal{P}'_0; \Phi \uplus \Psi''_0 \uplus \{\mathsf{w} \xrightarrow{a,t''_0} u\}; t''_0)$ (for some $\mathcal{P}'_0$). We have that items 1 and 2 are clearly satisfied as well as item (i). Now, regarding item (ii), we have that $t'_0 = t''_0$ and $t' = t''$. Therefore, we conclude that $t'_0 = t' - t + \delta$ thanks to our induction hypothesis. Now, to establish that $\Psi'_0 = \mathsf{Shift}(\Psi', \delta - t)$, relying on our induction hypothesis, it only remains to show that $t''_0 = t'' + (\delta - t)$. This is item (ii).

Case LET and NEW rules. In such a case, $\mathcal{K}' = (\mathcal{P}'; \Phi \uplus \Psi''; t'')$ (for some $\mathcal{P}'$), and we apply that same rule on $\mathcal{K}''_0$, and obtain $\mathcal{K}'_0 = (\mathcal{P}'_0; \Phi \uplus \Psi''_0; t''_0)$. We conclude easily (for each property) relying on our induction hypothesis.

Case RST rule. In such a case, $\mathcal{K}' = (\mathcal{P}'; \Phi \uplus \Psi''; t'')$ with $\mathcal{P}' = \{\lfloor P \rfloor_a^0\} \cup \mathcal{Q}'$ for some P, a, t^a and $\mathcal{Q}'$, and $\mathcal{P}'' = \{\lfloor \mathtt{reset}.P \rfloor_a^{t^a}\} \uplus \mathcal{Q}'$. We apply the same rule on $\mathcal{K}''_0$, and obtain $\mathcal{K}'_0 = (\mathcal{P}'_0; \Phi \cup \Psi''_0; t''_0)$. Relying on our induction hypothesis, we easily obtain the fact that items 1 and 2 are satisfied. Regarding item (i), we conclude using our induction hypothesis for processes in $\mathcal{Q}'$, and the property is satisfied for $\lfloor P \rfloor_a^0$. Regarding

items (ii) and (iii), since the global time and the frame have not evolved, we conclude relying on our induction hypothesis.

Case IN rule. In such a case, $\mathcal{K}' = (\mathcal{P}'; \Phi \uplus \Psi''; t'')$ (for some $\mathcal{P}'$), and we apply the same rule on $\mathcal{K}_0''$ to get $\mathcal{K}_0' = (\mathcal{P}_0'; \Phi \uplus \Psi_0''; t_0'')$. The difficult part is to show that the rule can indeed be applied on $\mathcal{K}_0''$. By hypothesis, we know that $\mathcal{K}'' \xrightarrow{a,\mathtt{in}^\star(u)}_{\mathcal{T}_0} \mathcal{K}'$, and thus $\mathcal{P}'' = \lfloor \mathtt{in}^\star(x).P \rfloor_a^{t^a} \uplus \mathcal{Q}$ for some x, a, t^a and $\mathcal{Q}$, and we know that there exists $b \in \mathcal{A}_0$ and $t^b \in \mathbb{R}_+$ such that $t^b < t'' - \mathsf{Dist}_{\mathcal{T}_0}(b, a)$ and:

- if $b \in \mathcal{A}_0 \setminus \mathcal{M}_0$ then $u \in img(\lfloor \Phi'' \rfloor_b^{t^b})$;
- if $b \in \mathcal{M}_0$ then $u = R\Phi\downarrow$ for some recipe R such that for all $\mathsf{w} \in vars(R)$ there exists $c \in \mathcal{A}_0$ such that $\mathsf{w} \in dom(\lfloor \Phi'' \rfloor_c^{t^b - \mathsf{Dist}_{\mathcal{T}_0}(c,b)})$.

Moreover, in case $\star$ is $< t_g$ for some t_g, we know in addition that $t^a < t_g$.

We first assume that $\mathtt{in}^\star(u)$ is a simple input (not a guarded one). We show that we can apply on $\mathcal{K}_0''$ the same rule using the same recipe R. The message will be sent by the same agent $b \in \mathcal{A}_0$. The time t_0^b at which the message is sent is $t_0^b = t^b + (\delta - t)$. Note that $t_0^b \geq t^b$ since $\delta - t \geq 0$, and $img(\lfloor \Psi'' \rfloor_b^{t^b}) = img(\lfloor \Psi_0'' \rfloor_b^{t_0^b})$ (and $dom(\lfloor \Psi'' \rfloor_b^{t^b}) = dom(\lfloor \Psi_0'' \rfloor_b^{t_0^b})$ as well) since $\Psi_0'' = \mathsf{Shift}(\Psi'', \delta - t)$, and $t_0^b = t^b + (\delta - t)$. We distinguish two cases:

- if $b \in \mathcal{A}_0 \setminus \mathcal{M}_0$ then we know that

$$\begin{array}{rcl} u & \in & img(\lfloor \Phi'' \rfloor_b^{t^b}) \\ & = & img(\lfloor \Phi \rfloor_b^{t^b}) \cup img(\lfloor \Psi'' \rfloor_b^{t^b}) \\ & \subseteq & img(\lfloor \Phi \rfloor_b^{t_0^b}) \cup img(\lfloor \Psi_0'' \rfloor_b^{t_0^b}) \end{array}$$

- if $b \in \mathcal{M}_0$ then we consider $\mathsf{w} \in vars(R)$. We have that:

$$\begin{array}{rcl} w & \in & dom(\lfloor \Phi'' \rfloor_c^{t^b - \mathsf{Dist}_{\mathcal{T}_0}(c,b)}) \\ & = & dom(\lfloor \Phi \rfloor_c^{t^b - \mathsf{Dist}_{\mathcal{T}_0}(c,b)}) \cup dom(\lfloor \Psi'' \rfloor_c^{t^b - \mathsf{Dist}_{\mathcal{T}_0}(c,b)}) \\ & \subseteq & dom(\lfloor \Phi \rfloor_c^{t_0^b - \mathsf{Dist}_{\mathcal{T}_0}(c,b)}) \cup dom(\lfloor \Psi_0'' \rfloor_c^{t_0^b - \mathsf{Dist}_{\mathcal{T}_0}(c,b)}) \end{array}$$

Thus, in both cases, this allows us to conclude. Now, in case $\star$ is $< t_g$, by hypothesis we know that $t^a < t_g$, and thanks to item (i), we know that $\mathcal{P}_0'' = \lfloor \mathtt{in}^\star(x).P \rfloor_a^{t^a} \uplus \mathcal{Q}_0$. This allows us to conclude.

□

4.1.3. Executable processes

The last hypothesis that we will consider is the executability of the roles. Note that this is not a strong assumption. Indeed, our model allows to define roles, e.g. $\mathtt{V}(\mathsf{x}_0, \mathsf{x}_1)$ or $\mathtt{P}(\mathsf{x}_0, \mathsf{x}_1)$, that contain private data of agents different from x_0. To prevent these unrealistic behaviours, we rely on the template $\mathcal{I}_0$ used to derive the initial knowledge of each agent.

Definition 4.1. *Given a set $\mathcal{I}_0 = \{u_1, \ldots, u_k\}$ of terms, we say that a parametrised process P is $\mathcal{I}_0$-executable if for any term u (resp. v) occurring in an* `out` *(resp. a* $\mathtt{let}_{\mathtt{mess}}$*) command, there exists a term $R \in \mathcal{T}(\Sigma^+_{\mathsf{pub}}, \{\mathsf{w}_1, \ldots, \mathsf{w}_k\} \cup bn(P) \cup bv(P))$ such that $u =_{\mathtt{E}} R\sigma\downarrow$ (resp. $v\downarrow =_{\mathtt{E}} R\sigma\downarrow$) where $\sigma = \{\mathsf{w}_1 \to u_1, \ldots, \mathsf{w}_k \to u_k\}$.*

We say that a distance-bounding protocol $\mathcal{P}_{\mathsf{db}} = (\mathtt{V}(\mathsf{x}_0, \mathsf{x}_1), \mathtt{P}(\mathsf{x}_0, \mathsf{x}_1))$ is $\mathcal{I}_0$-executable if $\mathtt{V}(\mathsf{x}_0, \mathsf{x}_1)$ and $\mathtt{P}(\mathsf{x}_0, \mathsf{x}_1)$ are $\mathcal{I}_0$-executable.

Example 4.2. *Going back to Example 2.8 and considering $\mathcal{I}_0 = \{\mathsf{x}_1, \mathsf{sk}(\mathsf{x}_0), \mathsf{ssk}(\mathsf{x}_0)\}$, we can show that the two roles represented in Example 4.1 are $\mathcal{I}_0$-executable.*

According to Definition 4.1, we have $\sigma = \{\mathsf{w}_1 \to \mathsf{x}_1, \mathsf{w}_2 \to \mathsf{sk}(\mathsf{x}_0), \mathsf{w}_3 \to \mathsf{ssk}(\mathsf{x}_0)\}$, and for instance, the first output of $\mathtt{P}(\mathsf{x}_0, \mathsf{x}_1)$, i.e. $u = \mathsf{aenc}(\langle n_P, \mathsf{sign}(n_P, \mathsf{ssk}(\mathsf{x}_0))\rangle, \mathsf{pk}(\mathsf{x}_1))$ is such that $u =_{\mathtt{E}} R\sigma\downarrow$ with $R = \mathsf{aenc}(\langle n_P, \mathsf{sign}(n_P, \mathsf{w}_3)\rangle, \mathsf{pk}(\mathsf{w}_1))$. The SPADE protocol, as described by $(\mathtt{V}(\mathsf{x}_0, \mathsf{x}_1), \mathtt{P}(\mathsf{x}_0, \mathsf{x}_1))$ is thus $\mathcal{I}_0$-executable.

4

4.2. Reducing the topologies

When analysing a protocol w.r.t. distance hijacking, mafia or terrorist fraud, an infinite number of topologies must be considered: any topology in $\mathcal{C}^{t_0}_{\mathsf{DH}}$ and $\mathcal{C}^{t_0}_{\mathsf{MF}}$. Remember that the scenarios that are considered for the re-authentication step in a terrorist fraud is a kind of mafia fraud (with an initial knowledge extended with the data of the first step). The set of topologies that should be analysed when looking for terrorist fraud is thus the same, i.e. $\mathcal{C}^{t_0}_{\mathsf{MF}}$ as for mafia fraud.

In this section, we establish two reduction results, i.e. for each set of topologies, to get rid of this source of unboundedness. More precisely we prove that, $\mathcal{C}^{t_0}_{\mathsf{DH}}$ and $\mathcal{C}^{t_0}_{\mathsf{MF}}$ can be represented by a unique, and rather simple, topology on which the analyses can focus on to look for distance hijacking, mafia, and terrorist fraud.

4.2.1. Mafia and terrorist frauds

Figure 4.2: Topologies $\mathcal{T}^{t_0}_{\mathsf{MF}}$ where t_0 is the proximity threshold.

The same set of topologies, $\mathcal{C}^{t_0}_{\mathsf{MF}}$, must be considered when verifying mafia or terrorist fraud resistance. In the remainder of this section, we will prove that it can be represented by a unique and rather simple topology which involves only four agents, located at only two different locations (see Figure 4.2). In the remainder of this section, we prove that if a trace exists considering an arbitrary topology $\mathcal{T} \in \mathcal{C}^{t_0}_{\mathsf{MF}}$ then, a similar trace can be executed in $\mathcal{T}^{t_0}_{\mathsf{MF}}$. This proof proceeds in four stages:

1. we transform any honest agent but v_0 and p_0 in dishonest ones (Lemma 4.1);
2. relying on the executability hypothesis, we simplify the initial configuration getting rid of processes executed by malicious agents (Lemma 4.2);

3. we reduce the number of attackers by placing them ideally (Lemma 4.3);
4. we rename agents preserving their locations to reach the reduced topology $\mathcal{T}_{\mathsf{MF}}^{t_0}$ (Lemma 4.4).

The first stage consists in proving that we can corrupt honest participants and still keep the same trace. Indeed a dishonest agent is actually more powerful than an honest one.

Lemma 4.1. *Let $\mathcal{T}_0 = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}_0, v_0, p_0)$ be a topology, and $\mathcal{K}_0$ be a configuration built on $\mathcal{T}_0$. Let $\mathcal{H}_0 \subseteq \mathcal{A}_0 \setminus \mathcal{M}_0$. Let $\mathcal{K}$ be a configuration such that $\mathcal{K}_0 \xrightarrow{\mathsf{tr}}_{\mathcal{T}_0} \mathcal{K}$. We have that $\mathcal{K}_0 \xrightarrow{\mathsf{tr}}_{\mathcal{T}'} \mathcal{K}$ where $\mathcal{T}' = (\mathcal{A}_0, \mathcal{M}_0 \cup \mathcal{H}_0, \mathsf{Loc}_0, v_0, p_0)$.*

Proof. We show this result by induction on the length of the derivation $\mathcal{K}_0 \xrightarrow{\mathsf{tr}}_{\mathcal{T}_0} \mathcal{K}$. The base case, i.e. tr is the empty trace, is trivial. To conclude, it is sufficient to show that:

$$\mathcal{K}_1 \xrightarrow{a,\alpha}_{\mathcal{T}_0} \mathcal{K}_2 \text{ implies } \mathcal{K}_1 \xrightarrow{a,\alpha}_{\mathcal{T}'} \mathcal{K}_2.$$

Let $\mathcal{K}_1 = (\mathcal{P}_1; \Phi_1; t_1)$. We consider each rule of the semantics one by one. Actually, the only rule that depends on the status (honest/dishonest) of the agents in the underlying topology is the rule IN. In such a case, we have that $\alpha = \mathtt{in}^\star(u)$ for some u. Moreover, following the notation introduced in Figure 4.1, we know that there exist $b \in \mathcal{A}_0$ (the agent responsible of the corresponding output) and $t_b \in \mathbb{R}_+$ (the time at which the output has been triggered). The only interesting case is when $b \in \mathcal{H}_0$, and therefore b is now a malicious agent in the topology $\mathcal{T}'$ whereas b was an honest one in the topology $\mathcal{T}_0$. Since, the IN rule was triggered in $\mathcal{T}_0$, we know that $u \in img(\lfloor \Phi_1 \rfloor_b^{t_b})$, and therefore there exists $\mathsf{w} \in dom(\lfloor \Phi_1 \rfloor_b^{t_b})$ such that $\mathsf{w}\Phi_1\downarrow = u$. Actually, it is easy to see that choosing the recipe $R = \mathsf{w}$ (and $c = b$) allows us to conclude. Indeed, we have that $t_b - \mathsf{Dist}_{\mathcal{T}'}(b, b) = t_b$, and therefore we conclude since we have already shown that $\mathsf{w} \in dom(\lfloor \Phi_1 \rfloor_b^{t_b})$. □

Applying Lemma 4.1, we can assume that all the agents but v_0 and p_0 are malicious. In the second stage, we are going to simplify the configuration by proving that, relying on the executability hypothesis presented in Section 4.1.3, it is not necessary to consider processes executed by dishonest agents. Indeed, the initial knowledge contained in the frame must be enough to mimic these processes.

Lemma 4.2. *Let $\mathcal{T}_0$ be a topology, $\mathcal{D}_0$ be a subset of malicious agents, and $\mathcal{K}_0 = (\mathcal{P}_0; \Phi_0; t_0)$ be a configuration built on $\mathcal{T}_0$ such that the process P_a is executable w.r.t. $img(\lfloor \Phi_0 \rfloor_a^{t_0})$ for any $\lfloor P_a \rfloor_a^{t_a} \in \mathcal{P}_0$ with $a \in \mathcal{D}_0$. Let $\mathcal{K} = (\mathcal{P}; \Phi_0 \uplus \Phi^+; t)$ be a configuration such that $\mathcal{K}_0 \xrightarrow{\mathsf{tr}}_{\mathcal{T}_0} \mathcal{K}$. We have that*

$$(\overline{\mathcal{P}_0}; \Phi_0; t_0) \xrightarrow{\overline{\mathsf{tr}}\sigma}_{\mathcal{T}_0} (\overline{\mathcal{P}}\sigma; \Phi_0 \uplus \overline{\Phi^+}\sigma; t)$$

where $\overline{\mathcal{P}_0}$ (resp. $\overline{\mathcal{P}}$, $\overline{\Phi^+}$, $\overline{\mathsf{tr}}$) is obtained from $\mathcal{P}_0$ (resp. $\mathcal{P}$, Φ^+, tr) by removing processes (resp. frame or trace elements) located in $a \in \mathcal{D}_0$ and $\sigma(n) = \mathsf{c}_0 \in \Sigma_0$ for any freshly generated name n used to trigger the NEW rules executed by agent $a \in \mathcal{D}_0$ in tr.

Proof. We show this result assuming that $\mathcal{D}_0$ contains a unique element a_0. The general result can then easily be obtained by a simple induction on the size of $\mathcal{D}_0$. More precisely, we show the following results by induction on the length of $\mathcal{K}_0 \xrightarrow{\mathsf{tr}}_{\mathcal{T}_0} \mathcal{K} = (\mathcal{P}; \Phi_0 \uplus \Phi^+; t)$:

1. for all $\lfloor P \rfloor_{a_0}^{t_a} \in \mathcal{P}$, $P\sigma$ is executable w.r.t. $\Psi(t)$;
2. for all $(\mathsf{w} \xrightarrow{a_0,t_a} u) \in \Phi^+$, there exists a recipe $R \in \mathcal{T}(\Sigma^+_{\mathsf{pub}}, dom(\Psi(t_a)))$ such that $R\Psi(t_a)\downarrow =_{\mathtt{E}} u\sigma$;
3. $(\overline{\mathcal{P}_0}; \Phi_0, t_0) \xrightarrow{\overline{\mathsf{tr}}\sigma}_{\mathcal{T}} (\overline{\mathcal{P}}\sigma; \Phi_0 \uplus \overline{\Phi^+}\sigma; t)$;

where $\Psi(t) = \Phi_0 \uplus \{ \lfloor \Phi^+\sigma \rfloor_b^{t - \mathsf{Dist}_{\mathcal{T}_0}(b,a_0)} \mid b \neq a_0\}$.

The base case, i.e. when tr is empty, is trivial. Now, we assume that

$$\mathcal{K}_0 \xrightarrow{\mathsf{tr}}_{\mathcal{T}_0} \mathcal{K} = (\mathcal{P}; \Phi_0 \uplus \Phi^+; t) \xrightarrow{a,\alpha}_{\mathcal{T}_0} \mathcal{K}' = (\mathcal{P}'; \Phi_0 \uplus \Phi'; t')$$

and thanks to our induction hypothesis, we know that the three properties above hold on $\mathcal{K}$. We note σ' the substitution regarding the trace $\mathsf{tr}.(a, \alpha)$. We do a case analysis on the rule involved in the last step.

Rule TIM. In such a case, we have that $\Phi' = \Phi^+$ and $\mathcal{P}' = \mathcal{P}$. Since $t' \geq t$, we have that $\Psi(t) \subseteq \Psi(t')$, and this allows us to conclude.

Rule OUT. Items (1) and (3) are quite obvious. Regarding item (2), the only non trivial case is when $a = a_0$. We have to show that the element added to the frame, namely $(\mathsf{w} \xrightarrow{a_0,t} u)$ satisfies the expected property. Let $\mathcal{P} = \{ \lfloor \mathtt{out}(u).P \rfloor_{a_0}^{t_a} \} \uplus \mathcal{P}_1$ and $\mathcal{P}' = \{ \lfloor P \rfloor_{a_0}^{t_a} \} \uplus \mathcal{P}_1$. By item (1), we know that the process $\mathtt{out}(u\sigma).P\sigma$ responsible of this output is executable w.r.t. $img(\Psi(t))$, and thus there exists $R \in \mathcal{T}(\Sigma^+_{\mathsf{pub}}, dom(\Psi(t)) \cup bn(\mathtt{out}(u\sigma).P\sigma) \cup bv(\mathtt{out}(u\sigma).P\sigma))$ such that $R\Psi(t)\downarrow = u\sigma$. More precisely we have that $R \in \mathcal{T}(\Sigma^+_{\mathsf{pub}}, dom(\Psi(t)))$ because $u\sigma$ does not contain any bound names/variables, and $img(\psi(t))$ does not contain names that belong to $bn(P\sigma)$ since Φ_0 does not contain any name (only agent identities) and $\Phi^+\sigma$ neither since any bound name is freshened when executing the NEW rule (and thus before any output).

Rule LET. Items (2) and (3) are quite immediate. Indeed, regarding item (2) we conclude by remarking that $\sigma' = \sigma$ and $\Phi' = \Phi^+$. Regarding item (3) we simply note that if a term $u\downarrow$ is a constructor term then $u\sigma\downarrow$ is a constructor term too. Finally, the only non trivial point to establish is item (1) when $a = a_0$. Let $\mathcal{P} = \lfloor \mathtt{let}_{\mathtt{mess}}\ x = v\ \mathtt{in}\ P' \rfloor_{a_0}^{t_a} \uplus \mathcal{P}_1$, $\mathcal{P}' = \lfloor P'\{x \mapsto v\downarrow\} \rfloor_{a_0}^{t_a} \uplus \mathcal{P}_1$, $\Phi' = \Phi^+$, $t' = t$, and $\sigma' = \sigma$. By hypothesis, we know that the process $(\mathtt{let}_{\mathtt{mess}}\ x = v\sigma\ \mathtt{in}\ P'\sigma)$ is executable w.r.t. $\Psi(t)$, thus there exists $R \in \mathcal{T}(\Sigma^+_{\mathsf{pub}}, dom(\Psi(t)))$ such that $R\Psi(t)\downarrow =_{\mathtt{E}} v\sigma\downarrow$ (for the same reason as previously). To prove item (1), we have to show that $(P'\{x \mapsto v\downarrow\})\sigma$ is executable w.r.t. $\Psi(t') = \Psi(t)$. Let u be a term occurring in an output (resp. a let) in $(P'\{x \mapsto v\downarrow\})\sigma = P'\sigma\{x \mapsto v\sigma\downarrow\}$. We have that there exists u_0 that occurs in an output (resp. a let) in $P'\sigma$ such that

$u_0\{x \mapsto v\sigma\downarrow\} = u$, and by hypothesis, we know that $(\mathtt{let}_{\mathtt{mess}}\ x = v\sigma\ \mathtt{in}\ P'\sigma)$ is executable w.r.t. $\Psi(t)$, i.e. there exists $R_0 \in \mathcal{T}(\Sigma^+_{\mathsf{pub}}, dom(\Psi(t)) \cup bn(P'\sigma) \cup bv(P'\sigma) \cup \{x\})$ such that $R_0\Psi(t)\downarrow =_{\mathtt{E}} u_0\downarrow$ (actually there is no need of normalisation in case of an output). Let $R' = R_0\{x \mapsto R\}$. We have that $R' \in \mathcal{T}(\Sigma^+_{\mathsf{pub}}, dom(\Psi(t)) \cup bn(P'\sigma) \cup bv(P'\sigma))$ and

$$R'\Psi(t')\downarrow =_{\mathtt{E}} (R_0\{x \mapsto R\})\Psi(t)\downarrow =_{\mathtt{E}} (R_0\Psi(t)\{x \mapsto v\sigma\downarrow\})\downarrow =_{\mathtt{E}} (u_0\downarrow\{x \mapsto v\sigma\downarrow\})\downarrow =_{\mathtt{E}} u\downarrow.$$

Note that $u = u_0\{x \mapsto v\sigma\downarrow\}$ is a constructor term when u_0 is a constructor term. Thus, no normalisation is needed in case u is a term occurring in an output, and we have that $R'\Psi(t')\downarrow =_{\mathtt{E}} u$.

4

Rule NEW. In that case we have that $\mathcal{P} = \lfloor \mathtt{new}\ n.P' \rfloor_a^{t_a} \uplus \mathcal{P}_1$, $\mathcal{P}' = \lfloor P'\{n \mapsto n'\} \rfloor_a^{t_a} \uplus \mathcal{P}_1$, $\Phi' = \Phi^+$ and $t' = t$. The only interesting case is when $a = a_0$ and thus $\sigma' = \sigma \cup \{n' \mapsto \mathsf{c}_0\}$. Note that items (2) and (3) are immediate: we have that $\overline{\mathsf{tr}.(a_0, \tau)} = \overline{\mathsf{tr}}$ and n' is fresh and thus neither occurs in $\mathcal{P}_1$ nor in Φ^+.

Regarding item (1) we have to prove that $P'\{n \mapsto n'\}\sigma'$ is executable w.r.t. $\Psi(t)$. Let u be a term occurring in an output (resp. a let) in $P'\{n \mapsto n'\}$. We have that there exists a term u_0 that occurs in an output (resp. a let) in P' such that $u_0\{n \mapsto n'\} = u$. By hypothesis, $(\mathtt{new}\ n.P')\sigma$ is executable w.r.t. $\Psi(t)$, and thus there exists $R \in \mathcal{T}(\Sigma^+_{\mathsf{pub}}, dom(\Psi(t)) \cup bn(P'\sigma) \cup \{n\} \cup bv(P'\sigma))$ such that $R\Psi(t)\downarrow = u_0\sigma$ in case of an output (resp. $u_0\sigma\downarrow$ in case of a let). We note $R' = R\{n \mapsto \mathsf{c}_0\}$. We have that $P'\sigma = P'\sigma'$, and thus we obtain that $R' \in \mathcal{T}(\Sigma^+_{\mathsf{pub}}, dom(\Psi(t)) \cup bn(P'\sigma') \cup bv(P'\sigma'))$ and:

$$R'\Psi(t)\downarrow = R\{n \mapsto \mathsf{c}_0\}\Psi(t)\downarrow = R\Psi(t)\downarrow\{n \mapsto \mathsf{c}_0\}\downarrow = u_0\sigma\downarrow\{n \mapsto \mathsf{c}_0\}\downarrow = u\sigma'\downarrow.$$

Note that $u\sigma' = u_0\sigma\{n \mapsto \mathsf{c}_0\}$ is a constructor term when u_0, is a constructor term. Thus, no normalisation is needed in case u is a term occurring in an output.

Rule RST. In such a case, the result trivially holds.

Rule IN. In case $a = a_0$, the only non trivial point is to establish item (1), and this can be done in a similar way as it was done in Rule LET. Otherwise, i.e. when $a \neq a_0$, the non trivial point is to establish item (3). Let $\alpha = \mathtt{in}^\star(u)$. We have to establish that the IN rule can still be fired with the value $u\sigma\downarrow$ despite the fact that some elements have been removed from Φ^+.
Following the notations introduced in Figure 4.1, we know that there exist $b \in \mathcal{A}_0$ and $t_b \in \mathbb{R}_+$ such that $t_b \leq t - \mathsf{Dist}_{\mathcal{T}_0}(b, a)$ and a recipe R such that $R(\Phi_0 \uplus \Phi^+)\downarrow = u$ and for all $\mathsf{w} \in vars(R)$ there exists $c \in \mathcal{A}_0$ such that $\mathsf{w} \in dom(\lfloor \Phi_0 \uplus \Phi^+ \rfloor_c^{t_b - \mathsf{Dist}_{\mathcal{T}_0}(c,b)})$.

1. If $b \notin \mathcal{M}_0$ then we know that $R = \mathsf{w}_0$ for some w_0 and we have that $\lfloor \Phi_0 \uplus \Phi^+ \rfloor_b^{t_b} = \lfloor \Phi_0 \uplus \overline{\Phi^+} \rfloor_b^{t_b}$ because $b \neq a_0$. Thus, the rule can be applied considering the frame $\lfloor \Phi_0 \uplus \overline{\Phi^+\sigma} \rfloor_b^{t_b}$.

2. If $b \in \mathcal{M}_0$, then in case $c \neq a_0$, or $c = a_0$ with $\mathsf{w} \in \Phi_0$, then it we immediately conclude since σ does not apply on it. The interesting case is when $\mathsf{w} \in dom(\lfloor \Phi^+ \rfloor_{a_0}^{t_b - \mathsf{Dist}_{\mathcal{T}_0}(a_0,b)})$, and we have to reconstruct $(\mathsf{w}\Phi^+)\sigma\downarrow$ with elements available in $\Phi_0 \uplus \overline{\Phi^+}\sigma$.

   Let w be a variable such that $\mathsf{w} \in vars(R)$ and $\mathsf{w} \in dom(\lfloor \Phi^+ \rfloor_{a_0}^{t_b - \mathsf{Dist}_{\mathcal{T}_0}(a_0,b)})$. Thanks to item (2), we know that there exists $t_\mathsf{w} \leq t_b - \mathsf{Dist}_{\mathcal{T}_0}(a_0, b)$ and a recipe $R_\mathsf{w} \in \mathcal{T}(\Sigma_{\mathsf{pub}}, dom(\Psi(t_\mathsf{w})))$ such that $R_\mathsf{w}\Psi(t_\mathsf{w})\downarrow =_\mathtt{E} (\mathsf{w}\Phi^+)\sigma$. By definition of $\Psi(\_)$, we have that

$$\Psi(t_\mathsf{w}) = \Phi_0 \uplus \{ \lfloor \Phi^+\sigma \rfloor_c^{t_\mathsf{w} - \mathsf{Dist}_{\mathcal{T}_0}(c,a_0)} \mid c \neq a_0\}.$$

   Moreover, we know that

$$t_\mathsf{w} - \mathsf{Dist}_{\mathcal{T}_0}(c, a_0) \leq t_b - (\mathsf{Dist}_{\mathcal{T}_0}(c, a_0) + \mathsf{Dist}_{\mathcal{T}_0}(a_0, b)) \leq t_b - \mathsf{Dist}_{\mathcal{T}_0}(c, b).$$

   Thus, we have that $vars(R_\mathsf{w}) \subseteq dom(\Phi_0) \uplus dom(\{ \lfloor \Phi^+\sigma \rfloor_c^{t_b - \mathsf{Dist}_{\mathcal{T}_0}(c,b)} \mid c \neq a_0\}$.

   Let θ be the substitution with domain $vars(R) \cap dom(\lfloor \Phi^+\sigma \rfloor_{a_0}^{t_b - \mathsf{Dist}_{\mathcal{T}_0}(a_0,b)})$ and such that $\mathsf{w}\theta = R_\mathsf{w}$ as defined above. We have that $R\theta \in \mathcal{T}(\Sigma^+_{\mathsf{pub}}, dom(\Phi_0 \uplus \overline{\Phi^+}\sigma))$ and

$$\begin{array}{rcl} R\theta(\Phi_0 \uplus \overline{\Phi^+}\sigma)\downarrow & =_\mathtt{E} & R(\Phi_0 \uplus \{\mathsf{w} \mapsto R_\mathsf{w}(\Phi_0 \uplus \overline{\Phi^+}\sigma) \mid \mathsf{w} \in dom(\theta)\})\downarrow \\ & =_\mathtt{E} & R(\Phi_0 \uplus \{\mathsf{w} \mapsto (\mathsf{w}\Phi^+)\sigma \mid \mathsf{w} \in dom(\theta)\})\downarrow \\ & =_\mathtt{E} & R(\Phi_0 \uplus \Phi^+\sigma)\downarrow \\ & =_\mathtt{E} & (R(\Phi_0 \uplus \Phi^+))\sigma\downarrow \\ & =_\mathtt{E} & (R(\Phi_0 \uplus \Phi^+)\downarrow)\sigma\downarrow \\ & =_\mathtt{E} & u\sigma\downarrow. \end{array}$$

   Note that for all $\mathsf{w} \in vars(R\theta)$, by definition of $\Psi(t_\mathsf{w})$, we have that $(\mathsf{w} \xrightarrow{c,t_c} v) \in \Phi_0 \uplus \overline{\Phi^+}\sigma$ (for some c, t_c, v) such that $t_c \leq t_b - \mathsf{Dist}_{\mathcal{T}_0}(c, b)$, and thus the IN rule can be triggered at the same time and relying on the same agent b as in the original execution trace.

□

The two first stages simplify the initial configuration but many dishonest agents located at many different locations still belong to the topology. To reduce the number of these dishonest agents, we got some inspiration from the work of Nigam *et al.* in [NTU16]. In this work, the authors prove that considering topologies in which each honest agent is accompanied by a dishonest one located at the same place is enough. These canonical topologies are defined as follows: given a set $V = \{a_1, \ldots, a_p\}$ of honest agents together with a location function $\mathsf{Loc}_V : V \mapsto \mathbb{R}^3$, we define the canonical topology $\mathcal{T}_{\mathsf{Loc}_V}$ associated to Loc_V as $\mathcal{T}_{\mathsf{Loc}_V} = (V \uplus \mathcal{M}_0, \mathcal{M}_0, \mathsf{Loc}_V \uplus \mathsf{Loc}_0, v_0, p_0)$ where:

- $\mathcal{M}_0 = \{i_1, \ldots, i_p\}$ with $i_1 \ldots, i_p \in \mathcal{A} \setminus V$;

- $\mathsf{Loc}_0(i_j) = \mathsf{Loc}(a_j)$ for $j \in \{1, \ldots, p\}$.

We can now ideally place the dishonest agents, i.e. one close to each honest agent. The following lemma proves that, given an execution, and whatever the underlying topology is, the same execution can be executed in the topology considering a dishonest agent close to each honest one.

Lemma 4.3. *Let $\mathcal{T} = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}, v_0, p_0)$ be a topology, $\mathcal{K}_0 = (\mathcal{P}_0; \Phi_0; t_0)$ and $\mathcal{K}$ be two configurations built on $\mathcal{T}_0$ such that $\mathcal{K}_0 \xrightarrow{\mathsf{tr}}_{\mathcal{T}} \mathcal{K}$, and H be a set of agents such that*

$$\{a \mid \lfloor P \rfloor_a^t \in \mathcal{P}_0 \text{ or } \lfloor \Phi_0 \rfloor_a^t \neq \emptyset\} \subseteq H \subseteq \mathcal{A}_0 \setminus \mathcal{M}_0.$$

We have that $(\mathcal{P}_0; \Phi_0; t_0) \xrightarrow{\mathsf{tr}}_{\mathcal{T}_{\mathsf{Loc}_{|H}}} \mathcal{K}$ where $\mathcal{T}_{\mathsf{Loc}_{|H}}$ is the canonical topology associated to $\mathsf{Loc}_{|H}$.

We note that the set H must at least contain the names of the agents executing a process in $\mathcal{P}_0$ or occurring in the frame Φ_0 to maintain that $\mathcal{K}_0$ is a configuration when considering the topology $\mathcal{T}_{\mathsf{Loc}_{|H}}$.

Proof. We show this result by induction on the length of the derivation $\mathcal{K}_0 \xrightarrow{\mathsf{tr}}_{\mathcal{T}_{\mathsf{Loc}_{|H}}} \mathcal{K}$. The base case, i.e. tr is the empty trace, is trivial. To conclude, it is sufficient to show that:

$$\mathcal{K}_1 \xrightarrow{a,\alpha}_{\mathcal{T}} \mathcal{K}_2 \text{ implies } \mathcal{K}_1 \xrightarrow{a,\alpha}_{\mathcal{T}_{\mathsf{Loc}_{|H}}} \mathcal{K}_2.$$

In the following, we note $\mathcal{T}_{\mathsf{Loc}_{|H}} = (\mathcal{A}', \mathcal{M}', \mathsf{Loc}', v_0, p_0)$. We do the proof considering each rule of the semantics one by one but, actually, the only rule that depends on the underlying topology is the rule IN. In such a case, we have that $\alpha = \mathtt{in}^\star(u)$ for some message u. Moreover, we denote $\mathcal{K}_1 = (\mathcal{P}_1; \Phi_1; t_1)$ and following the notations introduced in Figure 4.1, we know that there exist $b \in \mathcal{A}_0$ (the agent responsible of the corresponding output) and $t_b \leq t_1 - \mathsf{Dist}_{\mathcal{T}}(b, a)$ (the time at which the output has been triggered) that satisfy the conditions of the rule. We distinguish two cases:

1. In case $b \in \mathcal{A}_0 \setminus \mathcal{M}_0$, then we know that there exists $\mathsf{w} \in img(\lfloor \Phi_1 \rfloor_b^{t_b})$ such that $\mathsf{w}\Phi_1{\downarrow} = u$. By definition of H, we have that $b \in H$, and therefore $b \in \mathcal{A}' \setminus \mathcal{M}'$. The same rule applies for the same reason.

2. In case $b \in \mathcal{M}_0$, then we know that there exists a recipe R such that $R\Phi_1{\downarrow} = u$, and for all $\mathsf{w} \in vars(R)$ there exists $c \in \mathcal{A}_0$ such that $\mathsf{w} \in dom(\lfloor \Phi_1 \rfloor_c^{t_b - \mathsf{Dist}_{\mathcal{T}}(c,b)})$. We show that the same rule applies using the same recipe R. However, the agent responsible of the output will be the agent i_a such that $\mathsf{Loc}'(i_a) = \mathsf{Loc}'(a) = \mathsf{Loc}(a)$ (note that $a \in H$), and this output will be performed at time t_1 (instead of t_b). We have that $R\Phi_1{\downarrow} =_{\mathtt{E}} u$. Now, let $\mathsf{w} \in vars(R)$. Let $c \in \mathcal{A}_0$ such that $\mathsf{w} \in dom(\lfloor \Phi_1 \rfloor_c^{t_b - \mathsf{Dist}_{\mathcal{T}}(c,b)})$. We have that $c \in H$. It remains to show that

$\mathsf{w} \in dom(\lfloor \Phi_1 \rfloor_c^{t_1 - \mathsf{Dist}_{\mathcal{T}_{\mathsf{Loc}_{|H}}}(c,i_a)})$. For this, it is actually sufficient to establish that $t_1 - \mathsf{Dist}_{\mathcal{T}'}(c, i_a) \geq t_b - \mathsf{Dist}_{\mathcal{T}}(c, b)$. We know that:

$$\begin{array}{rrcl} & t_1 - \mathsf{Dist}_{\mathcal{T}}(b,a) & \geq & t_b \\ \Rightarrow & t_1 - \mathsf{Dist}_{\mathcal{T}}(b,a) - \mathsf{Dist}_{\mathcal{T}}(c,b) & \geq & t_b - \mathsf{Dist}_{\mathcal{T}}(c,b) \\ \Rightarrow & t_1 - (\mathsf{Dist}_{\mathcal{T}}(b,a) + \mathsf{Dist}_{\mathcal{T}}(c,b)) & \geq & t_b - \mathsf{Dist}_{\mathcal{T}}(c,b) \\ \Rightarrow & t_1 - \mathsf{Dist}_{\mathcal{T}}(c,a) & \geq & t_b - \mathsf{Dist}_{\mathcal{T}}(c,b) \end{array}$$

The last implication comes from the triangle inequality for the distance. Then, we obtain the expected result since $\mathsf{Loc}'(i_a) = \mathsf{Loc}'(a)$, and thus $\mathsf{Dist}_{\mathcal{T}}(c,a) = \mathsf{Dist}_{\mathcal{T}_{\mathsf{Loc}_{|H}}}(c,a)$ since $a, c \in H$.

This concludes the proof. □

Finally, the last step consists in reducing the number of different agent names that appear in the frame. Indeed, if Lemma 4.3 enables us to reduce the number of agents belonging to the topology, it does not modify the frame, which may still contain messages involving names of agents who are no longer present in the topology. To avoid such a situation, we prove in Lemma 4.4 that we can always rename agents preserving their status (honest/dishonest).

Lemma 4.4. *Let $\mathcal{K}, \mathcal{K}'$ be two configurations built on $\mathcal{T} = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}, v_0, p_0)$ such that $\mathcal{K} \xrightarrow{\mathsf{tr}}_{\mathcal{T}} \mathcal{K}'$, and $\rho : \mathcal{A} \to \mathcal{A}_0$ be a renaming such that $\mathsf{Loc}(\rho(a)) = \mathsf{Loc}(a)$ for any $a \in \mathcal{A}_0$, and $\rho(a) \in \mathcal{M}_0$ for any $a \in \mathcal{M}_0$. We have that:*

$$\mathcal{K}\rho \xrightarrow{\mathsf{tr}\rho}_{\mathcal{T}} \mathcal{K}'\rho.$$

Proof. We show this result by induction on the length of $\mathcal{K} \xrightarrow{\mathsf{tr}}_{\mathcal{T}} \mathcal{K}'$. The base case, i.e. tr is the empty trace, is trivial. To conclude, it is sufficient to show that $\mathcal{K}_1 \xrightarrow{a,\alpha}_{\mathcal{T}} \mathcal{K}_2$ with $\mathcal{K}_1$ a configuration built on $\mathcal{T}$ implies that $\mathcal{K}_1\rho \xrightarrow{\rho(a),\rho(\alpha)}_{\mathcal{T}} \mathcal{K}_2\rho$. First, note that $\mathcal{K}_1$ only involves processes located at $a \in \mathcal{A}_0$, and therefore actions along the derivation are only executed by agents in $\mathcal{A}_0$ whose locations remain unchanged by ρ. We consider each rule of the semantics one by one. The only rules that are not trivial are the rules LET and IN.

Case of the rule LET. In such a case, we have that $\mathcal{K}_1 = (\mathcal{P}_1; \Phi_1; t_1)$ and $\mathcal{K}_2 = (\mathcal{P}_2; \Phi_2; t_2)$ with $\mathcal{P}_1 = \lfloor \mathtt{let}_{\mathtt{mess}}\ x = u\ \mathtt{in}\ P \rfloor_a^{t_a} \uplus \mathcal{P}$, $\mathcal{P}_2 = \lfloor P\{x \mapsto u\downarrow\} \rfloor_a^{t_a} \uplus \mathcal{P}$, $\Phi_2 = \Phi_1$, and $t_2 = t_1$. We know that $u\downarrow \in \mathcal{T}(\Sigma_c^+, \mathcal{N} \cup \mathcal{A} \cup \mathbb{R}_+)$, and therefore we have that $(u\rho)\downarrow \in \mathcal{T}(\Sigma_c^+, \mathcal{N} \cup \mathcal{A} \cup \mathbb{R}_+)$ by applying the same rewriting rule at each step. This allows us to apply the rule LET and to obtain the expected result.

Case of the rule IN. In such a case, we have that $\mathcal{K}_1 = (\mathcal{P}_1; \Phi_1; t_1)$ and $\mathcal{K}_2 = (\mathcal{P}_2; \Phi_2; t_2)$ with $\mathcal{P}_1 = \lfloor \mathtt{in}^\star(x).P \rfloor_a^{t_a} \uplus \mathcal{P}$, $\mathcal{P}_2 = \lfloor P\{x \mapsto u\} \rfloor_a^{t_a} \uplus \mathcal{P}$, $\Phi_2 = \Phi_1$, and $t_2 = t_1$. We know that there exists $b \in \mathcal{A}_0$, $t_b \in \mathbb{R}_+$ such that $t_b \leq t_1 - \mathsf{Dist}_{\mathcal{T}}(b,a)$, and a recipe R such that $u = R\Phi_1\downarrow$, and for all $\mathsf{w} \in vars(R)$ there exists $c \in \mathcal{A}_0$ such that $w \in dom(\lfloor \Phi_1 \rfloor_c^{t_b - \mathsf{Dist}_{\mathcal{T}}(c,b)})$. Moreover, when $b \in \mathcal{A}_0 \setminus \mathcal{M}_0$, we know that $R \in \mathcal{W}$. To

conclude that the rule IN can be applied, we simply have to show that $R(\Phi\rho)\downarrow = u\rho$. Note that the renaming ρ keeps the locations of the agents in $\mathcal{A}_0$ unchanged, and therefore the conditions about distance are still satisfied. We have that $R(\Phi\rho)\downarrow = (R\Phi)\rho\downarrow$ since $R \in \mathcal{T}(\Sigma^+_{\mathsf{pub}}, \mathcal{W})$. We know that $(R\Phi)\downarrow = u$, and therefore we have that $(R\Phi)\rho\downarrow = u\rho$ by applying the same rewriting rule at each step. Note that $u\rho$ does not contain any destructor symbol and is thus in normal form. To conclude, it remains to ensure that when $\rho(b) \in \mathcal{A}_0 \setminus \mathcal{M}_0$, then $R \in \mathcal{W}$. Actually, we know that if $\rho(b) \notin \mathcal{M}_0$ then $b \notin \mathcal{M}_0$ by hypothesis, and this allows us to conclude. □

Combining these four lemmas, we are now able to state and prove our main reduction result that allows us to get rid of the quantification over all the topologies. The security analysis can be done considering solely the topology $\mathcal{T}^{t_0}_{\mathsf{MF}}$., Given a frame Φ_0, we note $names(\Phi_0)$ the set of agent names occurring in it.

Theorem 4.1. *Let $\mathcal{I}_0$ be a template, $\mathcal{P}_{\mathsf{db}}$ a protocol, $t_0 \in \mathbb{R}_+$ a threshold, and Φ_0 an initial frame such that $names(\Phi_0) \subseteq \{v_0, p_0\}$.*

There exists a topology $\mathcal{T}_0 = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}_0, v_0, p_0) \in \mathcal{C}^{t_0}_{\mathsf{MF}}$ and a valid initial configuration $\mathcal{K}$ for $\mathcal{P}_{\mathsf{db}}$ w.r.t. $\mathcal{T}_0$ and $\Phi^{\mathcal{T}_0}_{\mathcal{I}_0} \cup \Phi_0$ such that

$$\mathcal{K} \xrightarrow{\mathsf{tr}}_{\mathcal{T}_0} (\lfloor \mathsf{end}(v_0, p_0) \rfloor^{t_v}_{v_0} \uplus \mathcal{P}; \Phi; t)$$

if, and only if, there exists a valid initial configuration $\mathcal{K}'$ for $(\mathtt{V}, \mathtt{P})$ w.r.t. $\Phi^{\mathcal{T}^{t_0}_{\mathsf{MF}}}_{\mathcal{I}_0} \cup \Phi_0$ such that

$$\mathcal{K}' \xrightarrow{\mathsf{tr}'}_{\mathcal{T}^{t_0}_{\mathsf{MF}}} (\lfloor \mathsf{end}(v_0, p_0) \rfloor^{t_v}_{v_0} \uplus \mathcal{P}'; \Phi'; t').$$

Proof. Since $\mathcal{T}^{t_0}_{\mathsf{MF}} \in \mathcal{C}^{t_0}_{\mathsf{MF}}$, the implication from right to left is easy to prove. Thus, we consider the other one ($\Rightarrow$). Let $\mathcal{T}_0 = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}_0, v_0, p_0) \in \mathcal{C}^{t_0}_{\mathsf{MF}}$ and $\mathcal{K}_0 = (\mathcal{P}_0; \Phi^{\mathcal{T}_0}_{\mathcal{I}_0} \cup \Phi_0; 0)$ be a valid initial configuration for $\mathcal{P}_{\mathsf{db}}(\mathtt{V}(\mathsf{x}_0, \mathsf{x}_1), \mathtt{P}(\mathsf{x}_0, \mathsf{x}_1))$ w.r.t. $\mathcal{T}_0$ and $\Phi^{\mathcal{T}_0}_{\mathcal{I}_0} \cup \Phi_0$ such that:

$$\mathcal{K}_0 \xrightarrow{\mathsf{tr}}_{\mathcal{T}_0} (\lfloor \mathsf{end}(v_0, p_0).P \rfloor^{t_a}_{v_0} \uplus \mathcal{P}; \Phi^{\mathcal{T}_0}_{\mathcal{I}_0} \cup \Phi_0 \cup \Phi; t).$$

To establish this result, we combine the previous lemmas to show that there exists a corresponding trace of execution in $\mathcal{T}^{t_0}_{\mathsf{MF}}$. As already announced, the proof is performed in four steps.

Step 1. Applying Lemma 4.1 with $\mathcal{H} = \mathcal{A}_0 \setminus (\mathcal{M}_0 \cup \{v_0, p_0\})$, we obtain a topology $\mathcal{T}_1 = (\mathcal{A}_1, \mathcal{M}_1, \mathsf{Loc}_1, v_0, p_0)$ where $\mathcal{A}_1 = \mathcal{A}_0$, $\mathcal{M}_1 = \mathcal{A}_0 \setminus \{v_0, p_0\}$, and $\mathsf{Loc}_1 = \mathsf{Loc}_0$. We have that:

$$\mathcal{K}_0 \xrightarrow{\mathsf{tr}}_{\mathcal{T}_1} (\lfloor \mathsf{end}(v_0, p_0) \rfloor^{t_a}_{v_0} \uplus \mathcal{P}; \Phi^{\mathcal{T}_0}_{\mathcal{I}_0} \cup \Phi_0 \cup \Phi; t).$$

We now consider the configuration $\mathcal{K}^+_0 = (\mathcal{P}_0; \Phi^{\mathcal{T}_1}_{\mathcal{I}_0} \cup \Phi_0; 0)$. Since $\Phi^{\mathcal{T}_0}_{\mathcal{I}_0} \subseteq \Phi^{\mathcal{T}_1}_{\mathcal{I}_0}$ (indeed by adding malicious agents, we have only increased the knowledge of the attacker) and thus we have that:

$$\mathcal{K}^+_0 \xrightarrow{\mathsf{tr}}_{\mathcal{T}_1} (\lfloor \mathsf{end}(v_0, p_0) \rfloor^{t_a}_{v_0} \uplus \mathcal{P}; \Phi^{\mathcal{T}_1}_{\mathcal{I}_0} \cup \Phi_0 \cup \Phi; t).$$

Step 2. Since the configuration $\mathcal{K}_0$ is valid, we know that for all $\lfloor Q \rfloor_{a_0}^{t_a} \in \mathcal{P}_0$ (but $\mathsf{V_{end}}(v_0, p_0)$), either $Q = \mathtt{V}(a_0, a_1)$ or $Q = \mathtt{P}(a_0, a_1)$ for some $a_1 \in \mathcal{A}_0 = \mathcal{A}_1$. We assume w.l.o.g. that we are in the first case. Let $\tau = \{\mathsf{x}_0 \mapsto a_0, \mathsf{x}_1 \mapsto a_1\}$. We know that for any term u occurring in an output or a let construction in Q, there exists a corresponding term u' occurring in an output or a let construction in $\mathtt{V}(\mathsf{x}_0, \mathsf{x}_1)$ such that $u = u'\tau$. Let $\mathcal{I}_0 = \{v_1, \ldots, v_k\}$, and $\sigma = \{\mathsf{w}_1 \mapsto v_1, \ldots, \mathsf{w}_k \mapsto v_k\}$. Since $\mathtt{V}(\mathsf{x}_0, \mathsf{x}_1)$ is $\mathcal{I}_0$-executable by definition of a protocol, there exists a term $R \in \mathcal{T}(\Sigma^+_{\mathsf{pub}}, \{\mathsf{w}_1, \ldots, \mathsf{w}_k\} \cup bn(\mathtt{V}(\mathsf{x}_0, \mathsf{x}_1)) \cup bv(\mathtt{V}(\mathsf{x}_0, \mathsf{x}_1)))$ such that $u' =_{\mathtt{E}} R\sigma\downarrow$. Thus, we know that $u\downarrow = u'\tau\downarrow =_{\mathtt{E}} R\sigma\downarrow\tau\downarrow = R\sigma\tau\downarrow$. Actually, we have that $\{v_1\tau, \ldots, v_k\tau\} \subseteq \mathsf{Knows}(\mathcal{I}_0, a_0, \mathcal{A}_1)$. Therefore if $a_0 \in \mathcal{M}_1$, since $\mathsf{Knows}(\mathcal{I}_0, a_0, \mathcal{A}_1) \subseteq img(\lfloor \Phi^{\mathcal{T}_1}_{\mathcal{I}_0} \rfloor_{a_0}^{0})$, we have that Q is executable w.r.t. $img(\Phi^{\mathcal{T}_1}_{\mathcal{I}_0} \cup \Phi_0)$.

Therefore, we can apply Lemma 4.2 with $\mathcal{K}_0^+ = (\mathcal{P}_0; \Phi^{\mathcal{T}_1}_{\mathcal{I}_0} \cup \Phi_0; 0)$ and $\mathcal{D}_0 = \mathcal{M}_1$ and conclude that:

$$(\overline{\mathcal{P}_0}; \Phi^{\mathcal{T}_1}_{\mathcal{I}_0} \cup \Phi_0; 0) \xrightarrow{\overline{\mathsf{tr}}\sigma_2}_{\mathcal{T}_1} (\lfloor \mathsf{end}(v_0, p_0) \rfloor_{v_0}^{t_a} \uplus \overline{\mathcal{P}}\sigma_2; \Phi^{\mathcal{T}_1}_{\mathcal{I}_0} \cup \Phi_0 \cup \overline{\Phi}\sigma_2; t)$$

where $\sigma_2(n) = \mathsf{c}_0 \in \Sigma_0$ for any name n freshly generated by an agent in $\mathcal{M}_1$ and $\overline{\mathcal{P}}$ (resp. $\overline{\Phi}$, and $\overline{\mathsf{tr}}$) is obtained from $\mathcal{P}$ (resp. Φ, tr) by removing processes (frame elements, actions) located in $a \in \mathcal{D}_0 = \mathcal{M}_1$.

Step 3. We now consider $\Phi^{\mathcal{T}_1+}_{\mathcal{I}_0}$ the same frame as $\Phi^{\mathcal{T}_1}_{\mathcal{I}_0}$ but frame elements located at $a \in \mathcal{M}_1 \setminus \{v_0, p_0, i_2\}$ are moved to v_0 and those located at i_2 are moved to p_0. Let $\delta_0 = \max(\mathsf{Dist}_{\mathcal{T}_1}(a, b))$ for any $a, b \in \mathcal{A}_1$. Clearly, we have that:

$$(\overline{\mathcal{P}_0}; \Phi^{\mathcal{T}_1+}_{\mathcal{I}_0} \cup \Phi_0; \delta_0) \xrightarrow{\overline{\mathsf{tr}}\sigma_2}_{\mathcal{T}_1} (\lfloor \mathsf{end}(v_0, p_0) \rfloor_{v_0}^{t_a} \uplus \overline{\mathcal{P}}\sigma_2; \Phi^{\mathcal{T}_1+}_{\mathcal{I}_0} \cup \Phi_0 \cup \mathsf{Shift}(\overline{\Phi}\sigma_2, \delta_0, ;)t + \delta_0).$$

Indeed, adding a delay δ_0 in the initial configuration, all the messages moved from an agent $a \in \mathcal{M}_1$ to v_0 or p_0 can be used by a at the time δ_0 and thus the initial execution can be followed shifted by δ_0. Applying Proposition 4.1, we can turn the first configuration into an initial one, i.e. with a global time set to 0. Therefore, we have that:

$$(\overline{\mathcal{P}_0}; \Phi^{\mathcal{T}_1+}_{\mathcal{I}_0} \cup \Phi_0; 0) \xrightarrow{\overline{\mathsf{tr}}\sigma_2}_{\mathcal{T}_1} (\lfloor \mathsf{end}(v_0, p_0) \rfloor_{v_0}^{t_a} \uplus \mathcal{P}'; \Phi'; t')$$

where $\mathcal{P}'$ (resp. Φ') coincides with $\overline{\mathcal{P}}\sigma_2$ (resp. $\Phi^{\mathcal{T}_1+}_{\mathcal{I}_0} \cup \Phi_0 \cup \mathsf{Shift}(\overline{\Phi}\sigma_2, \delta_0)$) up to local clocks (resp. timestamps). Then, we apply Lemma 4.3 on $\mathcal{T}_1$ and $(\overline{\mathcal{P}_0}; \Phi^{\mathcal{T}_1+}_{\mathcal{I}_0} \cup \Phi_0; 0)$ with $H = \{v_0, p_0\}$. We deduce that

$$(\overline{\mathcal{P}_0}; \Phi^{\mathcal{T}_1+}_{\mathcal{I}_0} \cup \Phi_0; 0) \xrightarrow{\overline{\mathsf{tr}}\sigma_2}_{\mathcal{T}_2} (\lfloor \mathsf{end}(v_0, p_0) \rfloor_{v_0}^{t_a} \uplus \mathcal{P}'; \Phi'; t')$$

where $\mathcal{T}_2 = (\{v_0, p_0, i_1, i_2\}, \{i_1, i_2\}, \mathsf{Loc}_H, v_0, p_0)$ is the canonical topology associated to H and $\mathsf{Loc}|_H$.

Step 4. To reduce the size of the initial frame, now we apply Lemma 4.4 on the configuration $\mathcal{K}_0' = (\overline{\mathcal{P}_0}; \Phi^{\mathcal{T}_1+}_{\mathcal{I}_0} \cup \Phi_0; 0)$ using $\rho : \mathcal{A} \to \mathcal{A}_1' = \{v_0, p_0, i_1, i_2\}$ such that $\rho(a) = i_1$ for any $a \notin \mathcal{A}_1'$, and $\rho(a) = a$ otherwise. We have that:

$$(\overline{\mathcal{P}_0}\rho; \Phi^{\mathcal{T}_1+}_{\mathcal{I}_0}\rho \cup \Phi_0\rho; 0) \xrightarrow{\overline{\mathsf{tr}}\sigma_2\rho}_{\mathcal{T}_2} (\lfloor \mathsf{end}(v_0, p_0) \rfloor_{v_0}^{t_a} \uplus \mathcal{P}'\rho; \Phi'\rho; t').$$

4

One may note that we can shorten the distance between v_0, i_1 and p_0, i_2 and keep the trace executable. The resulting topology, i.e. $(\mathcal{A}'_1, \{i_1, i_2\}, \mathsf{Loc}, v_0, p_0)$ with $\mathsf{Loc}(v_0) = \mathsf{Loc}(i_1)$, $\mathsf{Loc}(p_0) = \mathsf{Loc}(i_2)$ and $\mathsf{Dist}_{\mathcal{T}}(v_0, p_0) = t_0$, corresponds to $\mathcal{T}^{t_0}_{\mathsf{MF}}$.

To end this proof, it remains to turn $(\overline{\mathcal{P}_0\rho}; \Phi^{\mathcal{T}_1+}_{\mathcal{I}_0}\rho \cup \Phi_0\rho; 0)$ into a valid initial configuration. To do that, we simply have to move frame elements (those that we have added during Step 1 and moved during Step 3) located in v_0 to i_1 and those located in p_0 to i_2. This will not change the underlying execution since i_1 (resp. i_2) is located at the same place as v_0 (resp. p_0). In case there is no frame elements in i_1 or i_2 we add some elements, more precisely we add $\mathsf{Knows}(\mathcal{I}_0, i_1, \mathcal{A}'_1)$ or $\mathsf{Knows}(\mathcal{I}_0, i_2, \mathcal{A}'_1)$. These additional elements will not alter the underlying execution and the resulting frame corresponds to $\Phi^{\mathcal{T}^{t_0}_{\mathsf{MF}}}_{\mathcal{I}_0}$ up to possibly many replicates of the initial knowledge of agent i_1.

We may note that the process part $\overline{\mathcal{P}_0\rho}$ of the configuration satisfies Definition 2.3. Indeed, since $\mathcal{K}_0$ is valid, we have that $\lfloor \mathtt{V}_{\mathsf{end}}(v_0, p_0) \rfloor^0_{v_0} \in \mathcal{P}_0$ and by construction $\overline{\mathcal{P}_0\rho}$ still contains $\lfloor \mathtt{V}_{\mathsf{end}}(v_0, p_0) \rfloor^0_{v_0}$. Moreover, for all $\lfloor P' \rfloor^{t'}_{a'} \in \overline{\mathcal{P}_0\rho}$, by construction, we know that $a' \in \{v_0, p_0\} = \mathcal{A}'_1 \setminus \mathcal{M}'_1$. Finally, we have that there exists $\lfloor P'' \rfloor^{t'}_{a'} \in \mathcal{P}_0$ such that $P'' = \mathtt{V}(a', a_1)$ (resp. $P'' = \mathtt{P}(a', a_1)$) and $P' = P''\rho = \mathtt{V}(a', \rho(a_1))$ (resp. $P' = P''\rho = \mathtt{P}(a', \rho(a_1))$) with $\rho(a_1) \in \mathcal{A}'_1$. Therefore, we have that $(\overline{\mathcal{P}_0\rho}; \Phi^{\mathcal{T}^{t_0}_{\mathsf{MF}}}_{\mathcal{I}_0} \cup \Phi_0; 0) = (\overline{\mathcal{P}_0\rho}; \Phi^{\mathcal{T}^{t_0}_{\mathsf{MF}}}_{\mathcal{I}_0} \cup \Phi_0\rho; 0)$ is a valid initial configuration for $\mathcal{P}_{\mathsf{db}}$ w.r.t. $\mathcal{T}^{t_0}_{\mathsf{MF}}$.

This concludes the proof. □

As a direct corollary of the theorem above (applied with $\Phi_0 = \emptyset$), we have that we can restrict ourselves to consider the topology $\mathcal{T}^{t_0}_{\mathsf{MF}}$ when looking for a mafia fraud.

Corollary 4.1. *Let $\mathcal{I}_0$ be a template, $\mathcal{P}_{\mathsf{db}}$ a protocol, and $t_0 \in \mathbb{R}_+$ a threshold. We have that $(\mathtt{V}, \mathtt{P})$ admits a mafia fraud w.r.t. t_0-proximity if, and only if, there is an attack against t_0-proximity in the topology $\mathcal{T}^{t_0}_{\mathsf{MF}}$, i.e., there exists a valid initial configuration $\mathcal{K}$ for $(\mathtt{V}, \mathtt{P})$ w.r.t. $\mathcal{T}^{t_0}_{\mathsf{MF}}$ and $\emptyset$ such that:*

$$\mathcal{K} \to^*_{\mathcal{T}^{t_0}_{\mathsf{MF}}} (\lfloor \mathsf{end}(v_0, p_0) \rfloor^{t_v}_{v_0} \uplus \mathcal{P}; \Phi; t).$$

Theorem 4.1 also allows us to focus on a single topology when considering terrorist fraud resistance. However, this requires us to show that we can restrict ourselves to consider initial frames satisfying the hypothesis of Theorem 4.1, i.e. such that $\mathit{names}(\Phi_0) \subseteq \{v_0, p_0\}$.

Corollary 4.2. *Let $\mathcal{I}_0$ be a template, $(\mathtt{V}, \mathtt{P})$ a protocol, and $t_0 \in \mathbb{R}_+$ a threshold. We have that $(\mathtt{V}, \mathtt{P})$ is terrorist fraud resistant w.r.t. t_0-proximity if, and only if, for all semi-dishonest prover $P_{\mathtt{sd}}$ w.r.t. t_0 with frame $\Phi_{\mathtt{sd}}$ such that $\mathit{names}(\Phi_{\mathtt{sd}}) \subseteq \{v_0, p_0\}$, there exists a valid initial configuration $\mathcal{K}$ w.r.t. $\mathcal{T}^{t_0}_{\mathsf{MF}}$ and $\Phi^{\mathcal{T}^{t_0}_{\mathsf{MF}}}_{\mathcal{I}_0} \cup \Phi_{\mathtt{sd}}$ such that:*

$$\mathcal{K} \to^*_{\mathcal{T}^{t_0}_{\mathsf{MF}}} (\lfloor \mathsf{end}(v_0, p_0) \rfloor^{t'}_{v_0} \uplus \mathcal{P}; \Phi; t).$$

Proof. The implication from right to left is almost trivial. We consider a semi-dishonest prover $P_{\mathtt{sd}}$ with frame $\Phi_{\mathtt{sd}}$. If $\mathit{names}(\Phi_{\mathtt{sd}}) \subseteq \{v_0, p_0\}$, then we choose the topology $\mathcal{T}^{t_0}_{\mathsf{MF}} \in \mathcal{C}^{t_0}_{\mathsf{MF}}$ and we conclude relying on our hypothesis. Otherwise, we consider a bijective renaming σ from agent names outside $\{v_0, p_0\}$ to fresh constants in Σ_0 (i.e. never used in $(\mathtt{V}, \mathtt{P})$) and it is easy to see that $P_{\mathtt{sd}}\sigma$ with associated frame $\Phi_{\mathtt{sd}}\sigma$ is also a semi-dishonest prover. Now, we can rely on our hypothesis to obtain an execution trace in $\mathcal{T}^{t_0}_{\mathsf{MF}}$ (starting with $\Phi_{\mathtt{sd}}\sigma$) and applying σ^{-1} on it allows us to conclude.

Regarding the other implication, we consider a semi-dishonest prover $P_{\mathtt{sd}}$ with a frame $\Phi_{\mathtt{sd}}$ such that $\mathit{names}(\Phi_{\mathtt{sd}}) \subseteq \{v_0, p_0\}$. By hypothesis, there exists a valid initial configuration $\mathcal{K}_0$ for $\mathcal{P}_{\mathtt{db}}$ w.r.t. $\mathcal{T} \in \mathcal{C}^{t_0}_{\mathsf{MF}}$ and $\Phi^{\mathcal{T}}_{\mathcal{I}_0} \cup \Phi_{\mathtt{sd}}$ such that:

$$\mathcal{K} \xrightarrow{\mathsf{tr}}_{\mathcal{T}} (\, \lfloor \mathsf{end}(v_0, p_0) \rfloor^{t_v}_{v_0} \uplus \mathcal{P}; \Phi; t).$$

Theorem 4.1 applies since $\mathit{names}(\Phi_{\mathtt{sd}}) \subseteq \{v_0, p_0\}$, and thus we easily conclude. □

4.2.2. Distance hijacking

Unfortunately, the reduction proposed in case of mafia and terrorist frauds does not apply for distance hijacking attacks. Indeed, the third step of the reduction (Lemma 4.3) consists in placing a dishonest agent close to each honest one. This step introduces a dishonest agent in the vicinity of the verifier which is prohibited when considering distance hijacking scenarios.

Nevertheless, we show that under reasonable assumptions, we can reduce towards the topology $\mathcal{T}^{t_0}_{\mathsf{DH}} = (\mathcal{A}_{\mathsf{DH}}, \mathcal{M}_{\mathsf{DH}}, \mathsf{Loc}_{\mathsf{DH}}, v_0, p_0)$ which is composed of only three agents: v_0 and p_0 who are far away, and e_0 at the same location as p_0. The topology is depicted in Figure 4.3. More formally we have that $\mathcal{A}_{\mathsf{DH}} = \{p_0, v_0, e_0\}$, $\mathcal{M}_{\mathsf{DH}} = \{p_0\}$, $\mathsf{Loc}_{\mathsf{DH}}(p_0) = \mathsf{Loc}_{\mathsf{DH}}(e_0)$, and $\mathsf{Dist}_{\mathcal{T}^{t_0}_{\mathsf{DH}}}(p_0, v_0) = t_0$.

Figure 4.3: Topologies $\mathcal{T}^{t_0}_{\mathsf{DH}}$ where t_0 is the proximity threshold.

Given a process P, we denote $\overline{P}$ the process obtained from P by removing `reset` instructions, and replacing each occurrence of $\mathtt{in}^{<t}(x)$ by $\mathtt{in}(x)$. This notation is extended to a multiset of (extended) processes by applying the transformation to each process.

Theorem 4.2. *Let $\mathcal{I}_0$ be a template, $(\mathtt{V}, \mathtt{P})$ a protocol, and $t_0 \in \mathbb{R}_+$ a threshold. If $(\mathtt{V}, \mathtt{P})$ admits a distance hijacking attack w.r.t. t_0-proximity, then there exists a valid initial configuration $\mathcal{K}_0$ for $(\mathtt{V}, \mathtt{P})$ w.r.t. $\mathcal{T}^{t_0}_{\mathsf{DH}}$ and $\Phi^{\mathcal{T}^{t_0}_{\mathsf{DH}}}_{\mathcal{I}_0}$ such that $\mathcal{K}_0 = (\{\, \lfloor \mathtt{V}_{\mathsf{end}}(v_0, p_0) \rfloor^{0}_{v_0} \} \uplus \mathcal{P}_0; \Phi^{\mathcal{T}^{t_0}_{\mathsf{DH}}}_{\mathcal{I}_0}; 0)$ and*

$$(\{\, \lfloor \mathtt{V}_{\mathsf{end}}(v_0, p_0) \rfloor^{0}_{v_0} \} \uplus \overline{\mathcal{P}_0}; \Phi^{\mathcal{T}^{t_0}_{\mathsf{DH}}}_{\mathcal{I}_0}; 0) \xrightarrow{\mathsf{tr}}_{\mathcal{T}^{t_0}_{\mathsf{DH}}} (\, \lfloor \mathsf{end}(v_0, p_0) \rfloor^{t_v}_{v_0} \uplus \mathcal{P}'; \Phi; t).$$

Sadly, the resulting trace in the reduced topology $\mathcal{T}_{\mathsf{DH}}^{t_0}$ does not strictly appear as a distance hijacking attack w.r.t. t_0-proximity against the protocol. Indeed the initial configuration must be weakened: all the `reset` commands and guarded inputs are removed except those that belong to the target process, $\mathtt{V}_{\mathsf{end}}(v_0, p_0)$. As we will see in Chapter 5, this change is tight enough to not introduce false attacks and let us prove distance hijacking resistance for many protocols.

The proof of this theorem is more complex than the one presented in the previous section. The idea of this reduction is to move all the agents either at the same location as v_0, if he is in his vicinity or as p_0 otherwise. Unfortunately, this may lengthen the distance between agents and thus invalidate the witness trace of attack: it may break the timing constraints that applies on the guarded input. To ease the manipulation of the actions in a trace, we will first define more expressive labels, i.e. annotations, and an untimed semantics. This last will provide us more flexibility to reorder actions in the trace such that no remote agent execute actions during a challenge/response step. Finally, we will show how to lift a trace in this untimed semantics into a trace in our original timed semantics.

4

Annotations

We extend the existing labels with more informative annotations. More precisely, the first annotation will identify which process in the multiset has performed the action (session identifier). This will allow us to identify which specific agent performed some action. We also annotate the label with the global time at which the action has been done. In case of an output, we indicate the name of the handle $\mathtt{w}$ that has been used to store the output in the frame too. Finally, in case of an input, we indicate by a triplet (b, t_b, R) the name b of the agent responsible of the corresponding output, the time t_b at which this output has been performed, as well as the recipe R used to build this output.

Formally, a label is either empty (for the TIM rule) or of the form a, α with $\alpha \in \{\tau, \mathtt{out}(u), \mathtt{in}^\star(u)\}$, and thus an annotated label is:

- empty for the TIM rule;
- $(a, \alpha, s, t, \mathtt{w})$ when the underlying action (a, α) is of the form $(a, \mathtt{out}(u))$. In such a case, s is the session identifier of the agent responsible of this action, t is the global time at which this output has been done, and $\mathtt{w}$ is the handle added in the frame;
- $(a, \alpha, s, t, (b, t_b, R))$ when the underlying action (a, α) is of the form $(a, \mathtt{in}^\star(u))$. In such a case, s is the session identifier of the agent responsible of this action, t is the global time at which this input has been done, b is the agent responsible of the corresponding output, t_b the time at which this output has been done ($t_b \leq t$), and R the recipe that has been used to forge this output;
- $(a, \alpha, s, t, \emptyset)$ otherwise.

Thanks to these annotations, we are able to formally define dependencies between actions. This will be useful when reordering actions.

Definition 4.2. *Given a topology $\mathcal{T}$ and an execution $\mathcal{K}_0 \xrightarrow{L_1}_{\mathcal{T}} \mathcal{K}_1 \xrightarrow{L_2}_{\mathcal{T}} \cdots \xrightarrow{L_n}_{\mathcal{T}} \mathcal{K}_n$ we note $L_i = (a_i, \alpha_i, s_i, t_i, r_i)$ and $L_j = (a_j, \alpha_j, s_j, t_j, r_j)$ for $i, j \in \{1, \ldots, n\}$ and we say that L_j is* dependent *of L_i, denoted $L_j \hookrightarrow L_i$, if $i < j$, and:*

- *either $s_i = s_j$ (and thus $a_i = a_j$), and in that case L_j is* sequentially-dependent *of L_i, denoted $L_j \hookrightarrow_s L_i$;*
- *or $\alpha_i = \mathtt{out}(v)$, $\alpha_j = \mathtt{in}^\star(u)$, and $r_i \in vars(R_j)$ with $r_j = (b_j, t_j^b, R_j)$. In that case L_j is* data-dependent *of L_i, denoted $L_j \hookrightarrow_d L_i$.*

We note $\hookrightarrow^*$ the transitive closure of $\hookrightarrow$ and $L_j \not\hookrightarrow^* L_i$ when L_j is not dependent of L_i.

Note that if two actions are dependent, i.e. $(a_j, \alpha_j, s_j, t_j, r_j) \hookrightarrow^* (a_i, \alpha_i, s_i, t_i, r_i)$, then either they have been executed by the same agent or enough time must have elapsed between t_i and t_j in order to let a message travel from the location of agent a_i to the location of agent a_j. This is formally stated in the following lemma.

Lemma 4.5. *Let $\mathcal{T}$ be a topology, $\mathcal{K}_0 \xrightarrow{\mathsf{tr}_1 \ldots . \mathsf{tr}_n}_{\mathcal{T}} \mathcal{K}_1$ be an execution, and $i, j \in \{1, \ldots, n\}$ be such that $\mathsf{tr}_j \hookrightarrow^* \mathsf{tr}_i$. We have that $t_j \geq t_i + \mathit{Dist}_{\mathcal{T}}(a_i, a_j)$ where $\mathsf{tr}_i = (a_i, \alpha_i, s_i, t_i, r_i)$ and $\mathsf{tr}_j = (a_j, \alpha_j, s_j, t_j, r_j)$.*

Proof. By definition of $\hookrightarrow^*$, there exists $k \geq 1$ and $i_1, \ldots, i_k \in \{1, \ldots, n\}$ such that:

$$\mathsf{tr}_j = \mathsf{tr}_{i_1} \hookrightarrow \mathsf{tr}_{i_2} \hookrightarrow \ldots \hookrightarrow \mathsf{tr}_{i_k} = \mathsf{tr}_i.$$

We do the proof by induction on the length of this sequence. If $k = 1$ then $\mathsf{tr}_i = \mathsf{tr}_j$, and thus $t_i = t_j$ and $a_i = a_j$. The result trivially holds. Otherwise, relying on the induction hypothesis, we deduce that $t_{i_2} \geq t_i + \mathsf{Dist}_{\mathcal{T}}(a_i, a_{i_2})$. We distinguish two cases depending on the nature of the dependency $\mathsf{tr}_j \hookrightarrow \mathsf{tr}_{i_2}$.

Case $\mathsf{tr}_j \hookrightarrow_s \mathsf{tr}_{i_2}$. In such a case, we have that $a_j = a_{i_2}$ and $j \geq i_2$, and thus $t_j \geq t_{i_2}$. Therefore, we have:

$$t_j \geq t_{i_2} \geq t_i + \mathsf{Dist}_{\mathcal{T}}(a_i, a_{i_2}) = t_i + \mathsf{Dist}_{\mathcal{T}}(a_i, a_j).$$

Case $\mathsf{tr}_j \hookrightarrow_d \mathsf{tr}_{i_2}$. In such a case, we have that $r_j = (b, t_b, R)$, $r_{i_2} = \mathsf{w}$ and $\mathsf{w} \in vars(R)$. By definition of the IN rule we have that $t_j \geq t_b + \mathsf{Dist}_{\mathcal{T}}(b, a_j)$, and $t_b \geq t_{i_2} + \mathsf{Dist}_{\mathcal{T}}(a_{i_2}, b)$. Combining these inequalities with the one given by our induction hypothesis, we get:

$$t_j \geq t_i + \mathsf{Dist}_{\mathcal{T}}(a_i, a_{i_2}) + \mathsf{Dist}_{\mathcal{T}}(b, a_j) + \mathsf{Dist}_{\mathcal{T}}(a_{i_2}, b).$$

Relying on the triangle inequality, we get $t_j \geq t_i + \mathsf{Dist}_{\mathcal{T}}(a_i, a_j)$, and we conclude the proof. □

Untimed semantics

To have more flexibility when reordering actions, we define an untimed semantics. Given a configuration $\mathcal{K} = (\mathcal{P}; \Phi; t)$, we note $\mathsf{untimed}(\mathcal{K})$ the configuration associated to $\mathcal{K}$, i.e. $\mathsf{untimed}(\mathcal{K}) = (\mathcal{P}'; \Phi')$ with:

- $\mathcal{P}' = \{ \lfloor P \rfloor_a \mid \lfloor P \rfloor_a^t \in \mathcal{P} \text{ for some } t\}$;
- $\Phi' = \{\mathsf{w} \xrightarrow{a} u \mid (\mathsf{w} \xrightarrow{a,t} u) \in \Phi \text{ for some } t\}$.

The *untimed semantics* is then defined as follows: $\mathcal{K} \overset{a,\alpha,s,r}{\rightsquigarrow}_{\mathcal{T}} \mathcal{K}'$ if there exist $\mathcal{K}_0$ and $\mathcal{K}_0'$ such that $\mathcal{K}_0 \xrightarrow{a,\alpha,s,t,r'}_{\mathcal{T}} \mathcal{K}_0'$ (for some rule other than the (TIM) rule) with $\mathcal{K} = \mathsf{untimed}(\mathcal{K}_0)$, $\mathcal{K}' = \mathsf{untimed}(\mathcal{K}_0')$, and r is equal to r' up to the time annotation t^b in case of an input.

We may note that Definition 4.2 can be immediately adapted for untimed actions. As mentioned above, this untimed semantics provides more flexibility to reorder actions in a trace. Indeed, if two actions are independent, then we can swap them in the trace.

Lemma 4.6. *Let $\mathcal{T}$ be a topology, and $\mathcal{K}_0 \overset{L_1}{\rightsquigarrow}_{\mathcal{T}} \mathcal{K} \overset{L_2}{\rightsquigarrow}_{\mathcal{T}} \mathcal{K}_2$ be an execution such that $L_2 \not\leftrightarrow L_1$. We have that $\mathcal{K}_0 \overset{L_2}{\rightsquigarrow}_{\mathcal{T}} \mathcal{K}' \overset{L_1}{\rightsquigarrow}_{\mathcal{T}} \mathcal{K}_2$ for some configuration $\mathcal{K}'$.*

Proof. We consider $\mathcal{K}_0 = (\mathcal{P}_0; \Phi_0)$, $\mathcal{K} = (\mathcal{P}; \Phi)$, and $\mathcal{K}_2 = (\mathcal{P}_2; \Phi_2)$ three configurations such that $\mathcal{K}_0 \overset{L_1}{\rightsquigarrow}_{\mathcal{T}} \mathcal{K} \overset{L_2}{\rightsquigarrow}_{\mathcal{T}} \mathcal{K}_2$ with $L_2 \not\leftrightarrow L_1$. Let $L_1 = (a_1, \alpha_1, s_1, r_1)$ and $L_2 = (a_2, \alpha_2, s_2, r_2)$. Since $L_2 \not\leftrightarrow L_1$ we have that $s_1 \neq s_2$. Therefore, we have that $\mathcal{P}_0 = \lfloor \alpha_1.P_1 \rfloor_{a_1} \uplus \lfloor \alpha_2.P_2 \rfloor_{a_2} \uplus \mathcal{Q}$, $\mathcal{P} = \lfloor P_1' \rfloor_{a_1} \uplus \lfloor \alpha_2.P_2 \rfloor_{\alpha_2} \uplus \mathcal{Q}$, and $\mathcal{P}_2 = \lfloor P_1' \rfloor_{a_1} \uplus \lfloor P_2' \rfloor_{a_2} \uplus \mathcal{Q}$ for some actions α_1 and α_2.

In case $\alpha_1 = \mathtt{out}(v)$ and $\alpha_2 = \mathtt{in}^\star(u)$, we have that $\Phi_2 = \Phi = \Phi_0 \uplus \{\mathsf{w} \xrightarrow{a_1} v\}$. Since $L_2 \not\leftrightarrow_d L_1$, we know that $\mathsf{w} \notin \mathit{vars}(r_2)$, and thus $\mathit{vars}(r_2) \subseteq \mathit{dom}(\Phi_0)$. Now, let $\mathcal{K}' = (\lfloor \alpha_1.P_1 \rfloor_{a_1} \uplus \lfloor P_2' \rfloor_{a_2} \uplus \mathcal{Q}; \Phi_0)$. Relying on the fact that $\mathsf{w} \notin \mathit{vars}(r_2)$ in case $\alpha_1 = \mathtt{out}(v)$ and $\alpha_2 = \mathtt{in}^\star(u)$, it is easy to see that $\mathcal{K}_0 \overset{L_2}{\rightsquigarrow}_{\mathcal{T}} \mathcal{K}' \overset{L_1}{\rightsquigarrow}_{\mathcal{T}} \mathcal{K}_2$. The other cases can be done in a rather similar way. □

Toward the proof of Theorem 4.2

This proof proceeds in five steps:

1. we show that an attack against t_0-proximity for $\mathcal{P}_{\mathsf{db}} = (\mathtt{V}(\mathsf{x}_0, \mathsf{x}_1), \mathtt{P}(\mathsf{x}_0, \mathsf{x}_1))$ in a topology $\mathcal{T}$ remains a (weak) attack against t_0-proximity for $\mathcal{P}_{\mathsf{db}}$ in $\mathcal{T}$ having removed all the `reset` and guarded inputs in the initial configuration but in $\mathtt{V}_{\mathsf{end}}(v_0, p_0)$;
2. we consider the trace witnessing the attack into the untimed semantics to be able to reorder its actions such that only agents close to v_0 act between a reset action and its following guarded input;
3. we move the agents close to the verifier v_0 at his location and the agents far from v_0 to p_0's location;

4. we lift this new trace in the untimed semantics into a trace in the timed one;

5. we reduce the number of agent names that appear in the execution applying Lemma 4.4

Amongst these five steps, we may note that the first, the third and the fifth ones are almost immediate. Indeed, in the first step, we simply remove reset actions and replace guarded inputs by standard inputs. These transformations enable more behaviours. In the third step, the main point is that the topology is no longer relevant in the untimed semantics. Finally, the fifth step is the same as the last step of the proof of Theorem 4.1 presented before.

Let us focus on the steps (2) and (4). The step (2) is a consequence of the conjunction of Proposition 4.2 presented below and Lemma 4.5. The main idea of the proof of Theorem 4.2 is to move all the actions that do not depend on the reset or the guarded input outside the rapid phase. Then, applying Lemma 4.5 on the actions that remain during the rapid phase, we deduce that they must be executed by agents close to the verifier v_0.

Finally in order to lift the trace in the timed semantics in step (4), we just notice that since all the agents close to v_0 has been moved at its location at step (3). The rapid phase can thus be performed (almost) instantaneously. Indeed, messages take no time to travel between agents who are involved during the rapid phase.

The main proposition used to clean the trace in between a `reset` command and the following guarded input is Proposition 4.2. Its proof, together with the full proof of Theorem 4.2, are available in Appendix B.

Proposition 4.2. *Let $\mathcal{T}$ be a topology, and $\mathcal{K}_0 \overset{\mathsf{tr}_1 \dots \mathsf{tr}_n}{\rightsquigarrow}_{\mathcal{T}} \mathcal{K}_n$ be an execution with $n \geq 2$. We have that there exists a bijection $\varphi : \{1, \dots, n\} \to \{1, \dots, n\}$ such that:*

- $\mathcal{K}_0 \overset{\mathsf{tr}'_1 \dots \mathsf{tr}'_n}{\rightsquigarrow}_{\mathcal{T}} \mathcal{K}_n$ *with* $\mathsf{tr}_i = \mathsf{tr}'_{\varphi(i)}$ *for all* $i \in \{1, \dots, n\}$*; and*
- *for all j such that $\varphi(1) < j < \varphi(n)$, we have that* $\mathsf{tr}'_{\varphi(n)} \hookrightarrow^* \mathsf{tr}'_j \hookrightarrow^* \mathsf{tr}'_{\varphi(1)}$.

This proposition will be applied on the sub-executions that correspond to rapid phases, i.e. such that tr_1 is a `reset` action and tr_n a guarded input. It allows to reorder (note that φ is a bijection) the actions in a trace such that all the actions independent of the first action are moved before (i.e. at the beginning of the execution) and all the actions on which the last action does not depend on are moved after (i.e. at the end of the execution).

4.2.3. About restricted agents

The reduction results presented in the two previous sections assume that an agent is able to act as a verifier and as a prover at the same time. This assumption is evident when looking at Lemma 4.4: the projection function ρ does not make any difference between agent identities that execute prover roles or verifier roles.

Depending on the application, it can happen that some scenarios are not realisable. For example, few protocols, like SPADE [BGG+16] or EMV-payment protocols, have been designed assuming that identities are correctly distributed by an authority, i.e. an agent cannot simultaneously be a prover and a verifier. Fortunately our reduction results can easily be adapted to handle this restriction. We obtain similar reduced topologies in which almost all the agents must be duplicated in order to get one honest (resp. dishonest) prover representative and one honest (resp. dishonest) verifier representative at each location. These two topologies are presented in Figure 4.4.

All the results presented so far in this chapter apply modulo small changes: first, the template $\mathcal{I}_0$ used to derive the initial knowledge of an agent should be split in order to differentiate the knowledge of a prover from the verifier's one. As expected Definition 2.3 must be adapted too in order to carefully fill the initial configuration depending on each identity (either a prover of a verifier). Finally, amongst all the technical lemmas, only Lemma 4.4 must be slightly modified in order to respect the status (prover/verifier) of the agents when applying the projection function. The proof can be immediately adapted keeping in mind that a prover (resp. verifier) identity must only be projected on another prover (resp. verifier) identity.

Note that it is important to decide whether an agent is able to act as a prover and a verifier at the same time or not. Indeed, depending on this choice of modelling, protocols may be proved secure or not. An example of such a protocol is the SPADE protocol [BGG+16] which can be proved mafia fraud resistant when distinguishing both status, but suffers from a mafia fraud if an agent can act as a verifier and a prover at the same time.

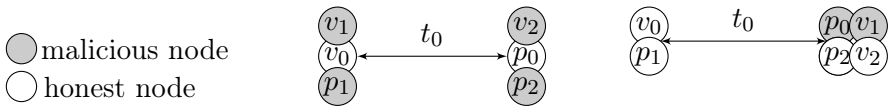


Figure 4.4: Topologies $\mathcal{T}^{t_0}_{\mathsf{MF}_{new}}$, and $\mathcal{T}^{t_0}_{\mathsf{DH}_{new}}$

4.3. Reducing the set of semi-dishonest provers

When considering terrorist fraud, another reduction result is needed to get rid of the infinite number of semi-dishonest provers that should be considered to model all the possible collusion behaviours. This result only holds for a restricted class of protocols, named *well-formed distance-bounding protocols*, that match few additional assumptions. These assumptions rely on the existence of a *unifier* for a set $\mathcal{U}$ of equations and the notion of *quasi-free* symbol of function. We first introduce these two notions before formally defining the class of protocols we consider and stating the main reduction result.

4.3.1. Preliminaries

As usual, given a convergent rewriting system, an equational theory $\mathtt{E}$ and a set $\mathcal{U}$ of equations between terms, σ is a $(\mathcal{R},\mathtt{E})$-*unifier* for $\mathcal{U}$ if $u_1\sigma{\downarrow} =_{\mathtt{E}} u_2\sigma{\downarrow}$ and both $u_1\sigma{\downarrow}$ and $u_2\sigma{\downarrow}$ are constructor terms for any $u_1 = u_2 \in \mathcal{U}$. We denote by $\mathsf{csu}_{\mathcal{R},\mathtt{E}}(\mathcal{U})$ a minimal set of $(\mathcal{R},\mathtt{E})$-unifiers for $\mathcal{U}$ which is also *complete*, i.e. such that for any σ unifier of $\mathcal{U}$, there exists $\theta \in \mathsf{csu}(\mathcal{U})$ such that $\sigma =_{\mathtt{E}} \tau \circ \theta$ for some τ. When it is clear from the context, we note $\mathsf{csu}(\mathcal{U})$ instead of $\mathsf{csu}_{\mathcal{R},\mathtt{E}}(\mathcal{U})$.

From now on, we assume that for all set of equations $\mathcal{U}$, if $\mathsf{csu}(\mathcal{U})$ exists then it is reduced to a singleton, and we note $\theta_{\mathcal{U}}$ this element. Even if this assumption seems very restrictive, it is satisfied as soon as the rewriting system contains only one rule per destructor symbol which is often verified.

Example 4.3. *Let us consider the set of equations that appear in the verifier role* $\mathtt{V}(v_0,p_0)$ *presented in Example 4.1:*

$$\begin{aligned}\mathcal{U} = \{&y_1 = \mathsf{adec}(x_1,\mathsf{sk}(v_0)),\\ &y^1_{\mathsf{check}} = \mathsf{eq}(\mathsf{proj}_1(y_1),\mathsf{check}(\mathsf{proj}_2(y_1),\mathsf{spk}(p_0))),\\ &x^0_H = \mathsf{prf}(\langle \mathsf{proj}_1(y_1), n_V\rangle),\\ &x^1_H = \mathsf{proj}_1(y_1) \oplus m_V \oplus x^0_H,\\ &y^2_{\mathsf{check}} = \mathsf{eq}(x_2,\mathsf{answer}(c,x^0_H,x^1_H)),\\ &y^3_{\mathsf{check}} = \mathsf{eq}(x_3,\mathsf{prf}(\langle \mathsf{proj}_1(y_1), n_V, m_V, c, x_2\rangle))\big\}\,.\end{aligned}$$

We have that $\mathsf{csu}(\mathcal{U}) = \{\theta_{\mathcal{U}}\}$*, where* $\theta_{\mathcal{U}}$ *is the following substitution:*

$$\begin{aligned}&\{x_1 \to \mathsf{aenc}(\langle x_{n_P}, \mathsf{sign}(x_{n_P},\mathsf{ssk}(p_0))\rangle, \mathsf{pk}(v_0)),\\ &y_1 \to \langle x_{n_P}, \mathsf{sign}(x_{n_P},\mathsf{ssk}(p_0))\rangle,\\ &x^0_H \to \mathsf{prf}(\langle x_{n_P}, n_V\rangle),\\ &x^1_H \to x_{n_P} \oplus m_V \oplus \mathsf{prf}(\langle x_{n_P}, n_V\rangle),\\ &x_2 \to \mathsf{answer}(c, \mathsf{prf}(\langle x_{n_P}, n_V\rangle), x_{n_P} \oplus m_V \oplus \mathsf{prf}(\langle x_{n_P}, n_V\rangle)),\\ &x_3 \to \mathsf{prf}(\langle x_{n_P}, n_V, m_V, c, \mathsf{answer}(c, \mathsf{prf}(\langle x_{n_P}, n_V\rangle), x_{n_P} \oplus m_V \oplus \mathsf{prf}(\langle x_{n_P}, n_V\rangle))\rangle),\\ &y^1_{\mathsf{check}} \to \mathsf{ok},\ y^2_{\mathsf{check}} \to \mathsf{ok},\ y^3_{\mathsf{check}} \to \mathsf{ok}\big\}\,.\end{aligned}$$

We can note that all the message variables used to check term equalities (i.e., y^1_{check}*,* y^2_{check} *and* y^3_{check} *) are reduced to the public constant* ok *in order to obtain a constructor term. Note that this computation is closely related to the notion of variants, or more precisely* $\mathsf{variants}_{\mathsf{C}}$*, introduced in Chapter 3. Indeed, the remaining variables are instantiated so that all the destructor symbols can be reduced.*

A symbol of function $\mathsf{f} \in \Sigma_c^+$ is *quasi-free* if it occurs neither in the equations used to generate the relation $=_{\mathtt{E}}$ nor in the right-hand side of a rewriting rule. This implies that such a symbol cannot be introduced during a normalisation.

Since such symbols does not interact with the rewriting system and the equational theory, we will be able to reduce the number semi-dishonest provers as soon as the challenge only appears under quasi-free symbols of function in the answer.

To do so, we first need the following lemma: if a term u reduces to a term $\mathsf{f}(u_1, \ldots, u_k)$ with f a quasi-free symbol of function then it must contain $\mathsf{f}(u_1, \ldots, u_k)$ as a subterm (up to a normalisation). Informally this states that, if u is the answer to a challenge, then each quasi free symbol of function that appears in its normal form $u{\downarrow}$, already appears in u.

Lemma 4.7. *Let u be a term such that $u{\downarrow} =_{\mathtt{E}} \mathsf{f}(u_1, \ldots, u_k)$ with f a quasi-free function symbol. We have that there exist $u'_1, \ldots, u'_k$ such that $\mathsf{f}(u'_1, \ldots, u'_k) \in st(u)$ and $u'_i{\downarrow} =_{\mathtt{E}} u_i$ for any $i \in \{1, \ldots, k\}$.*

4

Proof. Let u be a term, and $u{\downarrow}$ its normal form. Therefore, we have that $u \to t_1 \to t_2 \ldots \to t_n = u{\downarrow}$. We prove the result by induction on the length n of this derivation.

Base case: $n = 0$. We have that $u{\downarrow} = u$, and thus $u =_{\mathtt{E}} \mathsf{f}(u_1, \ldots, u_k)$. Since our theory is non-trivial, and since f does not occur in equations in $\mathtt{E}$, we have that there exists $u'_1, \ldots, u'_k$ such that $\mathsf{f}(u'_1, \ldots, u'_k) \in st(u)$ and $u'_i =_{\mathtt{E}} u_i$ for any $i \in \{1, \ldots, k\}$. Note that $u'_1, \ldots, u'_k$ are in normal form since u is in normal form, and thus the result holds.

Induction step. In such a case, applying our induction hypothesis, we know that there exist $u'_1, \ldots, u'_k$ such that $\mathsf{f}(u'_1, \ldots, u'_k) \in st(t_1)$ and $u'_i{\downarrow} =_{\mathtt{E}} u_i$ for any $i \in \{1, \ldots, k\}$. We denote $\mathsf{g}(v'_1, \ldots, v'_\ell) \to v'$ the rewrite rule applied at position p to rewrite u in t_1. We have that there exists a substitution θ such that $u|_p =_{\mathtt{E}} g(v'_1, \ldots, v'_k)\theta$ and $t_1 = u[v'\theta]_p$. We have that $\mathsf{f}(u'_1, \ldots, u'_n) \in st(t_1)$, and we distinguish two cases depending on the position p_{f} at which this subterm occurs in $t_1 = u[v\theta]_p$:

1. p_{f} is a position in $u[\_]_p$. In such a case, we have that either $\mathsf{f}(u'_1, \ldots, u'_n) \in st(u[\_]_p)$ (in case p_{f} is not a prefix of p); or $u|_{p_{\mathsf{f}}} = \mathsf{f}(u''_1, \ldots, u''_n)$ for some $u''_1, \ldots, u''_n$, and we have that $u|_{p_{\mathsf{f}}} \to \mathsf{f}(u'_1, \ldots, u'_n)$ with $u''_i = u'_i$ or $u''_i \to u'_i$. Therefore, we have that there exist $u''_1, \ldots, u''_n$ such that $\mathsf{f}(u'_1, \ldots, u'_n) \in st(u)$ and $u''_i{\downarrow} =_{\mathtt{E}} u_i$ for any $i \in \{1, \ldots, k\}$.

2. p_{f} is a position below p, i.e. p is a prefix of p_{f}. In such a case, since f does not occur in v' (by definition of quasi-free), we have that $\mathsf{f}(u'_1, \ldots, u'_k) \in st(x\theta)$ for some $x \in vars(v') \subseteq vars(v'_1, \ldots, v'_\ell)$. Therefore, we have that there exists $t' =_{\mathtt{E}} u|_p$ such that $\mathsf{f}(u'_1, \ldots, u'_k) \in st(t')$. Since we only consider non-trivial theory, and since f does not occur in equations in $\mathtt{E}$, we have that there exists $u''_1, \ldots, u''_k$ such that $\mathsf{f}(u''_1, \ldots, u''_k) \in st(u|_p)$ and $u''_i =_{\mathtt{E}} u'_i$ for any $i \in \{1, \ldots, k\}$. Note that $u''_1, \ldots, u''_k$ are in normal form since any subterm of $u|_p$ is in normal form. Therefore, we have that there exist $u''_1, \ldots, u''_n$ such that $\mathsf{f}(u''_1, \ldots, u''_n) \in st(u)$ and $u''_i{\downarrow} =_{\mathtt{E}} u_i$ for any $i \in \{1, \ldots, k\}$.

This concludes the proof. □

Example 4.4. *In order to illustrate Lemma 4.7, let us consider* f *a quasi-free symbol of function and the term* $u = \mathsf{adec}(\mathsf{aenc}(\mathsf{f}(\mathsf{proj}_1(\langle x, y\rangle)), \mathsf{pk}(id)), \mathsf{sk}(id))$. *We have that* u *reduces to* $\mathsf{f}(x)$, *i.e.* $u{\downarrow} = \mathsf{f}(x)$, *which already occurs in* $st(u)$ *up to normalisation. Indeed,* $\mathsf{f}(\mathsf{proj}_1(\langle x, y\rangle)) \in st(u)$ *and* $\mathsf{f}(\mathsf{proj}_1(\langle x, y\rangle)) \to \mathsf{f}(x)$.

Given an execution, we know that each input message can be forged by some recipes. Assuming that u is a response to a challenge c, we can thus deduce that there exists a recipe R which can be used to forge it. The following lemma allows us to decompose this recipe in order to obtain sub-recipes that deduce the maximal subterms of u which do not contain c.

Lemma 4.8. *Let* Φ *be a frame and* $c \in \mathcal{N}$ *such that* $c \notin st(img(\Phi))$, *and* $\Phi^+ = \Phi \cup \{\mathsf{w}_c \xrightarrow{v_0,t_0} c\}$. *Let* R *be a recipe such that* $R\Phi^+{\downarrow} =_{\mathtt{E}} u$. *Let* C *be a context of minimal size made of quasi-free public function symbols such that* $u = C[c, u_1, \ldots, u_k]$ *for some* $u_1, \ldots, u_k$ *and* c *does not occur in* $st(\{u_1, \ldots, u_k\})$. *For any* $i \in \{1, \ldots, k\}$, *we have that there exists* R_i *such that* $R_i\Phi^+{\downarrow} =_{\mathtt{E}} u_i$.

Proof. We prove this result by structural induction on the context C.

Base case: C is empty. In such a case, we have that either $k = 0$; or $k = 1$ with $u_1 = u$. In both case, the result trivially holds.

Inductive case: $C = \mathsf{f}(C_1, \ldots, C_p)$. In such a case, by minimality of C, we know that c occurs in u. We have that $u = \mathsf{f}(u'_1, \ldots, u'_p)$ and for all $i \in \{1, \ldots, p\}$ we note $\{u^i_1, \ldots, u^i_{p_i}\} \subseteq \{u_1, \ldots, u_p\}$ the set of terms involved in the sub-context C_i i.e. $C_i[u^i_1, \ldots, u^i_{p_i}] = u'_i$. Note that $\bigcup_{1 \leq i \leq p}\{u^i_1, \ldots, u^i_{p_i}\} = \{u_1, \ldots, u_p\}$.

Applying Lemma 4.7 on the term $R\Phi^+$, we deduce that there exist $v_1, \ldots, v_p$ such that $\mathsf{f}(v_1, \ldots, v_p) \in st(R\Phi^+)$ and $v_i{\downarrow} =_{\mathtt{E}} u'_i$ for any $i \in \{1, \ldots, p\}$. Now, since c occurs in u we have that there exists $i_0 \in \{1, \ldots, p\}$ such that $u^{i_0}_1 = c$. Therefore we have that c occurs in $v_{i_0}{\downarrow}$ because C_{i_0} only contains quasi-free function symbols (i.e. function symbols which do not occur in $\mathtt{E}$). By consequence we have that $\mathsf{f}(v_1, \ldots, v_p) \notin img(\Phi^+)$ and thus there exist $R_1, \ldots, R_p$ such that $\mathsf{f}(R_1, \ldots, R_p) \in st(R)$ with $R_i\Phi^+{\downarrow} =_{\mathtt{E}} u'_i$ for any $1 \leq i \leq k$.

Our induction hypothesis applies for any $i \in \{1, \ldots, p\}$ and we obtain that for all $v \in \{u^i_1, \ldots, u^i_{p_i}\}$, there exists a recipe R_v such that $R_v\Phi^+{\downarrow} =_{\mathtt{E}} v$. Considering the previous remark stating that $\bigcup_{1 \leq i \leq p}\{u^i_1, \ldots, u^i_{p_i}\} = \{u_1, \ldots, u_k\}$, this allows us conclude the proof. □

Example 4.5. *In order to illustrate Lemma 4.8, we consider the frame* $\Phi = \{\mathsf{w}_0 \xrightarrow{a,t_a} u_1, \mathsf{w}_1 \xrightarrow{a,t_a} u_2\}$ *and the term* $u = \mathsf{f}(c, \langle u_1, u_2\rangle)$ *such that* $R(\Phi \uplus \{\mathsf{w}_c \xrightarrow{v_0,t_v} c\}){\downarrow} = u$ *with* $R = \mathsf{f}(\mathsf{w}_c, \langle \mathsf{w}_1, \mathsf{w}_2\rangle)$. *Thanks to Lemma 4.8, we have that* R *contains a subterm which deduces* $\langle u_1, u_2\rangle$. *Indeed, such a recipe is for example* $R' = \langle \mathsf{w}_1, \mathsf{w}_2\rangle$. *This reasoning may be applied to more complex terms whenever all the symbols on top of* c *in* u *are quasi-free.*

4.3.2. Most general semi-dishonest prover

We have now all the material needed to define the class of protocols for which we will be able to reduce the number of semi-dishonest provers that must be considered.

Definition 4.3. *A* well-formed distance-bounding *protocol is a protocol* $\mathcal{P}_{\mathsf{db}}$ *such that:*

(i) The verifier and prover roles, resp. $\mathtt{V}(\mathtt{x}_0,\mathtt{x}_1)$ *and* $\mathtt{P}(\mathtt{x}_0,\mathtt{x}_1)$ *have the form:*

- $\mathtt{V}(\mathtt{x}_0,\mathtt{x}_1) = \mathtt{block}_V.\mathtt{new}\ c.\mathtt{reset}.\mathtt{out}(c).\mathtt{in}^{<2\times t_0}(x).\mathtt{block}'_V$*; and*
- $\mathtt{P}(\mathtt{x}_0,\mathtt{x}_1) = \mathtt{block}_P.\mathtt{in}(y_c).\mathtt{out}(u).\mathtt{block}'_P$

where $\mathtt{block}_X$ *and* $\mathtt{block}'_X$ *with* $X \in \{V,P\}$ *is a sequence of actions without reset and guarded input instructions. Moreover, we assume that* $\mathtt{out}(c)$ *(resp.* $\mathtt{in}(y_c)$*) corresponds to the* $i_0{}^{th}$ *communication action of* $\mathtt{V}(\mathtt{x}_0,\mathtt{x}_1)$ *(resp.* $\mathtt{P}(\mathtt{x}_0,\mathtt{x}_1)$*) for some* i_0*.*

(ii) $(\lfloor\widetilde{\mathtt{V}}\rfloor^0_{v_0} \uplus \lfloor\widetilde{\mathtt{P}}\rfloor^0_{p_0}; \emptyset; 0) \xrightarrow{\mathsf{tr}}_{\mathcal{T}^0_{\mathsf{basic}}} (\lfloor 0\rfloor^0_{v_0} \uplus \lfloor 0\rfloor^0_{p_0}; \Phi; 0)$ *with:*

$$\mathsf{tr} = \begin{cases} (a_1,\mathtt{out}(m_1)).(b_1,\mathtt{in}(m_1))\ldots(a_{i_0-1},\mathtt{out}(m_{i_0-1})).(b_{i_0-1},\mathtt{in}(m_{i_0-1})) \\ (v_0,\mathtt{out}(m_{i_0})).(p_0,\mathtt{in}(m_{i_0})).(p_0,\mathtt{out}(m_{i_0+1})).(v_0,\mathtt{in}^{<2\cdot t_0}(m_{i_0+1})) \\ (a_{i_0+2},\mathtt{out}(m_{i_0+2})).(b_{i_0+2},\mathtt{in}(m_{i_0+2}))\ldots(a_n,\mathtt{out}(m_n)).(b_n,\mathtt{in}(m_n)) \end{cases}$$

up to τ *actions, and* $\{a_i,b_i\} = \{v_0,p_0\}$ *for any* $i \in \{1,\ldots,n\} \setminus \{i_0,i_0+1\}$*. The processes* $\widetilde{\mathtt{V}}$ *(resp.* $\widetilde{\mathtt{P}}$*) corresponds to* $\mathtt{V}(v_0,p_0)$ *(resp.* $\mathtt{P}(p_0,v_0)$*) in which* `new` *commands are removed, and* $\mathcal{T}^0_{\mathsf{basic}}$ *is the topology only composed of agents* v_0,p_0 *honest and located at the same place.*

(iii) Let $\mathcal{U} = \{x = u \mid \text{"}\mathtt{let}_{\mathtt{mess}}\ x = u\ \mathtt{in}\text{ "}$ *occurs in* $\mathtt{V}(v_0,p_0)\}$ *and* $\{\theta_{\mathcal{U}}\}$ *its complete set of unifiers. We assume that* $(x_1,\ldots,x_k)\theta_{\mathcal{U}}{\downarrow}\sigma =_{\mathtt{E}} m_{i_1},\ldots,m_{i_k}$ *where* $x_1,\ldots,x_k$ *are the variables occurring in input in the role* $\mathtt{V}_0(v_0,p_0)$*,* $i_1,\ldots,i_k$ *are the indices among* $1,\ldots,n$ *corresponding to input performed by* v_0*, and* σ *is a bijective renaming from variables to* $bn(\mathtt{P}(p_0,v_0))$*.*

(iv) We assume the existence of a context C *made of quasi-free public function symbols such that* $u = C[y_c,u_1,\ldots,u_l]$*, and* y_c *does not occur in* $u_1,\ldots,u_l$*.*

The first condition puts some syntactic restrictions on the form of each role. Indeed, we assume that a well-formed DB protocol is a two-party protocol with rather simple roles: the rapid phase of the verifier role consists in a single challenge/response exchange. The second condition assumes that if no attacker interferes, these two roles together will execute until the end. For sake of simplicity, we consider in item (*ii*) instances of the roles in which bound names are not freshened. This is possible because we consider only one instance of each role (and they do not share bound names). The third condition gives us a constraint about exchanged messages. Even if it may seem restrictive this is always verified. Otherwise, it would mean that some terms that are exchanged are never checked, i.e. are useless. Finally, the fourth condition is used to ensure that there

exists at least one semi-dishonest prover. This semi-dishonest prover will output the terms $u_1, \ldots, u_l$ in advance to let his accomplice compute (as indicated by C) the answer to the challenge from $u_1, \ldots, u_l$ and the challenge c' he will receive from the verifier. Actually, the best strategy for the semi-dishonest prover will consist in considering $C_{\mathtt{V},\mathtt{P}}$ the smallest context (in terms of number of symbols) satisfying the requirements.

Example 4.6. *The SPADE protocol described in Example 4.1 is a well-formed DB protocol. The roles* $\mathtt{V}(\mathsf{x}_0, \mathsf{x}_1)$ *and* $\mathtt{P}(\mathsf{x}_0, \mathsf{x}_1)$ *as described in Example 4.1 satisfy the requirements stated in item (i). We may note that* $i_0 = 3$*. Regarding item (ii), we can exhibit the following trace (up to* τ *actions):*

$$\begin{aligned}\mathsf{tr} = {} & (p_0, \mathtt{out}(m_1)).(v_0, \mathtt{in}(m_1)).(v_0, \mathtt{out}(m_2)).(p_0, \mathtt{in}(m_2)).(v_0, \mathtt{out}(c)).(p_0, \mathtt{in}(c)) \\ & (p_0, \mathtt{out}(m_4)).(v_0, \mathtt{in}^{<2\cdot t_0}(m_4)).(p_0, \mathtt{out}(m_5)).(v_0, \mathtt{in}(m_5))\end{aligned}$$

where the messages m_1*,* m_2*,* m_3*,* m_4*, and* m_5 *are as follows:*

$$\begin{aligned}m_1 &= \mathsf{aenc}(\langle n_P, \mathsf{sign}(n_P, \mathsf{ssk}(p_0))\rangle, \mathsf{pk}(p_0)),\\ m_2 &= \langle m_V, n_V\rangle,\\ m_4 &= \mathsf{answer}(c, \mathsf{prf}(\langle n_P, n_V\rangle), n_P \oplus m_V \oplus \mathsf{prf}(\langle n_P, n_V\rangle)), \text{ and}\\ m_5 &= \mathsf{prf}(\langle n_P, n_V, m_V, c, \mathsf{answer}(c, \mathsf{prf}(\langle n_P, n_V\rangle), n_P \oplus m_V \oplus \mathsf{prf}(\langle n_P, n_V\rangle))\rangle).\end{aligned}$$

The set $\mathcal{U}$ *described in item (iii) is exactly the same as the one presented in Example 4.3, and thus we have:*

$$\begin{aligned}\theta_{\mathcal{U}} = \{ & x_1 \to \mathsf{aenc}(\langle x_{n_P}, \mathsf{sign}(x_{n_P}, \mathsf{ssk}(p_0))\rangle, \mathsf{pk}(v_0)),\\ & y_1 \to \langle x_{n_P}, \mathsf{sign}(x_{n_P}, \mathsf{ssk}(p_0))\rangle,\\ & x_H^0 \to \mathsf{prf}(\langle x_{n_P}, n_V\rangle),\\ & x_H^1 \to x_{n_P} \oplus m_V \oplus \mathsf{prf}(\langle x_{n_P}, n_V\rangle),\\ & x_2 \to \mathsf{answer}(c, \mathsf{prf}(\langle x_{n_P}, n_V\rangle), x_{n_P} \oplus m_V \oplus \mathsf{prf}(\langle x_{n_P}, n_V\rangle)),\\ & x_3 \to \mathsf{prf}(\langle x_{n_P}, n_V, m_V, c, \mathsf{answer}(c, \mathsf{prf}(\langle x_{n_P}, n_V\rangle), x_{n_P} \oplus m_V \oplus \mathsf{prf}(\langle x_{n_P}, n_V\rangle))\rangle),\\ & y^1_{\mathsf{check}} \to \mathsf{ok},\ y^2_{\mathsf{check}} \to \mathsf{ok},\ y^3_{\mathsf{check}} \to \mathsf{ok}\big\}.\end{aligned}$$

The substitution σ *is reduced to* $\{x_1 \to n_P\}$*. Finally, for item (iv), we can exhibit the context* $C = \mathsf{answer}(c, u_1, u_2)$ *where* u_1 *and* u_2 *are the terms outputted by the semi-dishonest prover described in Example 2.11. One may note that this context is the smallest one that satisfies the requirements. Hence we have that* $C = C_{\mathtt{V},\mathtt{P}}$*.*

Up to now, according to Definition 2.6 (and Corollary 4.2) we had to consider all the possible semi-dishonest provers when verifying terrorist fraud resistance. Restricting our analysis to well-formed distance-bounding protocols, we are now able to define the *most general semi-dishonest prover* and prove that it is sufficient to focus our analysis on it.

Definition 4.4. *Let $\mathcal{P}_{\mathtt{db}}$ be a well-formed DB protocol. Following the notations of Definition 4.3, we note* $\mathtt{P}^*$ *the following process:*

$$\mathtt{P}^* = \big(\mathtt{block}_P.\mathtt{out}(u_1)\ldots\mathtt{out}(u_l).\mathtt{in}(y_c).\mathtt{out}(u).\mathtt{block}'_P\big)\{\mathtt{x}_0 \mapsto p_0, \mathtt{x}_1 \mapsto v_0\}$$

where $u_1, \ldots, u_l$ are the terms such that $u = C_{\mathtt{V},\mathtt{P}}[y_c, u_1, \ldots, u_l]$.

Example 4.7. *If we consider the semi-dishonest prover presented in Example 2.12, one can see that it corresponds to the process* $\mathtt{P}^*$ *defined above. One may note that this process is actually a semi-dishonest prover for $\mathcal{P}_{\mathtt{db}}$. This will be formalised through the following lemma.*

4

The behaviour of the most general semi-dishonest prover consists in providing to his accomplice all the material needed to pass the rapid phase just before it starts. For this purpose, he computes and sends the maximal sub-terms of u that do not contain the challenge y_c. His accomplice is then able to re-construct the response to the challenge using the context $C_{\mathtt{V},\mathtt{P}}$ composed of public function symbols.

As explicitly assumed in the speech up to now, we shall prove that the most general semi-dishonest prover $\mathtt{P}^*$ is a semi-dishonest prover following Definition 2.5, i.e., put together with $\lfloor \mathtt{V}(v_0, p_0) \rfloor_{v_0}^0$ the two processes can be fully executed. This result is immediate from the definition of a well-formed distance-bounding protocol thanks to the close correspondence between $\mathtt{P}^*$ and $\mathtt{P}(p_0, v_0)$.

Lemma 4.9. *Let $(\mathtt{V}, \mathtt{P})$ be a well-formed DB protocol. The most general semi-dishonest process* $\mathtt{P}^*$ *together with its associated frame Φ^* is a semi-dishonest prover. Moreover, we can assume that the trace* $\mathtt{tr}^*$ *witnessing this fact is such that :*

$$\mathtt{tr}^* = \begin{cases} (a_1, \mathtt{out}(m_1)).(b_1, \mathtt{in}(m_1)) \ldots (a_{i_0-1}, \mathtt{out}(m_{i_0-1})).(b_{i_0-1}, \mathtt{in}(m_{i_0-1})). \\ (p_0, \mathtt{out}(m^1_{i_0+1})).\ldots.(p_0, \mathtt{out}(m^l_{i_0+1})). \\ (v_0, \mathtt{out}(m_{i_0})).(v_0, \mathtt{in}^{<2\times t_0}(m_{i_0+1})) \\ (p_0, \mathtt{in}(m_{i_0})).(p_0, \mathtt{out}(m_{i_0+1})). \\ (a_{i_0+2}, \mathtt{out}(m_{i_0+2})).(b_{i_0+2}, \mathtt{in}(m_{i_0+2})) \ldots (a_n, \mathtt{out}(m_n)).(b_n, \mathtt{in}(m_n)) \end{cases}$$

up to τ actions where:

- $\{a_i, b_i\} = \{v_0, p_0\}$ *for any* $i \in \{1, \ldots, n\} \setminus \{i_0, i_0 + 1\}$*;*
- $m_{i_0+1} = C_{\mathtt{V},\mathtt{P}}[m_{i_0}, m^1_{i_0+1}, \ldots, m^l_{i_0+1}]$
- $(x_1, \ldots, x_k)\theta_{\mathcal{U}}{\downarrow}\sigma =_{\mathtt{E}} m_{i_1}, \ldots, m_{i_k}$ *where $x_1, \ldots, x_k$ are the variables occurring in input in the role* $\mathtt{V}(v_0, p_0)$ *and $i_1, \ldots, i_k$ are the indices among $1, \ldots, n$ corresponding to input performed by v_0, σ is a bijective renaming from variables to $bn(\mathtt{P}^*)$, and $\mathcal{U} = \{x = u \mid$ "$\mathtt{let}_{\mathtt{mess}}\ x = u\ \mathtt{in}$" *occurs in* $\mathtt{V}(v_0, p_0)\}$*. This equality holds up to a bijective renaming of names freshly generated along the execution.*

We note Φ^* the initial frame associated to the most general semi-dishonest prover, i.e. the frame resulting from the execution of $\mathtt{tr}^*$ *in which all the annotated times are set to* 0.

Proof. This proof strongly relies on Definition 4.3. First, remark that $\mathtt{tr}^*$ corresponds to the trace $\mathtt{tr}$ in which the extra outputs of $\mathtt{P}^*$ are executed just before the $i_0{}^{\text{th}}$ communication action i.e. the output of the challenge; and the answer to the challenge received by v_0 is anticipated. We show that this sequence of actions $\mathtt{tr}^*$ is an execution w.r.t. our semantics:

- 1^{st} line of actions: The actions can be executed following our semantics applying a TIM rule with a delay $\delta = t_0$ before each input. Indeed this delay enables the agent b_i to receive the message m_i sent by a_i. Moreover, since there is no guarded input the IN rule always applies.
- 2^{nd} line of actions: It only contains outputs and thus can trivially be executed.
- 3^{rd} line of actions: Before executing the output, we apply a TIM rule to let available all the previous messages (including $m^1_{i_0+1}, \ldots, m^l_{i_0+1}$) for the malicious agent p. Since $C_{\mathtt{V},\mathtt{P}}$ only contains public symbols of functions (otherwise there is a contradiction with item (iv) of Definition 4.3), we have that $R = C_{\mathtt{V},\mathtt{P}}[\mathsf{w}_{i_0}, \mathsf{w}^1_{i_0+1}, \ldots, \mathsf{w}^l_{i_0+1}]$ where w_{i_0} is the frame variable binding m_{i_0} and $\mathsf{w}^j_{i_0+1}$, $(1 \leq j \leq l)$ is the frame variable binding $m^j_{i_0+1}$, is a recipe deducing m_{i_0+1}. Finally the guarded input can be executed because w.l.o.g. we may assume that the reset action has been made right before the output of m_{i_0}.
- 4^{th} and 5^{th} lines of actions: These actions can be executed for the same reason as the first line applying a TIM rule with a delay $\delta = t_0$ before each input.

By construction we have that the first and the third item holds. Let us show that the second one holds too. Denoting Φ the current frame when executing the guarded input, we have that:

$$\begin{aligned} R\Phi\downarrow &= C_{\mathtt{V},\mathtt{P}}[\mathsf{w}_{i_0}, \mathsf{w}^1_{i_0+1}, \ldots, \mathsf{w}^l_{i_0+1}]\Phi\downarrow \\ &= C_{\mathtt{V},\mathtt{P}}[\mathsf{w}_{i_0}\Phi, \mathsf{w}^1_{i_0+1}\Phi, \ldots, \mathsf{w}^l_{i_0+1}\Phi]\downarrow \\ &= C_{\mathtt{V},\mathtt{P}}[m_{i_0}, m^1_{i_0+1}, \ldots, m^l_{i_0+1}]\downarrow. \end{aligned}$$

Note that, by construction, since u (in Definition 4.3) is only composed of constructor terms, the context $C_{\mathtt{V},\mathtt{P}}$ cannot contain destructor symbols (equality in item (iv) holds without normalisation). Hence we have that $m_{i_0+1} = R\Phi\downarrow = C_{\mathtt{V},\mathtt{P}}[m_{i_0}, m^1_{i_0+1}, \ldots, m^l_{i_0+1}]$. This concludes the proof. □

A noteworthy point is the strong correspondence that exists between Φ^* and any frame $\Phi_{\mathtt{sd}}$ associated to an arbitrary semi-dishonest prover $\mathtt{P}_{\mathtt{sd}}$. Informally we prove that any semi-dishonest prover must disclose, at least, as much information as $\mathtt{P}^*$. The following lemma formalises this property: the two traces match up to a substitution σ that models the extra-leakage on $\mathtt{P}_{\mathtt{sd}}$ side.

Proposition 4.3. *Let* $(\mathtt{V}, \mathtt{P})$ *be a well-formed DB protocol,* $\mathtt{P}^*$ *be its most general semi-dishonest prover with* Φ^* *its associated frame and* $\mathtt{tr}^*$ *a trace witnessing this fact (as given*

in Lemma 4.9). Let $\mathtt{P_{sd}}$ *be a semi-dishonest prover for* $(\mathtt{V},\mathtt{P})$ *together with its associated frame* $\Phi_{\mathtt{sd}}$*, and* $\mathtt{tr}$ *be a trace witnessing this fact.*

We have that there exists a substitution $\sigma : \mathcal{N} \to \mathcal{T}(\Sigma_c^+, \mathcal{N} \cup \mathcal{A})$ *from names freshly generated by* $\mathtt{P}^*$ *to constructor terms such that:*

(i) if $(v_0, \mathtt{in}^\star(u)) \in \mathtt{tr}^*$ *(resp.* $(v_0, \mathtt{in}^{<t}(u)) \in \mathtt{tr}^*$*), then* $(v_0, \mathtt{in}^\star(u\sigma)) \in \mathtt{tr}$ *(resp.* $(v_0, \mathtt{in}^{<t}(u\sigma)) \in \mathtt{tr}$*);*

(ii) if $(a, \mathtt{out}(u)) \in \mathtt{tr}^*$ *for some* $a \in \mathcal{A}$*, then* $R\Phi_{\mathtt{sd}}\downarrow =_{\mathtt{E}} u\sigma$ *for some recipe* R*.*

Proof. Along an execution, variables occurring in input as well as those occurring in a let instruction are instantiated. We denote by $\tau_{\mathtt{tr}}$ (resp $\tau_{\mathtt{tr}^*}$) the substitution associated to the given execution $\mathtt{tr}$ (resp. $\mathtt{tr}^*$).

Relying on Lemma 4.9, we note $\tau : \mathcal{X} \to \mathcal{N}$ the bijective renaming from variables to names freshly generated by $\mathtt{P}^*$ such that $(x_1, \ldots, x_k)\theta_{\mathcal{U}}\downarrow\tau =_{\mathtt{E}} m_{i_1}, \ldots, m_{i_k}$ where $x_1, \ldots, x_k$ are the variables occurring in input in $\mathtt{V}(v_0, p_0)$ and $m_{i_1}, \ldots, m_{i_k}$ are the inputted terms. By definition of $\theta_{\mathcal{U}}$ we have that there exists σ' such that $x_j\tau_{\mathtt{tr}}\downarrow =_{\mathtt{E}} x_j\theta_{\mathcal{U}}\sigma'\downarrow$ for any $j \in \{1, \ldots, k\}$. Moreover, for any $j \in \{1, \ldots, k\}$ we have that $x_j\tau_{\mathtt{tr}}$ is a message (and thus a constructor term) and we have also that $x_j\theta_{\mathcal{U}}\downarrow$ is a constructor term (by definition of $\theta_{\mathcal{U}}$). Thus, for any $j \in \{1, \ldots, k\}$ we have that $x_j\tau_{\mathtt{tr}} =_{\mathtt{E}} x_j\theta_{\mathcal{U}}\downarrow\sigma'$ for any $j \in \{1, \ldots, k\}$.

We consider $\sigma = \tau^{-1}\sigma'$ and establish each item separately.

Item (i): We consider $\mathtt{V}(v_0, p_0) = \alpha_1.\ldots.\alpha_k$. First, because $(v_0, \mathtt{in}^\star(u)) \in \mathtt{tr}^*$, we have that there exists $j \in \{1, \ldots, k\}$ such that $\alpha_{i_j} = \mathtt{in}^\star(x_j)$ and $x_j\tau_{\mathtt{tr}^*} = u\ (= m_{i_j})$. By definition of τ, we have that $u = x_j\theta_{\mathcal{U}}\downarrow\tau$.

Then, we know that the process has been entirely executed in $\mathtt{tr}$ and therefore there exists $(v_0, \mathtt{in}^\star(u')) \in \mathtt{tr}$ such that $u' = x_j\tau_{\mathtt{tr}}$. By definition of σ', we have $u' =_{\mathtt{E}} x_j\theta_{\mathcal{U}}\downarrow\sigma'$. By consequence we obtain that $u' = x_j\tau_{\mathtt{tr}} =_{\mathtt{E}} x_j\theta_{\mathcal{U}}\downarrow\sigma' = (u\tau^{-1})\sigma' = u\sigma$. This concludes the proof of item (i).

Item (ii): We have that $(a, \mathtt{out}(u)) \in \mathtt{tr}^*$. We distinguish several cases depending on the origin of this output.

In case $a = v_0$, it is an immediate corollary of item (i). Indeed, given exec^* (resp. exec) an execution which matches with the trace $\mathtt{tr}^*$ (resp. $\mathtt{tr}$) and witnesses the fact that $\mathtt{P}^*$ (resp. $\mathtt{P_{sd}}$) is a semi-dishonest prover, we can prove by induction on the length of $\mathtt{tr}$ that for each configuration $\mathcal{K}$ in exec, there exists a configuration $\mathcal{K}^*$ in exec^* such that if we note V^* the process executed by v_0 in $\mathcal{K}^*$ then $V^*\sigma$ is the process executed by v_0 in $\mathcal{K}$.

Now, we assume that $a = p_0$, and we distinguish two cases depending on whether $u = m^j_{i_0+1}$ $(1 \leq j \leq l)$ or not. If not, relying on Lemma 4.9, we have that $(v_0, \mathtt{in}(u)) \in \mathtt{tr}^*$ (or $(v_0, \mathtt{in}^{<t}(u)) \in \mathtt{tr}^*$), and since item (i) holds we have now that $(v_0, \mathtt{in}(u\sigma)) \in \mathtt{tr}$ (or $(v_0, \mathtt{in}^{<t}(u\sigma)) \in \mathtt{tr}$). We can thus deduce that exists a recipe R such that $R\Phi_{\mathtt{sd}}\downarrow =_{\mathtt{E}} u\sigma$.

Now, we assume that $u = m^j_{i_0+1}$ for some $j \in \{1, \ldots, l\}$. Thanks to Lemma 4.9, we know that $(v_0, \mathtt{in}^{<2\cdot t_0}(m_{i_0+1})) \in \mathtt{tr}^*$ and $m_{i_0+1} = C_{\mathtt{V},\mathtt{P}}[m_{i_0}, m^1_{i_0+1}, \ldots, m^l_{i_0+1}] = x\theta_{\mathcal{U}}\downarrow\tau$

where x in the variable occurring in the guarded input in $\mathtt{V}(v_0, p_0)$. According to item (i), we have that $(v_0, \mathtt{in}^{<2t_0}(m_{i_0+1}\sigma)) \in \mathsf{tr}$. Moreover, following the hypotheses on the structure of $\mathtt{V}(v_0, p_0)$, we know that there is a unique output in tr that is executed by v_0 before this guarded input and that contains the challenge m_{i_0} (note that $m_{i_0} = m_{i_0}\sigma$ because σ only applies on names generated by $\mathtt{P}^*$). In addition, p_0 cannot receive the challenge soon enough to make an output containing m_{i_0} available to fill the guarded input. Indeed, assume that it was possible, and let $t_{\mathtt{reset}}$ (resp. $t_{\mathtt{out}}$ and $t_{\mathtt{in}}$) the time when the `reset` action (resp. output of the challenge, reception of the guarded input) is executed in tr then we have that:

$$t_{\mathtt{in}} \geq t_{\mathtt{out}} + 2 \times \mathsf{Dist}_{\mathcal{T}_s^{t_0}}(v_0, p_0) \geq t_{\mathtt{reset}} + 2 \times \mathsf{Dist}_{\mathcal{T}_s^{t_0}}(v_0, p_0) = t_{\mathtt{reset}} + 2 \times t_0.$$

This is in contradiction with the constraint imposed by the guarded input: $t_{\mathtt{in}} - t_{\mathtt{reset}} < 2 \times t_0$. Finally, denoting Φ^+ the current frame when executing the guarded input, we deduce that $\Phi^+ = \Phi \cup \{\mathsf{w} \xrightarrow{v_0, t_{\mathtt{out}}} m_{i_0}\}$ for some sub-frame Φ such that $m_{i_0} \notin st(img(\Phi))$ and there exists a recipe R such that $R\Phi^+\downarrow =_{\mathtt{E}} m_{i_0+1}\sigma$. Lemma 4.8 applies and we conclude that there exists a recipe R_u such that $R_u\Phi^+\downarrow =_{\mathtt{E}} m^j_{i_0+1}\sigma = u\sigma$ and thus $R_u\Phi_{\mathtt{sd}}\downarrow =_{\mathtt{E}} u\sigma$ since $\Phi_{\mathtt{sd}}$ contains Φ^+. This concludes the proof. □

4.3.3. Main result

We are now able to state and prove the main reduction result which allows us to get rid of the universal quantification over the semi-dishonest prover by focusing on the most general one.

Theorem 4.3. *Let $\mathcal{I}_0$ be a template, $\mathcal{P}_{\mathsf{db}}$ be a well-formed DB protocol, and t_0 be a threshold. Let $\mathtt{P}^*$ be the most general semi-dishonest prover of $\mathcal{P}_{\mathsf{db}}$ together with its associated frame Φ^*. We have that $\mathcal{P}_{\mathsf{db}}$ is terrorist fraud resistant w.r.t. t_0-proximity if, and only if, there exists a topology $\mathcal{T} = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}_0, v_0, p_0) \in \mathcal{C}^{t_0}_{\mathsf{MF}}$ and a valid initial configuration $\mathcal{K}_0$ for $\mathcal{P}_{\mathsf{db}}$ w.r.t. $\mathcal{T}$ and $\Phi^* \cup \Phi^{\mathcal{T}}_{\mathcal{I}_0}$ such that:*

$$\mathcal{K}_0 \xrightarrow{\mathsf{tr}}_{\mathcal{T}} (\lfloor \mathsf{end}(v_0, p_0) \rfloor^{t'}_{v_0} \uplus \mathcal{P}; \Phi; t).$$

Proof. The direct implication is trivial since $\mathtt{P}^*$ is a semi-dishonest prover (Lemma 4.9). We concentrate on the other implication, and we have to show that the property holds for any semi-dishonest prover. Let $\mathtt{P}_{\mathtt{sd}}$ be a semi-dishonest prover for $(\mathtt{V}, \mathtt{P})$ with its associated frame $\Phi_{\mathtt{sd}}$, and $\mathsf{tr}_{\mathtt{sd}}$ be a trace witnessing this fact. We denote $\mathtt{P}^*$ the most general semi-dishonest prover, Φ^* its associated frame, and tr^* a trace witnessing this fact.

Applying Proposition 4.3, there exists a substitution $\sigma : \mathcal{N} \to \mathcal{T}(\Sigma_c^+, \mathcal{N} \cup \mathcal{A})$, from names freshly generated by $\mathtt{P}^*$ in tr^* to constructor terms such that:

(i) if $(v_0, \mathtt{in}(u)) \in \mathsf{tr}^*$, then $(v_0, \mathtt{in}(u\sigma)) \in \mathsf{tr}_{\mathtt{sd}}$.

(ii) if $(a, \mathtt{out}(u)) \in \mathsf{tr}^*$ for some $a \in \mathcal{A}_0$, then $R\Phi_{\mathtt{sd}}\downarrow =_{\mathtt{E}} u\sigma$ for some recipe R.

By hypothesis, there exist a topology $\mathcal{T} \in \mathcal{C}^{t_0}_{\mathsf{MF}}$ and a valid initial configuration $\mathcal{K}_0$ for $(\mathtt{V}, \mathtt{P})$ w.r.t. $\mathcal{T}$ and $\Phi^* \cup \Phi^{\mathcal{T}}_{\mathcal{I}_0}$ such that

$$\mathcal{K}_0 = (\mathcal{P}_{\mathsf{init}}; \Phi^{\mathcal{T}}_{\mathcal{I}_0} \cup \Phi^*; 0) \xrightarrow{\mathsf{tr}}_{\mathcal{T}} (\lfloor \mathsf{end}(v_0, p_0) \rfloor^{t'}_{v_0} \uplus \mathcal{P}; \Phi^{\mathcal{T}}_{\mathcal{I}_0} \cup \Phi^* \cup \Phi_{\mathtt{out}}; t)$$

for some frame $\Phi_{\mathtt{out}}$. Without loss of generality, we may assume that the topology $\mathcal{T} = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}_0, v_0, p_0)$ is such that $\mathcal{M}_0 \neq \emptyset$. Otherwise, we add such a malicious agent, and the trace remains executable. Applying the substitution σ along this execution, we obtain a valid execution (remember that our calculus does not feature else branches):

$$(\mathcal{P}_{\mathsf{init}}\sigma; \Phi^{\mathcal{T}}_{\mathcal{I}_0}\sigma \cup \Phi^*\sigma; 0) \xrightarrow{\mathsf{tr}\sigma}_{\mathcal{T}} (\lfloor \mathsf{end}(v_0, p_0) \rfloor^{t'}_{v_0} \uplus \mathcal{P}\sigma; \Phi^{\mathcal{T}}_{\mathcal{I}_0}\sigma \cup \Phi^*\sigma \cup \Phi_{\mathtt{out}}\sigma; t).$$

Actually, since names occurring in $dom(\sigma)$ are names freshly generated by $\mathtt{P}^*$, we have that $\mathcal{P}_{\mathsf{init}}\sigma = \mathcal{P}_{\mathsf{init}}$, $\Phi^{\mathcal{T}}_{\mathcal{I}_0}\sigma = \Phi^{\mathcal{T}}_{\mathcal{I}_0}$, and therefore, we have that:

$$(\mathcal{P}_{\mathsf{init}}; \Phi^{\mathcal{T}}_{\mathcal{I}_0} \cup \Phi^*\sigma; 0) \xrightarrow{\mathsf{tr}\sigma}_{\mathcal{T}} (\lfloor \mathsf{end}(v_0, p_0) \rfloor^{t'}_{v_0} \uplus \mathcal{P}\sigma; \Phi^{\mathcal{T}}_{\mathcal{I}_0} \cup \Phi^*\sigma \cup \Phi_{\mathtt{out}}\sigma; t).$$

Finally, from item (ii), we have that for any $u \in img(\Phi^*)$, there exists a recipe R such that $R\Phi_{\mathtt{sd}}\downarrow =_{\mathtt{E}} u\sigma$. We can thus deduce that for any term v and recipe R_v such that $R_v(\Phi^{\mathcal{T}}_{\mathcal{I}_0} \cup \Phi^*\sigma)\downarrow =_{\mathtt{E}} v$ we have that there exists R'_v such that $R'_v(\Phi^{\mathcal{T}}_{\mathcal{I}_0} \cup \Phi_{\mathtt{sd}})\downarrow =_{\mathtt{E}} v$. Indeed, for any $(\mathsf{w} \xrightarrow{a_{\mathsf{w}}, t_{\mathsf{w}}} u_{\mathsf{w}}) \in \Phi^*$ there exists a recipe R_{w} such that $R_{\mathsf{w}}\Phi_{\mathtt{sd}}\downarrow =_{\mathtt{E}} u_{\mathsf{w}}\sigma$. We can thus define $R'_v = R_v\{\mathsf{w} \to R_{\mathsf{w}} \mid \mathsf{w} \in dom(\Phi^*)\}$.

Starting by applying a TIM rule with a delay δ equal to twice the greatest distance between two agents in $\mathcal{T}$, we have:

$$(\mathcal{P}_{\mathsf{init}}; \Phi^{\mathcal{T}}_{\mathcal{I}_0} \cup \Phi_{\mathtt{sd}}; 0) \xrightarrow{\mathsf{tr}\sigma}_{\mathcal{T}} (\lfloor \mathsf{end}(v_0, p_0) \rfloor^{t'}_{v_0} \uplus \tilde{\mathcal{P}}; \Phi^{\mathcal{T}}_{\mathcal{I}_0} \cup \Phi_{\mathtt{sd}} \cup \Phi'_{\mathtt{out}}\sigma; t + \delta).$$

with $\phi'_{\mathtt{out}} = \{\mathsf{w} \xrightarrow{a, t+\delta} u\sigma \mid \mathsf{w} \xrightarrow{a,t} u \in \Phi_{\mathtt{out}}\}$. The delay δ enables a dishonest agent (there is one by assumption) to build any term occurring in $\Phi^*\sigma$ from $\Phi_{\mathtt{sd}}$. Remember that even if this delay modifies the value of the local clocks the trace is still executable thanks to Proposition 4.1 because each guarded input is preceded by a `reset` command in the two roles $\mathtt{V}(\mathsf{x}_0, \mathsf{x}_1)$ and $\mathtt{P}(\mathsf{x}_0, \mathsf{x}_1)$. This concludes the proof. □

The following corollary is immediate putting together Theorem 4.3 and Theorem 4.1. It shows that when checking for terrorist fraud resistance, it is sufficient to focus on a particular semi-dishonest prover and a particular topology.

Corollary 4.3. *Let $\mathcal{I}_0$ be a template, $(\mathtt{V}, \mathtt{P})$ be a well-formed DB protocol, and $t_0 \in \mathbb{R}_+$ be a threshold.. Let Φ^* be the frame associated to the most general semi-dishonest prover of $(\mathtt{V}, \mathtt{P})$. We have that $(\mathtt{V}, \mathtt{P})$ is terrorist fraud resistant w.r.t. t_0-proximity if, and only if, there exists a valid initial configuration $\mathcal{K}_0$ for $(\mathtt{V}, \mathtt{P})$ w.r.t. $\mathcal{T}^{t_0}_{\mathsf{MF}}$ and $\Phi^* \cup \Phi^{\mathcal{T}^{t_0}_{\mathsf{MF}}}_{\mathcal{I}_0}$ such that:*

$$\mathcal{K}_0 \xrightarrow{\mathsf{tr}}_{\mathcal{T}^{t_0}_{\mathsf{MF}}} (\lfloor \mathsf{end}(v_0, p_0) \rfloor^{t'}_{v_0} \uplus \mathcal{P}; \Phi; t).$$

Proof. We consider the first implication. If $(\mathtt{V},\mathtt{P})$ is terrorist fraud resistant w.r.t. t_0-proximity then there exist a topology $\mathcal{T} \in \mathcal{C}_{\mathsf{MF}}^{t_0}$ and a valid initial configuration $\mathcal{K}$ for $(\mathtt{V},\mathtt{P})$ w.r.t. $\mathcal{T}$ and $\Phi^* \cup \Phi_{\mathcal{I}_0}^{\mathcal{T}}$ such that:

$$\mathcal{K} \xrightarrow{\mathsf{tr}}_{\mathcal{T}} (\lfloor \mathsf{end}(v_0,p_0) \rfloor_{v_0}^{t_v} \uplus \mathcal{P}; \Phi; t)$$

Actually, this should be satisfied for any semi-dishonest prover and especially $\mathtt{P}^*$.

By definition of a protocol, we know that $\mathtt{V}$ and $\mathtt{P}$ are two $\mathcal{I}_0$-executable roles. Hence, the two processes ($\mathtt{V}(v_0,p_0)$ and $\mathtt{P}(p_0,v_0)$) which appear in the initial configuration that leads to Φ^* cannot involve agent names different from v_0 and p_0. Moreover, the attacker p cannot introduce new identities since the execution starts with an empty frame (agent names are not publicly available to forge a recipe except if provided in the initial frame). Finally we have that $\mathit{names}(\Phi^*) \cap \mathcal{A} \subseteq \{v_0, p_0\}$ and Theorem 4.1 applies in order to obtain that there exits a valid initial configuration $\mathcal{K}_0$ for $(\mathtt{V},\mathtt{P})$ w.r.t. $\mathcal{T}_{\mathsf{MF}}^{t_0}$ and $\Phi^* \cup \Phi_{\mathcal{I}_0}^{\mathcal{T}_{\mathsf{MF}}^{t_0}}$ such that:

$$\mathcal{K}_0 \xrightarrow{\mathsf{tr}'}_{\mathcal{T}_{\mathsf{MF}}^{t_0}} (\lfloor \mathsf{end}(v_0,p_0) \rfloor_{v_0}^{t'_v} \uplus \mathcal{P}'; \Phi'; t').$$

The other direction is an immediate application of Theorem 4.3 since the topology $\mathcal{T}_{\mathsf{MF}}^{t_0}$ belongs to $\mathcal{C}_{\mathsf{MF}}^{t_0}$. □

4.4. Conclusion

In comparison with standard security properties, mafia fraud, terrorist fraud or distance hijacking attack introduce new sources of unboundedness, and raise new challenges when considering automatic verification. Indeed, for each class of attacks an infinite number of topologies, here $\mathcal{C}_{\mathsf{MF}}^{t_0}$ or $\mathcal{C}_{\mathsf{DH}}^{t_0}$, have to be considered. Moreover, in case of terrorist fraud, the semi-dishonest prover, modelling the possible collusion behaviours, is also arbitrary and leads to another source of unboundedness. In this chapter we proved that, under few assumptions, we can get rid of these difficulties focusing our analysing on two rather simple topologies, $\mathcal{T}_{\mathsf{MF}}^{t_0}$ and $\mathcal{T}_{\mathsf{DH}}^{t_0}$, and a unique collusion behaviour, which we call the most general semi-dishonest prover $\mathtt{P}^*$. These reductions results constitute a first step for an automatic verification of distance-bounding protocols by removing sources of unboundedness. Unfortunately, even one simple topology cannot be immediately modelled in existing tools. In Chapter 5 we will explain how to encode the reduced topologies in the existing verification tool Proverif. Thereafter we will be able to perform a large case studies analysis considering both distance-bounding and payment protocols.

Implementations and case studies 5

In this chapter, we implement the theoretical results obtained in Chapter 3 and 4 to analyse many distance-bounding and contactless payment protocols. We first present several generic abstractions made on the protocols in order to analyse them through our frameworks. Then we explain how the procedure presented in Chapter 3 has been integrated upon the Akiss tool [CCCK16]. Lastly relying on the reduction results, we leverage the existing Proverif tool [Bla01] to allow the analyses of distance-bounding protocols. Both approaches are accompanied with an analysis of case studies and a discussion about their limitations. All the material is available at: `https://gitlab.inria.fr/adebant/verif-db/`

Contents

5.1. Few practical abstractions

The two approaches we developed require to make few abstractions on the protocols we want to analyse. First, we recall the main abstraction mentioned in Chapter 2 applying on the challenge/response exchange. Then, we discuss the classical problem in symbolic models which is the modelling of the exclusive-OR operator. To perform our analyses, we will have to model a weaker operator in order to match our theoretical development (Chapter 3) or the underlying model of existing tools (Proverif).

5.1.1. A unique challenge/response exchange

In Chapter 1 we presented the SPADE protocol (see Figure 1.3) as originally described in [BGG+16]. Similarly to almost all the distance-bounding protocols, the rapid phase, used to estimate the distance between the verifier and the prover, is made of n challenge/response exchanges. To minimise the computation time on the prover side and minimise the variance of the transmission delay, each challenge and each response are reduced to a bit, i.e. 0 or 1. This optimisation allows to compute a tighter bound of the distance in practice.

Unfortunately, symbolic models and procedures do not allow us to model single bits. Indeed since 0 and 1 are public values the attacker will be able to guess all the challenges and mount an attack. Since models and procedures do not handle probabilities the attack will always succeed, i.e. with probability 1. This is unrealistic in practice since an attacker might, of course, guess a bit, but only with probability $\frac{1}{2}$ which falls down the probability of success of the attack (he might only guess all the bits with probability $(\frac{1}{2})^n$).

To overcome this well-known limitation of symbolic models, the `new` commands allows to pick a fresh name unknown from the attacker. However, since this name cannot be guessed (even with probability $\frac{1}{2}$), it do not faithfully model a single bit neither. Hopefully, it can model a bit string that is long enough to be assumed unguessable. We therefore collapse the n challenge/response exchanges into a single one in which the challenge, i.e. the n-bits in a row, is modelled by a fresh name.

5.1.2. A weak exclusive-OR operator

Another abstraction applies to the exclusive-OR operator. When looking at symbolic verification, modelling such an operator is a difficult task due to its associativity and commutativity properties. Only very few existing tools handle it [BDGK17, DHRS18].

Unfortunately, the Proverif tool does not belong to this list [1] and thus, even if our theoretical result presented in Chapter 4 is strong enough to support such an operator, we have had to weaken it when performing the analysis of our case studies.

On the opposite, the Akiss tool has been recently updated to handle this operator [BDGK17], but we built on the original procedure (simpler and enough for a proof of concept) which does not. We thus weakened the operator to conduct our analyses too.

[1] Note that the Proverif tool has been extended to handle the exclusive-OR operator in [KT11], but the new procedure did not prove its efficiency in practice.

In both analyses, the algebraic properties of the weaker exclusive-OR operator we modelled are described through the following rules:

$$\begin{array}{rcl} \mathtt{cancelXor}(x \oplus y, x) & \to & y \\ \mathtt{cancelXor}(x \oplus y, y) & \to & x \\ \mathtt{cancelXor}(x, x \oplus y) & \to & y \\ \mathtt{cancelXor}(y, x \oplus y) & \to & x. \end{array}$$

The symbol `cancelXor` of arity 2 is a public symbol of function that an attacker can use to extract part of a message. In the Proverif tool, we also consider the rule

$$\mathtt{commutXor}(x \oplus y) \to y \oplus x$$

to strengthen our modelling. We do not in Akiss since it leads to non-termination issues.

Remark 5.1. *It is important to note that these rules do not perfectly model the exclusive-OR operator. Indeed, let us consider the rewriting system made of the previous rules and note* $\mathtt{E}_\oplus$ *the equational theory of the exclusive-OR operator (see Example 2.2). We define* $u = (x \oplus y) \oplus z$*. We have that:*

$$u \oplus y =_{\mathtt{E}_\oplus} x \oplus z \quad \textit{but} \quad (u \oplus y)\!\downarrow \neq (x \oplus z)\!\downarrow.$$

5.2. Integration in Akiss

In this section, we explain how the procedure presented in Chapter 3 has been integrated into the existing tool Akiss and present our case studies. First, even if the input syntax of the original procedure is close the one presented in Chapter 3, it must have been slightly extended to model the topologies and the configurations under study in a simple way. Second, since the last step of our new procedure consists in solving timing constraints, we had to fix the form of the constraints that may appear in our protocols to be able to implement a verification procedure.

5.2.1. A unique destructor symbol: eq

Our theoretical development allows to consider rather arbitrary sets of constructor and destructor symbols. However, the original underlying calculus of Akiss does not model constructor/destructor symbols. Instead it assumes that all the symbols are constructors and provides a specific command that allows equality tests. In order to be able to build over the existing tool, we had to match this setting. Thus we will consider the following restrictions:

- $\Sigma_d = \{\mathtt{eq}/2\}$; and
- all the $\mathtt{let}_{\mathtt{mess}}$ commands have the form: $\mathtt{let}_{\mathtt{mess}}\ x = \mathtt{eq}(v_1, v_2)$ with $v_1, v_2 \in \mathcal{T}(\Sigma^+, \mathcal{N} \cup \mathcal{A} \cup \mathbb{R}_+ \cup \mathcal{X})$; and
- the variables bound in $\mathtt{let}_{\mathtt{mess}}$ commands do not appear elsewhere in the process.

Even if these restrictions may appear restrictive they are not. As discussed in Remark 2.2, modelling a primitive, e.g. an encryption scheme, through a rewriting system or an equational theory provides a close semantics. Moreover, once all the symbols are constructor ones but eq, the last restriction about bound variables does not alter the expressiveness of the model: all the messages are constructor terms and can the bound variable can thus be substituted by the term in the remaining of the process.

Example 5.1. *To illustrate what this restriction implies, let us consider the following trace:*

$$\begin{aligned} T \;=\; & (a_1, \mathtt{in}(x_1)).\\ & (a_1, \mathtt{let}_{\mathtt{mess}}\ x_2^1 = \mathsf{proj}_1(\mathsf{adec}(x_1, \mathsf{sk}(a_1)))).\\ & (a_1, \mathtt{let}_{\mathtt{mess}}\ x_2^2 = \mathsf{proj}_2(\mathsf{adec}(x_1, \mathsf{sk}(a_1)))).\\ & (a_1, \mathtt{out}(x_2^1)).\\ & (a_1, \mathtt{out}(x_2^2)). \end{aligned}$$

This trace models the behaviour of an agent a_1 who waits for an encrypted message that contains a pair. He then outputs its two components.

Modelling adec, proj_1, *and* proj_2 *as destructor symbols ensures that x_1 will always be instantiated by a term* $\mathsf{aenc}(\langle u_1, u_2\rangle, \mathsf{pk}(a_1))$ *(with u_1 and u_2 two messages) during an execution. Considering them as constructor symbols allows more executions. For example, the first input might be filled by a constant $n \in \Sigma_0$. The* $\mathtt{let}_{\mathtt{mess}}$ *commands will no longer block the execution.*

Accepting this change, i.e. we now view proj_1 *and* proj_2 *as constructors, the trace can be rewritten as follows to match the constraint about the* $\mathtt{let}_{\mathtt{mess}}$ *commands:*

$$\begin{aligned} T' \;=\; & a_1, \mathtt{in}(x_1)).\\ & (a_1, \mathtt{out}(\mathsf{proj}_1(\mathsf{adec}(x_1, \mathsf{sk}(a_1))))).\\ & (a_1, \mathtt{out}(\mathsf{proj}_2(\mathsf{adec}(x_1, \mathsf{sk}(a_1)))))). \end{aligned}$$

5.2.2. An extended syntax

Verifying reachability properties in our timed model requires to define the topology under study and the global time of the initial configuration. In addition to the usual definitions which appear in the preamble of an Akiss file, we consider the two following lines:

```
topology =  [v0,(0,0,0),hon], [p0,(2,0,0),hon], [i1,(0,0,0),dis],
            [i2,(0,0,0),dis].

initTime =  0.
```

This models the reduced topology $\mathcal{T}_{\mathsf{MF}}^{t_0}$ (with $t_0 = 2$) made of four agents: `v0` and `p0` are honest (`hon`), while the two agents `i1` and `i2` are dishonest (`dis`). The triplets `(0,0,0)` and `(2,0,0)` defines the locations in the space of each agent. Thanks to this syntax, arbitrary topologies can be modelled: one may consider an arbitrary number of honest or dishonest agents, each located at arbitrary locations. Finally, the second line sets the initial global time to 0.

For sake of simplicity, the Akiss tool takes as input a process algebra that is richer than ours: the parallel composition command $P||Q$ can be used to describe processes. It avoids to let the user enumerate all the possible interleavings. For sake of simplicity we decided to add this command too and, when analysing a configuration, apply our procedure $\textsc{Reachability}(T, t_0, \mathcal{T}_0)$ to all the traces T that are obtained by interleaving of the parallel processes. This computation can easily be automatised.

Unfortunately, computing all the possible interleavings turns out to lead to a huge overhead of pre-computation. As a first approximation, when analysing the SPADE protocol (presented in Example 2.5) w.r.t. only one verifier session running in parallel with one prover session, we obtain up to $\binom{9+13}{9} = 497420$ interleavings. To limit this blow-up, we implemented two partial order reduction (POR) techniques that are well-known to be sound and complete when looking at reachability properties [MVB10, BDH17]. They are as follows:

- the commands $\mathtt{let}_{\mathtt{mess}}$, $\mathtt{let}_{\mathtt{time}}$, $\mathtt{if}_{\mathtt{time}}$, and $\mathtt{out}$ without timing annotations are executed as soon as possible; and
- consecutive inputs without timing annotation are executed in a raw.

Note that we do not perform any optimisation on outputs and inputs with timing annotation since it may break the completeness of the procedure. In practice, these optimisations significantly reduce the number of traces that need to be analysed and make the procedure practical.

Example 5.2. *Let* $T = (a, \mathtt{in}(x_1)).(a, \mathtt{in}(x_2)).(a, \mathtt{out}(u)) \,||\, (b, \mathtt{in}(x_3)).(b, \mathtt{out}(v))$. *If we naively compute all the interleavings we obtain* 10 *traces. Applying the first optimisation, i.e. execute the outputs that do not contain time annotation as soon as possible, we reduce this number to only* 3. *Indeed* $(a, \mathtt{in}(x_2)).(a, \mathtt{out}(u))$ *and* $(b, \mathtt{in}(x_3)).(b, \mathtt{out}(v))$ *can be considered as two indivisible blocks.*

Finally, the second optimisation allows to reduce again this number to 2. *Indeed the two first inputs executed by agent a do not contain annotation and thus can be executed in a row. The two traces are:*

$$T_1 = (a, \mathtt{in}(x_1)).(a, \mathtt{in}(x_2)).(a, \mathtt{out}(u)).(b, \mathtt{in}(x_3)).(b, \mathtt{out}(v)) \text{ and}$$
$$T_2 = (b, \mathtt{in}(x_3)).(b, \mathtt{out}(v)).(a, \mathtt{in}(x_1)).(a, \mathtt{in}(x_2)).(a, \mathtt{out}(u)).$$

5.2.3. About the timing constraints

The procedure presented in Chapter 3 proceeds in three steps: first, it generates the seed statements corresponding to the symbolic trace given in input. Then it performs the saturation step in order to obtain a finite set of solved statements. Finally, the last step consists in verifying the timing constraints induced by the reach statements which belong to the saturated knowledge base.

Our procedure has been proved sound and complete as soon as there exists an algorithm to decide whether a set of timing constraints is satisfiable during the last step. We

were thus free to fix the type of timing constraints that are accepted in $\texttt{if}_{\texttt{time}}$ commands in our implementation and which algorithm will be used to check the satisfiability.

In Section 3.4.1 (Chapter 3) we presented how the set of timing constraints is generated. Since all the constraints, except those generated by $\texttt{if}_{\texttt{time}}$ commands, are linear ones, a first approach would be to allow any linear constraints in $\texttt{if}_{\texttt{time}}$ commands and implement the Simplex algorithm [Dan63]. However, even if this algorithm is known to be efficient in practice, it may be exponential in worst case. We thus decided to be slightly more restrictive on the form of the constraints that are accepted, and we rely on a Difference Bound Matrix (DBM) [Dil89] to represent the set of constraints. The satisfiability is then decidable in polynomial time.

We only accept timing constraints of the form $z_j - z_i \sim_{ij} c_{ij}$ with $z_i, z_j \in \mathcal{Z}$, $c_{ij} \in \mathbb{R} \cup \{+\infty\}$ and $\sim_{ij} \in \{<, \leq\}$. Given a finite set of variables $\{z_1, \ldots, z_n\} \subseteq \mathcal{Z}$, a DBM $M = ((\sim_{ij}, c_{ij}))_{1 \leq i,j \leq n}$ is the adjacency matrix of the weighted graph $G = (V, E, \omega)$ with:

- $V = \{z_1, \ldots, z_n\}$, the set of vertices;
- $E = \{(z_i, z_j)\}_{1 \leq i,j \leq n}$, the set of edges;
- $\omega((z_i, z_j)) = (\sim_{ij}, c_{ij})$, with $\omega : \mathcal{Z} \times \mathcal{Z} \to \{<, \leq\} \times \mathbb{R}_+ \cup \{+\infty\}$ the weight function.

It cleverly represents the set of constraints $S = \{z_j - z_i \sim_{ij} c_{ij} \mid 1 \leq i, j \leq n\}$. Then, deciding whether S is satisfiable consists in deciding whether there is a cycle of negative length in G. We have implemented the Floyd-Warshall algorithm which terminates in $\mathcal{O}(n^3)$ to detect such cycles. To apply the Floyd-Warshall algorithm, we consider the lexicographical order on pairs assuming that $<$ is smaller than $\leq$, and the following addition for weights:

$$(\sim_{i_1 j_1}, c_{i_1 j_1}) + (\sim_{i_2 j_2}, c_{i_2 j_2}) = (min(\sim_{i_1 j_1}, \sim_{i_2 j_2}), c_{i_1 j_1} + c_{i_2 j_2}).$$

Example 5.3. *Let* $M = \begin{pmatrix} (\leq,4) & (<,+\infty) & (<,+\infty) & (<,3) \\ (<,+\infty) & (<,+\infty) & (\leq,5) & (<,+\infty) \\ (<,+\infty) & (<,+\infty) & (<,+\infty) & (\leq,1) \\ (<,+\infty) & (<,-2) & (<,+\infty) & (<,+\infty) \end{pmatrix}$.

This is the adjacency matrix of the graph G depicted in Figure 5.1 and it represents the set of the following constraints:

$$\begin{array}{lll} z_2 - z_1 \leq 4 & z_4 - z_1 < 3 & z_3 - z_2 \leq 5 \\ z_4 - z_3 \leq 1 & z_2 - z_4 < -2. & \end{array}$$

Applying the Floyd-Warshall algorithm on G, we can conclude that there is no cycle of negative weight. Hence, the constraints are satisfiable. Indeed, a possible solution is:

$$z_1 := 2, \quad z_2 := 0, \quad z_3 := 3, \quad z_4 := 4.$$

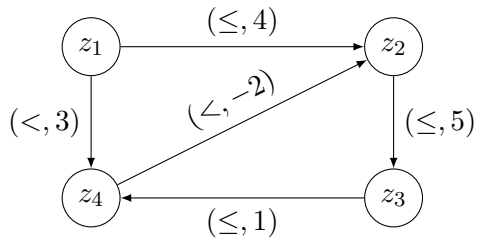


Figure 5.1: Graph representation of the DBM of Example 5.3.

5.2.4. Case studies and limitations

In this section, we demonstrate the usability in practice of our procedure. To do so, we analysed several distance-bounding and contactless payments protocols w.r.t. mafia frauds, terrorist frauds, and distance hijacking attacks.

Topology and initial configuration

Our procedure applies on a given topology and configuration. For the topology, we decided to consider the reduced ones presented in Chapter 4. They have been proved sufficient to capture all the possible attacks under some reasonable assumptions that are satisfied by our protocols (except MasterCard-RRP for which our analysis is therefore weaker). These two topologies, denoted $\mathcal{T}_{\mathsf{MF}}^{t_0}$ and $\mathcal{T}_{\mathsf{DH}}^{t_0}$, are recalled in Figure 5.2.

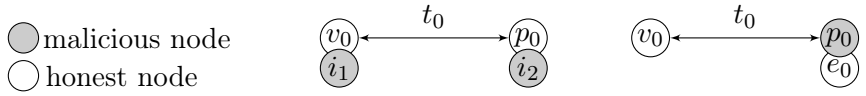


Figure 5.2: Reduced topologies $\mathcal{T}_{\mathsf{MF}}^{t_0}$ and $\mathcal{T}_{\mathsf{DH}}^{t_0}$

The initial configuration defines three elements: the number of sessions of each role that will be executed, the initial knowledge of the attackers, and the initial global time. To match the definition of valid initial configuration presented in Chapter 2, this time is set to 0. Again, to match the definition, the initial frame must contain the initial knowledge of each malicious agent derived from a template $\mathcal{I}_0$. Since the soundness and the completeness of our procedure assuming an empty initial frame, as explained in Example 3.2, we follow the usual construction in symbolic verification which consists in adding an extra process, solely made of outputs, that sends all the messages that belong to these knowledges. Finally we are going to consider as many sessions of each role as possible.

Case studies

We conducted our analyses on a standard laptop. As mentioned above, we consider the reduced topologies made of, at most, four agents. Then we proceeded first by considering one session for each of the honest agents, and then adding as many sessions as

possible. Note that when an attack is found, there is no need to consider more sessions. We have been able to consider up to five sessions before being limited by the exponential blow-up of the interleavings that causes non-termination in practice. When considering terrorist fraud, we considered the most-general semi-dishonest prover defined in Chapter 4. Note that a protocol is terrorist fraud resistant whenever a re-authentication is possible; in that case the number of traces is no longer relevant. On the contrary, to prove that a protocol suffers from a terrorist fraud, one may show that no re-authentication is possible: we thus need to consider as many sessions as possible.

The results we obtained are presented in Table 5.1. Almost all the protocols, except the Hancke and Kuhn [HK05] and Swiss-Knife [KAK+08] are subject to a mafia fraud and/or a distance hijacking attack and/or a terrorist fraud. For example, we retrieve the mafia fraud and the distance hijacking attack against the SPADE protocol we used to illustrate our security properties in Chapter 2. As claimed in Example 2.12, we also prove the fixed version of the SPADE protocol is mafia fraud resistant as expected. All the results are coherent with the literature [CGdR+15, CdRS18, MSTPTR18] and those, that will be presented in the next section, considering an unbounded number of sessions.

When looking at payment protocols (i.e. MasterCard RRP [EMV16] and PaySafe [CGdR+15]) one may note that they all resist to mafia frauds but none of them resist to distance hijacking attacks. Hopefully, in this context, mafia frauds are the most common scenario of attacks, i.e. an attacker tries to pay something abusing a remote and honest person. This is the attack that the EMV Co. specification explicitly mentions. However, even if the distance hijacking scenarios do not appear in the specification, we do think that payment protocol should resist against them. Indeed, a receipt for contactless payment might be considered as a proof of location since the underlying protocols pretend to ensure physical proximity. As an immediate consequence, an attacker may have incentives to abuse honest agents in the vicinity of a reader/verifier to pay on his behalf and obtain such an evidence. Regarding terrorist frauds, our procedure encountered non-termination issues due to the large size of the terms involved in such protocols.

Limitations

Our case studies illustrate the efficiency in practice of our procedure to analyse protocols with small terms. Our framework is expressive enough to analyse many distance-bounding protocols relying on the reductions results proposed in Chapter 4. Only the Brands and Chaum protocol [BC93] cannot be analysed w.r.t. terrorist frauds since the main theorem reducing the number of semi-dishonest provers dos not apply. However, when the protocols tend to involve more complex terms (e.g. made of complex signatures and MAC messages as in the payment protocols), the procedure encounters efficiency issues.

First, we can notice that the size of the terms significantly impacts the duration of the saturation step. This may result from the update function which discards fewer recipes than in the original procedure. Sadly, this limitation seems complex to overcome: keeping more recipes tends to be inherent to our timed model, according to Example 3.7, in order to get a procedure that is complete.

Then, the size of the terms also impacts the duration of the last step of the procedure: the larger a message is, the more recipes are available to an attacker to deduce it. This may increase a lot the size of the sets $\overline{L_i}$. To overcome this limitation we could cleverly tune our backtracking search relying on the partial order defined on frame variables to look for asap recipes only. It would just require algorithmic and implementation stuff since our theoretical development allows this.

Finally, the main limitation of the procedure lies in the quite small number of sessions that can be analysed. The limitation does not come from the core procedure, presented in Chapter 3, itself but rather from the number of interleavings blow-up. A first approach to overcome this limitation would have been to parallelise the verification. Indeed each trace can be analysed independently. Unfortunately this will not completely bridge the gap since the computation of all the interleavings quickly becomes the main bottleneck. An interesting solution would be to develop clever POR optimisations and prove their correctness.

5.3. Verification using Proverif

While the definitions presented in Chapter 2 require to consider an infinite number of topologies when analysing distance-bounding protocols, Chapter 4 establishes that it is sufficient to focus on two rather simple topologies, $\mathcal{T}^{t_0}_{\mathsf{MF}}$ and $\mathcal{T}^{t_0}_{\mathsf{DH}}$, recalled in Figure 5.2). In this section, we will demonstrate how they can be encoded into the well-known automatic verification tool Proverif [Bla01], in order to analyse a large number of case studies. Indeed, even if the Proverif tool does not allow to faithfully model time, it implements a notion of phases that will be sufficient to encode the reduced topologies and our timing constraints.

5.3.1. Encoding of the topologies

We decided to rely on the Proverif tool to perform our case studies. It shares a lot of similarities with our symbolic model presented in Chapter 2 and restrained in Chapter 4. Proverif handles both equational and rewriting systems, and describes protocols through a process algebra derived from the applied pi-calculus. The methodology we apply to encode the reduced topologies builds on the following subset of the Proverif calculus:

$$P \;:=\; 0 \mid \mathtt{in}(x).P \mid \mathtt{out}(u).P \mid \mathtt{let}_{\mathtt{mess}}\; x = v \;\mathtt{in}\; P \mid \mathtt{new}\; n.P \mid i : P \mid !P.$$

Almost all the commands are similar to those defined in our timed semantics. The main novelty is the phase command, denoted $i : P$. Informally phases model synchronisation points in the execution of a protocol. An execution always starts at phase 0. It can arbitrarily increase during the execution but, during phase i, only processes that are actually at this stage can be executed. Another difference is the presence of the replication $!P$ command which was deliberately omitted in our timed model since an arbitrary number of sessions may be added in a valid initial configuration. This syntactic sugar will be useful to define the configuration under study in Proverif.

This new semantics is formally described by a relation $\Rightarrow$ over configurations. A configuration is a tuple $(\mathcal{P}; \phi; i)$ where $\mathcal{P}$ is a multiset of processes, ϕ is a standard frame

Protocol	Mafia fraud			
	# sessions	# traces	time	status
SPADE [BGG+16]	2	–	2s	×
TREAD-Asym [ABG+17]	2	–	1s	×
TREAD-Sym [ABG+17]	4	7500	18min	✓
BC [BC93]	4	5635	37min	✓
Swiss-Knife [KAK+08]	3	1080	25s	✓
HK [HK05]	3	20	1s	✓
	4	3360	58s	✓
	5	30240	14min	✓
ISO/IEC 14443 protocols				
MasterCard RRP [EMV16]	2	35	6min	✓
NXP [Jan17]	2	126	4s	✓
PaySafe [CGdR+15]	2	4	5min	✓

Protocol	Distance hijacking			
	# sessions	# traces	time	status
SPADE [BGG+16]	2	–	4s	×
TREAD-Asym [ABG+17]	2	–	1s	×
TREAD-Sym [ABG+17]	2	–	1s	×
BC [BC93]	2	–	1s	×
Swiss-Knife [KAK+08]	3	7470	4min	✓
HK [HK05]	3	20	1s	✓
	4	3360	47s	✓
	5	30240	12min	✓
ISO/IEC 14443 protocols				
MasterCard RRP [EMV16]	2	–	8s	×
NXP [Jan17]	2	–	1s	×
PaySafe [CGdR+15]	2	–	3s	×

Protocol	Terrorist fraud			
	# sessions	# traces	time	status
SPADE [BGG+16]	2	–	3s	✓
TREAD-Asym [ABG+17]	2	–	1s	✓
TREAD-Sym [ABG+17]	4	–	3s	✓
BC [BC93]	*o.o.s.*	*o.o.s.*	*o.o.s.*	*o.o.s.*
Swiss-Knife [KAK+08]	2	-	1s	✓
HK [HK05]	2	20	1s	×
	4	3360	76s	×
	5	30240	27min	×
ISO/IEC 14443 protocols				
MasterCard RRP [EMV16]	2	35	*o.o.t.*	×
NXP [Jan17]	2	126	13s	×
PaySafe [CGdR+15]	2	4	*o.o.t.*	×

Table 5.1: Results of the analyses considering a bounded number of sessions (✓: proved secure, ×: attack, *o.o.s.*: out of scope, *o.o.t.*: out of time).

(without timing annotations) and $i \in \mathbb{N}$ is the current phase of the system. All the rules are given in Figure 5.3. One may note that, except the restriction about the current phase, it matches the untimed semantics presented in Section 4.2.2 (Chapter 4).

In	$(i : \mathtt{in}(x).P \uplus \mathcal{P}; \phi; i) \xrightarrow{\mathtt{in}(u)} (i : P\{x \mapsto u\} \uplus \mathcal{P}; \phi; i)$	with u a constructor term such that $u = R\phi\downarrow$ for some recipe R
Out	$(i : \mathtt{out}(u).P \uplus \mathcal{P}; \phi; i) \xrightarrow{\mathtt{out}(u)} (i : P \uplus \mathcal{P}; \phi \uplus \{\mathsf{w} \mapsto u\}; i)$	with $\mathsf{w} \in \mathcal{W}$ fresh
Let	$(i : \mathtt{let}_{\mathtt{mess}}\ x = v\ \mathtt{in}\ P \uplus \mathcal{P}; \phi; i) \overset{\tau}{\Rightarrow} (i : P\{x \mapsto v\downarrow\} \uplus \mathcal{P}; \phi; i)$	when $v\downarrow \in \mathcal{T}(\Sigma_c, \mathcal{N} \cup \mathcal{A})$
New	$(i : \mathtt{new}\ n.P \uplus \mathcal{P}; \phi; i) \overset{\tau}{\Rightarrow} (i : P\{n \mapsto n'\} \uplus \mathcal{P}; \phi; i)$	with $n' \in \mathcal{N}$ fresh.
Rep	$(i : !P \uplus \mathcal{P}; \phi; i) \overset{\tau}{\Rightarrow} (i : P \uplus (i : !P) \uplus \mathcal{P}; \phi; i)$	
Move	$(\mathcal{P}; \phi; i) \xrightarrow{\mathsf{phase}\ i'} (\mathcal{P}; \phi; i')$	with $i' > i$.
Phase	$(i : i' : P \uplus \mathcal{P}; \phi; i) \overset{\tau}{\Rightarrow} (i' : P \uplus \mathcal{P}; \phi; i)$	

Figure 5.3: Semantics for the Proverif calculus

Our encoding

Given a DB protocol $(\mathtt{V}, \mathtt{P})$, we propose a transformation that encodes the reduced topologies $\mathcal{T}^{t_0}_{\mathsf{MF}}$ and $\mathcal{T}^{t_0}_{\mathsf{DH}}$ into the Proverif calculus. However it requires that the role $\mathsf{V}_{\mathsf{end}}(\mathsf{x}_0, \mathsf{x}_1)$ is made of a unique challenge/response exchange, i.e. is of the form:

$$\mathsf{block}_1 \,.\, \mathtt{reset} \,.\, \mathtt{out}(u) \,.\, \mathtt{in}^{<t}(x) \,.\, \mathsf{block}_2 \,.\, \mathsf{end}(\mathsf{x}_0, \mathsf{x}_1)$$

where $\mathsf{block_i}$ is a sequence of actions (only simple inputs, outputs, let, and new instructions are allowed).

The main idea for the transformation is to use the notion of phases provided by Proverif to encode the critical phase, i.e. the challenge/response exchange. To do so, we consider three phases: phase 0 before the critical phase, phase 1 that starts when executing the `reset` action and stops just after the input of the response, i.e. the $\mathtt{in}^{<t}(x)$ action, and finally phase 2 to the remainder of the protocol. The two locations of the reduced topologies are then modelled as follows: agents close the verifier are allowed to execute actions during phase 1 while agents far away are not. Given a parametrised role (with no `reset` and $\mathtt{in}^{<t}(x)$ commands), the two corresponding transformations, denoted $\mathcal{F}^{<}$ and $\mathcal{F}^{\geq}$, are thus defined as follows:

- transformation $\mathcal{F}^{<}$: this transformation introduces the phase instructions with $i =$ 0, 1 and 2 considering all the possible ways of splitting the role into three parts (0,

1, and 2). Each phase instruction is placed before an `in` instruction. Such a slicing is actually sufficient for our purposes.

- transformation $\mathcal{F}^{\geq}$: this transformation does the same but we forbid the use of the instruction phase 1, jumping directly from phase 0 to phase 2.

The configuration, denoted $\mathcal{F}(\mathcal{T}, (\mathtt{V}, \mathtt{P}), \Phi_0, t_0)$, is the tuple $(\mathcal{P}; \phi; 0)$ where ϕ is such that $img(\phi) = img(\Phi_0)$, and $\mathcal{P}$ is the multiset that contains the following processes:

- $\mathtt{V}'_{\mathsf{end}}(v_0, p_0) = \mathsf{block}_1.\ 1 : \mathtt{out}(u).\mathtt{in}(x)\ .\ 2 : \mathsf{block}_2.\mathsf{end}(v_0, p_0)$;
- $!R(a, b)$ when $R(\mathsf{x}_0, \mathsf{x}_1) \in \mathcal{F}^{<}(\overline{\mathtt{V}}) \cup \mathcal{F}^{<}(\overline{\mathtt{P}})$, $a, b \in \mathcal{A}_0$, $\mathsf{Dist}_{\mathcal{T}}(v_0, a) < t_0$;
- $!R(a, b)$ when $R(\mathsf{x}_0, \mathsf{x}_1) \in \mathcal{F}^{\geq}(\overline{\mathtt{V}}) \cup \mathcal{F}^{\geq}(\overline{\mathtt{P}})$, $a, b \in \mathcal{A}_0$, $\mathsf{Dist}_{\mathcal{T}}(v_0, a) \geq t_0$;

where $\overline{\mathtt{P}}$ (resp. $\overline{\mathtt{P}}$) is the parametrised process `V` (resp. `P`) in which all the `reset` actions have been removed and all the guarded inputs replaced by simple ones.

We are now able to establish the following result that formally justifies the soundness of our transformation, i.e. if there is an attack then Proverif will find it.

Proposition 5.1. *Let $t_0 \in \mathbb{R}_+$, $\mathcal{I}_0$ be a template, $\mathcal{T} = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}_0, v_0, p_0)$ be a topology and $\mathcal{P}_{\mathsf{db}} = (\mathtt{V}(\mathsf{x}_0, \mathsf{x}_1), \mathtt{P}(\mathsf{x}_0, x_1))$ a distance-bounding protocol such that $V_{\mathsf{end}}(z_0, z_1)$ has the form:*

$$\mathsf{block}_1\ .\ \mathtt{reset}\ .\ \mathtt{out}(u)\ .\ \mathtt{in}^{<t}(x)\ .\ \mathsf{block}_2\ .\ \mathsf{end}(z_0, z_1)$$

with $t \leq 2 \times t_0$.
Let $\mathcal{K}_0$ be a valid initial configuration for the $\mathcal{P}_{\mathsf{prox}}$ w.r.t. $\mathcal{T}$ and an initial frame Φ_0. If $\mathcal{K}_0$ admits an attack w.r.t. t_0-proximity in $\mathcal{T}$, i.e.

$$\mathcal{K}_0 \xrightarrow{\mathsf{tr}}_{\mathcal{T}} (\lfloor \mathsf{end}(v_0, p_0) \rfloor_{v_0}^{t_v} \uplus \mathcal{P}; \Phi; t),$$

then we have that:

$$\mathcal{F}(\mathcal{T}, \overline{\mathcal{P}_{\mathsf{prox}}}, \Phi_0, t_0) \stackrel{\mathsf{tr}'}{\Longrightarrow} (\{2 : \mathsf{end}(v_0, p_0)\} \uplus \mathcal{P}; \phi; 2).$$

Moreover, in case there is no $a \in \mathcal{M}_0$ such that $\mathsf{Dist}_{\mathcal{T}}(a, v_0) < t_0$, we have that for any $\mathtt{in}(u)$ occurring in tr during phase 1, the underlying recipe R is either of the form $\mathsf{w} \in \mathcal{W}$, or only uses handles outputted in phase 0.

This proof relies on technical lemmas presented in Chapter 4 and proceeds in three steps:

1. it cleans the trace in a similar way as what has been done to establish the reduction results in case of distance hijacking attacks (Proposition 4.2). Only actions that depends on the `reset` and on which the guarded input depends on remain between these two actions.
2. relying on the dependency, it deduces timing constraints that prevent remote agents to execute such actions during the rapid phase (Lemma 4.5).
3. it concludes by proving that $\mathcal{F}(\mathcal{T}, \overline{\mathcal{P}_{\mathsf{db}}}, \Phi_0, t_0)$ subsumes all the possible valid initial configurations (e.g. relying on the replication command).

5.3.2. Proof of the soundness of the encoding

In this section, we present the formal proof of Proposition 5.1.

Proof. In the same spirit as the proof of Theorem 4.2, we are able to prove that:

$$\widetilde{\mathcal{K}}_0^\star \overset{\mathsf{tr}'_1.....\mathsf{tr}'_n}{\rightsquigarrow_{\mathcal{T}}} (\lfloor \mathsf{end}(v_0,p_0)\rfloor_{v_0} \uplus \widetilde{\mathcal{P}}; \widetilde{\Phi})$$

where $\widetilde{\mathcal{K}}_0^\star$ (resp. $\widetilde{\mathcal{P}}$, $\widetilde{\Phi}$) is the untimed counterpart of $\mathcal{K}_0$ (resp. $\mathcal{P}$, Φ) in which `reset` commands have been removed and guarded inputs have been replaced by simple inputs except in $\mathtt{V}_{\mathsf{end}}(v_0,p_0)$.

Moreover if we note i_0 (resp. j_0) the index of the `reset` (resp. guarded input) occurring in $\mathtt{V}_{\mathsf{end}}(v_0,p_0)$ we have that:

(i) for all $i \in \{i_0,\ldots,j_0\}$, the action tr'_i is executed by an agent $a_i \in \mathsf{Close}(v_0)$, i.e. $\mathsf{Dist}_{\mathcal{T}}(a_i,v_0) < t_0$;

(ii) for all $i \in \{i_0,\ldots,j_0\}$, if tr'_i is an input then the agent b_i responsible of the output is such that $b_i \in \mathsf{Close}(v_0)$, i.e. $\mathsf{Dist}_{\mathcal{T}}(b_i,v_0) < t_0$, or the recipe that is used to trigger the input only contains handles binding outputs executed before the `reset` command.

In the following we say that a process is initial if it starts by an input. Let s_0 be the session identifier of the process $\mathtt{V}_{\mathsf{end}}(v_0,p_0)$. In the following we will prove that there exists a trace preserving items (i) and (ii) and satisfying the following two properties:

(iii) all processes but s_0 are either initial when executing the `reset` action or let unchanged until the end of the execution, i.e. there is no action in trace corresponding to this process after the `reset` action.

(iv) all processes but s_0 are either initial when executing the $\mathtt{in}^{<t}(u)$ action or let unchanged until the end of the execution.

Item (iii): Assume that there exists a process in $\widetilde{\mathcal{K}}^\star_{\mathtt{reset}}$, the configuration just before the `reset` command, with a session identifier $s_k \neq s_0$ that is not initial. Let $k \in \{i_0,\ldots,n\}$ be the index of the first action corresponding to this process, i.e. tagged by the session identifier s_k in the remaining of the trace. If k does not exist then the process is let unchanged and item (iii) is satisfied. Otherwise, we prove the following claim.

Claim. *For all $i \in \{i_0,\ldots,k-1\}$ we have that $\mathsf{tr}'_k \not\hookrightarrow^* \mathsf{tr}'_i$.*

Proof. We note $\mathsf{tr}'_k = (a_k,\alpha_k,s_k,r_k)$ and $\mathsf{tr}'_i = (a_i,\alpha_i,s_i,t_i)$. Assume that $\mathsf{tr}'_k \hookrightarrow \mathsf{tr}'_i$. If $\mathsf{tr}'_k \hookrightarrow_s \mathsf{tr}'_i$ then we have that $s_i = s_k$. Contradiction. Otherwise we have $\mathsf{tr}'_k \hookrightarrow_d \mathsf{tr}'_i$ and thus $\alpha_k = \mathtt{in}^\star(v)$. Contradiction since the process identified by s_k in $\widetilde{\mathcal{K}}^\star_{\mathtt{reset}}$ is assumed to be not initial. □

Repeatedly applying Lemma 4.6 we are able to move the action tr'_k just before $\widetilde{\mathcal{K}}^{\star}_{\mathtt{reset}}$. Applying the same reasoning for all actions corresponding to s_k until we reach an initial process (and then to all processes that are not initial in $\widetilde{\mathcal{K}}^{\star}_{\mathtt{reset}}$) we obtain an execution:

$$\widetilde{\mathcal{K}}^{\star}_0 \overset{\mathsf{tr}'_1.....\mathsf{tr}'_{i_0-1}.\widetilde{\mathsf{tr}}_{i_0}.....\widetilde{\mathsf{tr}}_{i'_0-1}}{\rightsquigarrow_{\mathcal{T}}} \mathcal{K}'_{\mathtt{reset}} \overset{\widetilde{\mathsf{tr}}_{i'_0}.....\widetilde{\mathsf{tr}}_{j'_0-1}}{\rightsquigarrow_{\mathcal{T}}} \mathcal{K}'_{\mathtt{in}} \overset{\widetilde{\mathsf{tr}}_{j'_0}.....\widetilde{\mathsf{tr}}_{n}}{\rightsquigarrow_{\mathcal{T}}} (\lfloor \mathsf{end}(v_0, p_0) \rfloor_{v_0} \uplus \widetilde{\mathcal{P}}; \widetilde{\Phi})$$

that matches item (iii) by construction. Moreover, items (i) and (ii) hold since we do not introduce new actions between the `reset` and the guarded input (actually, we only move actions before the `reset`).

Item (iv): We follow the same reasoning as for item (iii). Assume there is a process that is not initial in $\mathcal{K}'_{\mathtt{in}}$ and note s_k its session identifier. Note k the index of the first action corresponding to session s_k after $\mathcal{K}'_{\mathtt{in}}$. If k does not exist then the process is not initial in $\mathcal{K}'_{\mathtt{reset}}$ and thus is let unchanged until the end on the execution according to item (iii) that holds. For the same reasons as previously, we can establish that the following claim holds.

Claim. *For all $i \in \{j'_0, \ldots, k-1\}$ we have that $\widetilde{\mathsf{tr}}_k \not\leftrightarrow^* \widetilde{\mathsf{tr}}_i$.*

Again, applying Lemma 4.6 we are able to move action $\widetilde{\mathsf{tr}_k}$ right before $\mathcal{K}'_{\mathtt{in}}$. We obtain a trace that satisfies item (iv). Moreover, the three items (i), (ii) and (iii) are still satisfied:

(i) all the actions we introduce between the `reset` and the guarded input are executed by agents $a \in \mathsf{Close}(v_0)$. Indeed, we know that item (ii) holds and thus the process identified by s_k is initial when executing the `reset` action. Hence, we can deduce that agent a must have executed an action in the meantime since it is no longer initial when executing the guarded input. Relying on item (i) (that holds on the previous trace) we deduce that $a \in \mathsf{Close}(v_0)$;

(ii) we do not introduce inputs between the `reset` and the guarded input;

(iii) the beginning of the trace (before the `reset`) is not modified ($k > j'_0$).

To conclude, it remains to show that such a trace can actually be mimicked from the configuration $\mathcal{F}(\mathcal{T}, (\mathtt{V}, \mathtt{P}), \Phi_0, t_0)$. Actually, the `reset` action is replaced by $\mathsf{phase}\ 1$, and the guarded input is replaced by a simple input followed by a $\mathsf{phase}\ 2$ action.

Let $\lfloor P \rfloor^t_a$ be a process occurring in $\widetilde{\mathcal{K}}^{\star}_0$, we know that $P = \mathtt{V}(a, b)$ or $P = \mathtt{P}(a, b)$ for some $b \in \mathcal{A}_0$. We now consider all the actions performed by this process along the execution, and in particular, we pay attention to the slicing of all these actions w.r.t. `reset` action and the guarded input. This gives us the corresponding process that we have to consider in our translation, so that it will be able to mimic all the actions of $\lfloor P \rfloor^t_a$. Items (iii) and (iv) allow one to ensure that our slicing (just before the inputs) is indeed sufficient. Our transformation $\mathcal{F}^{\geq}$ also forbids actions to be executed during phase 1, and this is justified by our item (i). Finally, item (ii) allows us to prevent the

Proverif attacker from acting in phase 1 in case there is no dishonest participant in the vicinity of v_0.

□

5.3.3. Case study and limitations

We can now present our case studies. Thanks to the reduction results presented in Chapter 4 and Proposition 5.1 we can analyse a DB protocol $\mathcal{P}_{\mathsf{db}}$ w.r.t. each class of attacks separately by analysing a unique configuration in Proverif:

- Mafia fraud: the configuration $\mathcal{F}(\mathcal{T}^{t_0}_{\mathsf{MF}}, \mathcal{P}_{\mathsf{db}}, \Phi^{\mathcal{T}^{t_0}_{\mathsf{MF}}}_{\mathcal{I}_0}, 0)$;
- Distance hijacking: the configuration $\mathcal{F}(\mathcal{T}^{t_0}_{\mathsf{DH}}, \mathcal{P}_{\mathsf{db}}, \Phi^{\mathcal{T}^{t_0}_{\mathsf{DH}}}_{\mathcal{I}_0}, 0)$;
- Terrorist fraud: the configuration $\mathcal{F}(\mathcal{T}^{t_0}_{\mathsf{MF}}, \mathcal{P}_{\mathsf{db}}, \Phi^{\mathcal{T}^{t_0}_{\mathsf{MF}}}_{\mathcal{I}_0} \cup \Phi^*, 0)$ where Φ^* is the frame associated to the most general semi-dishonest prover.

If the protocol is proved secure, then $\mathcal{P}_{\mathsf{db}}$ is resistant to the class of attacks we have considered. Otherwise, the attack trace that is returned by Proverif must be analysed to decide whether it is executable in our timed semantics, and thus corresponds to a real attack. Indeed, our reduction result slightly modifies the initial configuration by removing `reset` commands and guarded inputs in processes other than $\mathsf{V}_{\mathsf{end}}(v_0, p_0)$. Moreover, Proverif might find false attacks due to some internal optimisations; in that case, Proverif typically returns *cannot be proved*.

Remark 5.2. *We noticed that Proverif was always returning false attacks when applying our methodology to distance hijacking scenarios, i.e. scenarios in which there is no dishonest agents in the neighbourhood of the verifier. Indeed, even if we do not consider dishonest agents executing roles of the protocol during phase* 1*, the underlying attacker of Proverif is still able to interact with participants. To prevent such behaviours we slightly modified the Proverif code: during phase* 1 *the attacker can only forward messages sent by honest agents, or forge a new message, but only using the knowledge he got in phase* 0*. More precisely, in Proverif the capabilities of the attacker are described by Horn clauses made of a predicates* $\mathtt{attacker\_pi}(u)$ *with* $i \in \mathbb{N}$*, which means that the term* u *is known by the attacker in phase* i*. For example, to model that the attacker is able to build in phase* 0 *an encrypted message* $\mathsf{aenc}(x, y)$ *whenever he knows the terms* x *and* y*, Proverif considers the following clause:*

$$\mathtt{attacker\_p0}(\mathsf{aenc}(x, y)) \Leftarrow \mathtt{attacker\_p0}(x) \wedge \mathtt{attacker\_p0}(y).$$

The implementation of our restriction consists in removing almost all the clauses that involve the predicate $\mathtt{attacker\_p1}(\cdot)$*. We only keep the two following ones*[2]*:*

$$\textit{(1)}\quad \mathtt{attacker\_p1}(x) \Leftarrow \mathtt{attacker\_p0}(x)$$
$$\textit{(2)}\quad \mathtt{attacker\_p2}(x) \Leftarrow \mathtt{attacker\_p1}(x).$$

[2]We did not remove the clause that let the attacker listen on all channel it has in phase 1 but this clause is useless since we use a unique public channel.

These two clauses transfer the knowledge of the attacker from one phase to the next. Based on the extra property of Proposition 5.1, this modification should not miss any attack: either the inputted message u has been sent in phase 1 and a clause of the protocol will generate the predicate `attacker_p1`(u) *or it is built from messages previously sent during phase 0 and the procedure will generate the term in phase 0, i.e.* `attacker_p0`(u)*, and transfer it to phase 1 thanks to the Horn clause (1). Finally, the Horn clause (2) is necessary to let the attacker re-use messages sent during phase 1 in the remaining of the execution, i.e. in phase 2.*

Results

We analysed more than 25 protocols and the results are presented in Table 5.2. Proverif always returns in few seconds on a standard laptop, except for protocols marked by an asterisk which require up to 80min. The results are coherent with the other case studies in the literature [CdRS18, MSTPTR18, MSTPTR19].

5

Amongst these results, some of them are new. For example we found an attack against the SPADE [BGG+16] and propose the first proof of security for the fixed version of the SPADE protocols [Ger18] considering an unbounded number of sessions. This fix, mentioned in Example 2.12 consists in adding the identity of the prover in the first signature of the protocol. Another interesting result is the first analysis of the Meadows *et al.*'s protocol instantiated with the answering function $F(n_V, P, n_P) = \langle n_V \oplus n_P, P \rangle$. This variant of the protocol, mentioned in the original paper, could not be analysed relying on their authentication logic. From this aspect, our framework allows to automatically analyse a wider class of protocols.

Limitations

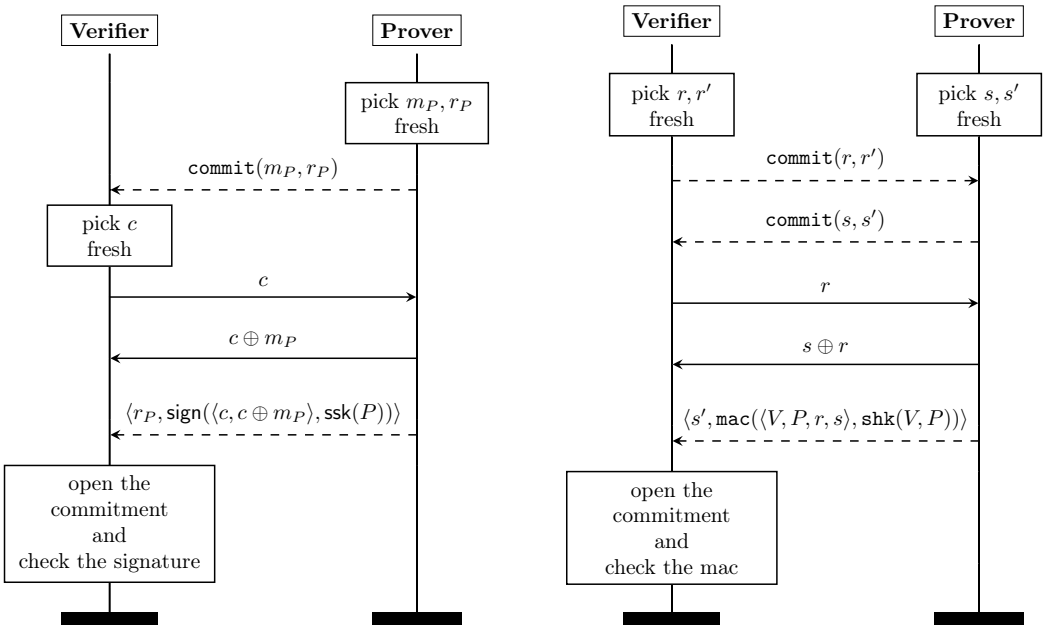
The case studies prove the efficiency of this approach in practice. However, few limitations exist. Indeed, while all the protocols satisfy the assumptions to be analysed w.r.t. mafia fraud or distance hijacking, the requirements to analyse them w.r.t. terrorist fraud are more restrictive. Two protocols, the Brands and Chaum's and MAD (presented in Figure 5.4), stays out of the scope of our reduction result that allows to define the most-general semi-dishonest prover. The former does not satisfy item (iv) of Definition 4.3 since the response to the challenge c is $c \oplus m$ where $\oplus$ is not quasi-free since it appears in the equational theory. The last does not satisfy item (i) due to a lack of freshness of the challenge. Since these two protocols suffer from a terrorist fraud, we could retrieve this vulnerability using our framework. Indeed, we can manually define the semi-dishonest prover we know that leads to the attack, and then use Proverif to check that there is no trace of re-authentication. Nevertheless, we note them *"out of scope"* in Table 5.2 to underline this limitation of our approach.

Another limitation comes from the model itself. In order to establish the reduction results we had to restrict our initial model by limiting the use of the $\texttt{if}_{\texttt{time}}$ and $\texttt{let}_{\texttt{time}}$ commands. This has no impact for almost all the protocols we analysed except for the NXP and MasterCard-RRP protocols. In these two protocols, the proximity check is

Protocols	**MF**	**DH**	**TF**
Basin's toy example [BCSS11]	✓	✓	✓
Brands and Chaum [BC93]			
• Signature	✓	×	*o.o.s.*
• Fiat-Shamir	✓	×	×
CRCS No-revealing sign [RC10]			
• No-revealing sign	✓	✓	×
• Revealing sign	✓	×	×
Eff-PKDB [KV16]			
• No protection	✓	✓	✓
• Protected	✓	✓	✓
Hancke and Kuhn [3] [HK05]	✓	✓	×
MAD (One-Way) [ČBH03]	✓	×	*o.o.s.*
Meadows *et al.* [MPP+07]			
• $f := \langle n_V \oplus n_P, P \rangle$	✓$^{(n)}$	✓$^{(n)}$	×$^{(n)}$
• $f := \langle n_V, n_P \oplus P \rangle$	✓	×	×
• $f := \langle n_V, \mathsf{f}(n_P, P) \rangle$	✓	✓	×
• $f := \langle n_V, P, n_P \rangle$	✓	✓	×
Munilla *et al.* [MP08]	✓	✓	×
SKI [BMV13]	✓	✓*	✓
SPADE			
• Original [BGG+16]	×	×*	✓
• Fixed [Ger18]	✓$^{(n)}$	×$^{*(n)}$	✓$^{(n)}$
Swiss-Knife			
• Original [KAK+08]	✓	✓	✓
• Modified version [FO13]	✓	✓	×
TREAD asymmetric [ABG+17, Ger18]			
• Original (using $\mathsf{id}_{\mathsf{priv}}$)	×	×	✓
• Fixed (using $\mathsf{id}_{\mathsf{priv}}$)	✓$^{(n)}$	×$^{(n)}$	✓$^{(n)}$
• Original (using $\mathsf{id}_{\mathsf{pub}}$)	×	×	✓
• Fixed (using $\mathsf{id}_{\mathsf{pub}}$)	✓$^{(n)}$	×$^{(n)}$	✓$^{(n)}$
TREAD symmetric [ABG+17]	✓	×	✓
ISO/IEC 14443 protocols			
• MasterCard RRP [EMV16]	✓	×	×
• NXP [Jan17]	✓	×	×
• PaySafe [CGdR+15]	✓	×*	×

[1]Following our abstractions, the protocols Tree-based [AT09], Poulidor [TRMA10], and Uniform [MTPTR16] have the same modelling as the Hancke and Kuhn protocol.

Table 5.2: Results on our case studies (×: attack found, ✓: proved secure, *o.o.s.*: out of scope)
We use the following abbreviations/annotations: MF = Mafia Fraud, DH = Distance Hijacking, TF = Terrorist Fraud, * = returned in significantly more than few seconds, $^{(n)}$ = no symbolic analysis reported before.



(a) Brands and Chaum (b) MAD

Figure 5.4: Descriptions of Brands and Chaum [BC93], and MAD [ČBH03] protocols.

performed w.r.t. to a time bound which is sent by the prover during the execution. This implies that a message must contain a time (which is forbidden in our model). To model this feature, we followed the approach proposed by Chothia *et. al.* [CdRS18]: we replace the time-bound by an uninterpreted symbol of function $\mathtt{timebound}(id)$. We then prove that this value is correctly authenticated during a session. To do so, we add the time bound received by the verifier, and w.r.t. which the time check would be performed, in the end command and check that it corresponds to the time-bound of the authenticated prover. This is can be verified by the Proverif tool defining the following correspondence property:

$$\mathsf{end}(v_0, p_0, u) \Rightarrow (u = \mathtt{timebound}(p_0)).$$

The two protocols satisfy this property.

5.3.4. Comparison with Mauw *et al.*'s framework

In this section, we come back to the framework proposed by Mauw *et al.* [MSTPTR18, MSTPTR19] to conduct a deeper comparison. Even if our security definitions are in line with theirs (as already discussed in Section 2.4.2 - Chapter 2), we would like to discuss some verification aspects regarding each class of attacks separately.

Mafia fraud

Mauw *et al.* proposed the following *secure distance-bounding* property:

> A protocol is *distance-bounding secure* if for all trace of execution
>
> $$\mathsf{tr}_1.\ldots.\mathsf{tr}_n.\mathsf{claim}(V, P, x, y)$$
>
> where V is an honest agent then there $i, j \in \{1, \ldots, n\}$ such that:
>
> - $\mathsf{tr}_i = x$ and $\mathsf{tr}_j = y$; and
> - $(t_y - t_x) \leq 2 \cdot \mathsf{Dist}(V, P')$ with $P \approx P'$.

Remember that $P \approx P'$ means $P = P'$ if P is honest, and P' is any malicious agent otherwise. As discussed in Section 2.4.2, this security property does not distinguish between mafia and terrorist fraud. To do so, the authors proposed a slightly modified property in which the agent P is assumed to be honest.

The main result presented in [MSTPTR18] consists in proving the distance-bounding secure property equivalent to another property, solely relying on the order of the actions in the traces of execution, which can be checked relying on existing tools. This new property, called *dbsec* is as follows:

> A protocol satisfies *dbsec* if for all **untimed** trace of execution
>
> $$\mathsf{tr}_1.\ldots.\mathsf{tr}_n.\mathsf{claim}(V, P, x, y)$$
>
> where V is an honest agent then there $i, j, k \in \{1, \ldots, n\}$ such that $i < k < j$:
>
> - $\mathsf{tr}_i = x$ and $\mathsf{tr}_j = y$; and
> - $\mathsf{tr}_k = (a_k, \alpha_k)$ with $a_k \approx P$.

Informally, this property ensures that whenever a verifier V authenticates a prover P, then agent P must have executed an action during the rapid phase, delimited by actions x ($= \mathsf{tr}_i$) and y ($= tr_j$). In case agent P is malicious, this action may have been executed by another malicious agent (instead of himself). As previously, this security property can be slightly modified to focus on mafia fraud only. This variant is called *dbsec_hnst*. Checking the *dbsec_hnst* property with the Tamarin verification tool [MSCB13], Mauw *et al.* succeeded in proving the mafia fraud resistance of a number of protocols.

Distance hijacking

As discussed in Section 2.4.2, Mauw *et al.* cannot focus on distance hijacking attacks when a protocol suffers from a mafia fraud. However, when *dbsec_hnst* holds and *dbsec* does not, the authors can conclude that the protocol that is studied admits a distance hijacking attack.

Unfortunately, if the *dbsec_hnst* property can be immediately checked relying on existing tools, we do think that *dbsec* cannot. Actually, existing tools will check a close, but

slightly different property when the agent P is malicious. Indeed, the underlying procedures of existing tools implement some (crucial for termination purposes) optimisations that enable the attacker to perform actions that are not visible in the traces.

To illustrate this gap, let us first consider the Brands and Chaum protocol [BC93] presented in Figure 5.4a. The prover commits a value m to the verifier before the rapid phase. This last starts when the verifier sends a challenge c and the prover must reply $c \oplus m$. Finally, to be authenticated, the prover signs a transcript of the session and sends it to the verifier. As introduced in [CRSC12], this protocol admits a well-known distance hijacking attack, presented in Figure 5.5.

The gap appears when analysing this protocol in which the verifier role is slightly modified so that the challenge c is sent together with a dummy constant, i.e. $\langle c, const \rangle$. The prover role is left unchanged. In this situation, the verifier is sending a pair while the prover is waiting for an atomic challenge. We can thus note that without the presence of an attacker, who applies the first projection operator to remove the dummy constant, the prover cannot receive the challenge and thus the verifier can not successfully end a session with the prover (even if the prover is close to him). The help of the attacker is needed.

This new protocol matches all the requirements to be analysed in their framework, and the Tamarin tool returns that the dbsec property does not hold. Therefore, we conclude that there is a mafia fraud or a distance hijacking attack. According to us, there is none of them.

This gap has been acknowledged by the authors and simply requires to check whether the trace returned by the verification tool is a valid witness of attack or not. Hopefully, such situations have never been encountered when analysing existing distance-bounding protocols.

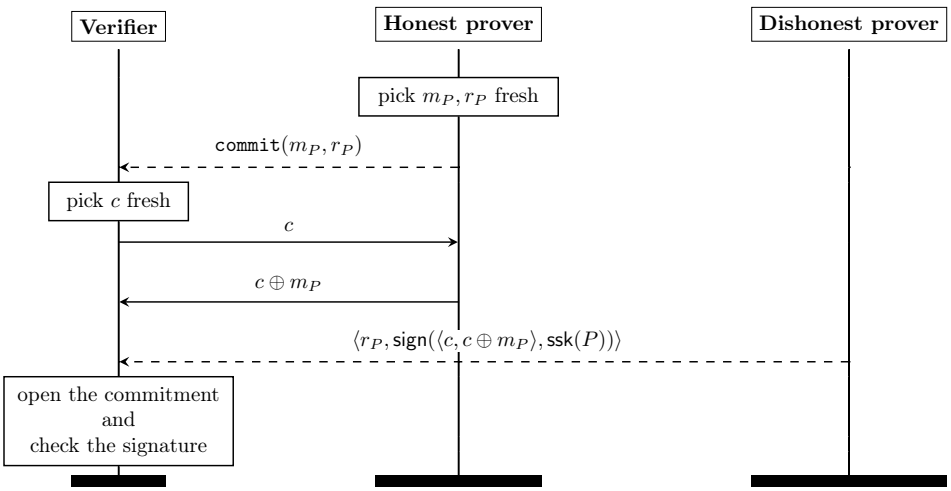


Figure 5.5: Distance hijacking attack against the Brands and Chaum protocol.

Terrorist fraud

Meanwhile we proposed our definition of terrorist frauds, Mauw *et al.* proposed theirs [MSTPTR19] which turned out to be completely in line with ours. We already discussed this in Section 2.4.2 but, here, we would like to discuss the analysis of the property within Tamarin. Especially because, at a first sight, they do not restrict the class of protocols which can be analysed, i.e. what we have called well-formed distance-bounding protocols.

In our framework, this restriction is due to the reduction result that allows to reduce the set of semi-dishonest provers. It turned out that they did not intend to perform such a reduction. Instead, in their framework, a manual proof is needed to reduce the possible collusion behaviours. This proof must be adapted for each protocol under study. Therefore, they provide a semi-automatic framework only, to analyse distance-bounding protocols w.r.t. terrorist fraud.

In some respects, our definition of well-formed distance-bounding protocols underlines a set of assumptions that a protocol must satisfy to dispense with the manual proof.

5.4. Conclusion

In conclusion, we showed how the results developed in Chapters 3 and 4 are used to analyse distance-bounding and contactless payment protocols. On a first hand, we implemented the procedure developed in Chapter 3 on top of the Akiss tool. It allows the analysis of protocols w.r.t. to a bounded number of sessions. This procedure has been proved sound and complete and terminates on almost all our case studies. It allows to look for attacks in a large class of protocols involving time but provides a limited confidence when no attack is found, due to the limited number of sessions that can be analysed. On the other hand, the framework based on the Proverif tool considers an unbounded number of sessions and thus provide a stronger guarantee when a protocol is proved secure. Unfortunately, this last requires more assumptions on the protocol under study and few distance-bounding protocols do not match the requirements.

Some limitations of each approach could be overcome, e.g. using parallelisation in our Akiss implementation. We could also probably relax the constraint about the freshness of the challenge in the Proverif model. However, we are also faced to some more limitations that seem to be more challenging to overcome: by-pass the interleaving blow-up by developing new POR optimisations in Akiss or extend our new procedure to take into account the exclusive-OR operator. This last is a cutting-edge area of research in symbolic verification: Akiss [BDGK17] and Tamarin [DHRS18] have recently been extended in this direction. On the first side, we hope that our procedure can be extended too but we fear an important impact on its efficiency. On the other side, we unfortunately encountered many non-termination issues due to the exclusive-OR operator when we tried to leverage Tamarin, instead of Proverif, to perform our case studies analysis. It seems that Mauw *et al.* have encountered similar issues when using the Tamarin tool to perform their analyses since roughly half of the protocols have been analysed considering a weak operator as described in Section 5.1.2. Improving the efficiency of existing procedure would thus be of great significance.

Part II

Verification of two novel EMV-payment protocols

Analysis of two novel EMV-payment protocols 6

In 2019, Chothia et al. *[CBC19] proposed two novel EMV-payment protocols. They claimed them resistant against relay attacks, even considering malicious readers. To achieve this strong security property, the protocols build on two specificities: first on a Trusted Platform Module (TPM) implemented on the reader side that is used to timestamp the rapid phase, then on the possibility to move the proximity check on the card or the bank side. Unfortunately, no existing model was suitable to analyse and prove the pretended security of these protocols.*

After a quick presentation of the two novel protocols, called PayBCR *and* PayCCR*, we will describe the model that will be used to analyse them. This last extends an existing one proposed by Mauw* et al. *[MSTPTR18]. We allow the proximity check to be performed by the reader (as usual), but also by the card or the bank. We also take into account mobility allowing agents to move across an execution. Taking some inspiration in [MSTPTR18], we define a security property solely relying on the causal order of the actions during an execution, and prove it equivalent to the extended security property. Finally, we analyse the two novel protocols and prove them secure.*

Contents

6.1. Two novel EMV-payment protocols

As mentioned in Chapter 1, one of the main security concerns in contactless payments is relay attacks: an EMV reader should not authenticate an honest EMV card which is far away, even if a man-in-the-middle attacker is located in-between and relays messages. To mitigate relay attacks, after 2016, Mastercard's EMV specification have included a distance-bounding protocol called PayPass-RRP. This protocol has been successfully proved secure against MiM attacks (i.e. mafia frauds) using symbolic models which include the two frameworks developed in the first part of this manuscript [CdRS18, DDW18, MSTPTR18, DD19].

The main underlying assumption of these analyses is that the reader behaves correctly and effectively performs the time/proximity check. In [CBC19], authors underlined that this assumption is too strong regarding payment protocols: a reader, i.e. a seller, may have incentives to take the payment, be it honest, relayed, or fraudulent in any way. This could lead to a new kind of collusion attacks in which the reader conspires with an attacker to accept relayed payments. It is important to note that this fraud is undetectable from the bank side since in the PayPass-RRP protocol, and in Visa's solution to protect against relays in contactless payments as well, the bank gets no evidence that the reader has performed a proximity check.

In this context, Chothia *et al.* [CBC19] proposed two novel EMV-payment protocols that build on the PayPass-RRP protocol. They have been designed to enforce physical proximity even considering rogue readers. They achieve this goal relying on a Trusted Platform Module (TPM) embedded in the reader to attest the timestamps used for the proximity check. These timestamps are sent to the card, or the bank, which can do the proximity check again, and thus prevent the reader from cheating (by detecting any fraud).

6.1.1. TPMs in a nutshell

A Trusted Platform Module (TPM) is a tamper-resistant hardware chip providing various functionalities, mainly of cryptographic nature. The functionality the two protocols rely on is the `TPM2_GetTime` command that is available since the TPM 2.0 specification provided by the Trusted Computing Group (TCG) that is in charge of standardising TPMs features. Given an input message u, the `TPM2_GetTime` command returns a signature over the message and an attested time:

$$\mathsf{sign}(\langle u, \texttt{TPM-AttesttedTime}\rangle, \mathsf{ssk}(\mathsf{tpm}))$$

with `TPM-AttestedTime` = (`Clock`, `Time`). Concretely, the `TPM-AttestedTime` is a timing data-structure that the TPM keeps: with the first being a non-volatile representation of the real time, set when the TPM is created [Gro16], and the second being a volatile value corresponding to the time passed since the last boot-up of the TPM (see page 205 of [Gro16]).

Since all the attacks which might apply against the `TPM-AttestedTime` are arguably impractical in practice, we assume that the timestamps given by the TPM via the command `TPM2_GetTime` are timing-secure, in the sense that an attacker cannot tamper the

timing data-structure. In particular, we assume that a TPM always signs the correct global time when requested.

6.1.2. Descriptions of PayBCR and PayCCR

The two novel EMV-payment protocols, called PayBCR and PayCCR, improve the existing PayPass-RRP protocol relying on TPMs. In order to perform the proximity check, the reader usually records the global time when outputting the challenge and compare it to the global time when receiving the answer. In PayBCR and PayCCR the reader makes two calls to a TPM which attests the two timestamps. These last are then transmitted to the bank (in PayBCR) or the card (in PayCCR) to let them perform the proximity check again. As an immediate consequence, if a reader decides to bypass the proximity check and accept any transaction, then it runs the risk to be caught by another party. Note that the authors proposed two protocols (instead of a unique one) for compatibility purposes with the PayPass-RRP protocol: PayBCR does not require to modify the card side, while PayCCR leaves the bank side unchanged.

Amongst the two protocols, we do think that PayBCR (which leaves the card side unchanged) is more likely to be implemented in the future. Modifying the bank process instead of the card one seems much less expensive for authorities. In the following of this chapter we will thus discuss the PayBCR protocol in priority.

A description of this protocol is presented in Figure 6.1. The reader starts by a query to the TPM to get a timestamp that initiates the rapid phase. This timestamp is forwarded to the card which answers with a fresh nonce n_C that will be given to the TPM in order to get a second timestamp and stop the rapid phase. The proximity check will be performed based on these two times t_1 and t_2. The remaining of the protocol is standard when looking at EMV protocols: the reader and the card exchange certificates and the card forges the AC and $SDAD$ messages based on its private keys (i.e. its asymmetric private signing key and the shared key with the bank). These two terms are used to check the integrity of the data and ensure mutual authentication. The reader then receives AC and $SDAD$ and can check the validity of the signature. Finally, the reader sends to the bank the AC message together with the necessary terms to check its validity. In addition it sends the two timestamps σ_1 and σ_2 to let the bank perform the proximity check again (in addition to standard verifications). At the end of a session, the bank must accept if, and only if, the card and the reader (or more precisely the TPM) were close during the transaction.

6.1.3. Existing models do not apply

As previously mentioned, the existing models do not apply to verify PayBCR and PayCCR. Indeed, they all implicitly assume two-party protocols and a proximity check entirely performed on the verifier, i.e. the reader, side. Obviously, the two-party protocol assumption is not satisfied by PayBCR and PayCCR since the bank plays an important role in the transactions. Moreover, the proximity check may be performed by either the card or the bank. Regarding the underlying model of the reduction results presented in Chapter 4, we cannot model checks performed by agents other than the verifier. Indeed,

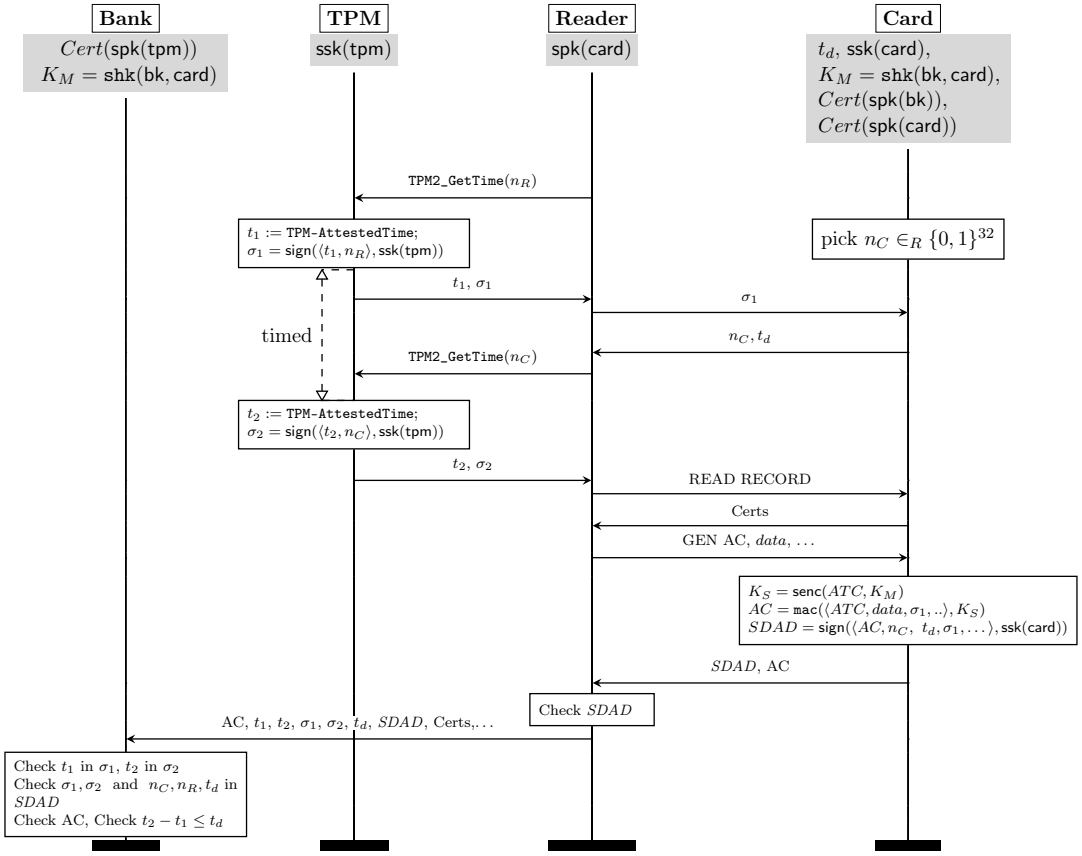


Figure 6.1: PayBCR [CBC19]: description of the protocol.

the proximity check is modelled by a `reset` action and the following guarded inputs. This enforces the agent performing the check to be the agent generating the timestamps. Unfortunately this is not what happens in PayBCR and PayCCR since, following its specification, a TPM cannot perform time checks.

When looking at Mauw *et al.* [MSTPTR18] model, this limitation is less clear but exists too. Indeed, to get rid of time and locations, thanks to the causality-based security property, authors require the claim event and the two timestamped actions to be executed by the same agent, i.e. the verifier. However, this security property appears more amendable than ours. In this chapter we will thus extend the result presented in [MSTPTR18].

6.2. A security model with mobility

In order to model such protocols, we are going to introduce in this section a calculus that embeds all the required features. This calculus is close to the one presented in Chapter 2. We therefore decided to solely focus on the differences. In particular, we will take mobility into account, i.e. we will assume that agents can move at anytime. Moreover we will present the security property proposed in [MSTPTR18] extended according to this new setting.

6.2.1. Messages and protocols

Regarding the way agents/messages are modelled, this is in line with what has been presented in Chapter 2. We assume, for sake of simplicity, that $\mathcal{A}$ are public constants and thus available to the attackers. We note $\Sigma_0^+ = \Sigma_0 \uplus \mathcal{A}$ to make this clear.

Regarding the protocol description, we define a process algebra close to the one introduced in Chapter 2:

$$\begin{array}{rcl} P, Q & := & 0 \\ & | & \mathtt{new}\ n.P \\ & | & \mathtt{in}(x).P \\ & | & \mathtt{out}(u).P \\ & | & \mathtt{let}_{\mathtt{mess}}\ x = v\ \mathtt{in}\ P \\ & | & \mathtt{gettime}(x).P \\ & | & \mathsf{check}(u_1, u_2, u_3).P \\ & | & \mathsf{claim}(u_1, u_2, u_3, u_4).P \end{array}$$

where $x \in \mathcal{X}$, $n \in \mathcal{N}$, $v \in \mathcal{T}(\Sigma, \mathcal{N} \cup \Sigma_0^+ \cup \mathcal{X} \cup \mathbb{R}_+)$, and $u, u_1, \ldots, u_4 \in \mathcal{T}(\Sigma_c, \mathcal{N} \cup \Sigma_0^+ \cup \mathcal{X} \cup \mathbb{R}_+)$.

Most of the commands remain the same compared to Chapter 2. We can note three differences: first, two new commands, $\mathsf{check}(u_1, u_2, u_3)$ and $\mathsf{claim}(u_1, u_2, u_3, u_4)$, are defined. These are events as usually defined in symbolic models. They will be used to state the security property presented below. Informally, the $\mathsf{check}(u_1, u_2, u_3)$ command will model that an agent would have performed the time check $u_2 - u_1 \leq u_3$ in real world (it thus replaces the $\mathtt{if}_{\mathtt{time}}$ command), and the $\mathsf{claim}(u_1, u_2, u_3, u_4)$ command means that a session has been successfully executed between agents u_1 and u_2 using the two timestamps u_3 and u_4 to perform the proximity check. The second difference lies in the input command. Indeed, the input/output commands are no longer annotated with a time variable to bind the current time when the action is executed. Instead, a new command `gettime` is defined for this purpose.

We then extended the definition of distance-bounding protocols considering multi-party protocols, i.e. any finite set of parametrised processes $\mathtt{P}(\mathsf{x}_0, \ldots, \mathsf{x}_n)$. In order to ensure that elements in $\mathbb{R}_+$ occurring in an execution have been either introduced by our `gettime` instruction, or by the attacker, we assume that the parametrised processes

do not contain any element in $\mathbb{R}_+$. This assumption will allow us to replace times by constants when getting rid of time.

Example 6.1. *The TPM functionality that is involved in the PayBCR and PayCCR protocols allows to get a signature of a pair made of an attested time and a message u given as input. It can be modelled through our syntax as follows:*

$$\begin{aligned}\mathsf{TPM}(\mathsf{x}_0) \;=\; & \mathtt{in}(x_n).\\ & \mathtt{gettime}(x_t).\\ & \mathtt{out}(\mathsf{sign}(\langle x_t, x_n\rangle, \mathsf{ssk}(\mathsf{x}_0))).\\ & 0\end{aligned}$$

The parameter x_0 will be instantiated by an agent name. Such a process is waiting for a message, and outputs its signature. The signature is done using the key $\mathsf{ssk}(\mathsf{x}_0)$ and a timestamp is added into the signature. The current time is obtained using the gettime *instruction.*

Similarly, we can define the roles $\mathsf{Bank}(\mathsf{x}_0)$, $\mathsf{Reader}(\mathsf{x}_0, \mathsf{x}_1, \mathsf{x}_2)$, and $\mathsf{Card}(\mathsf{x}_0, \mathsf{x}_1, \mathsf{x}_2)$ which respectively represent the commands executed by the bank, the reader and the card.

6.2.2. Mobility

In order to faithfully model the fact that transmitting a message takes time, the notion of topology has been defined in Chapter 2. It mainly defines the set of honest/dishonest agents involved in the configuration, and the location function $\mathsf{Loc} : \mathcal{A} \to \mathbb{R}^3$. Since we aim at modelling mobility, this location function must be extended to enable agents to move. We call a *mobility plan* a function that maps agent names and times to a location in the space. Moreover, to avoid unrealistic behaviours, like teleportation, we must ensure that agents do not move faster than the transmission speed, here the speed of light c_0. In the following definition we extend the notion of distance and locations introduced in Chapter 2.

Definition 6.1. *We note $\mathsf{Dist} : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}_+$ the distance function such that for any $\ell_1, \ell_2 \in \mathbb{R}^3$ we have:*

$$\mathsf{Dist}(\ell_1, \ell_2) = \frac{\|\ell_2 - \ell_1\|}{\mathsf{c}_0}$$

where $\|\cdot\|$ is the Euclidean norm.

A mobility plan Loc is a function $\mathsf{Loc} : \mathcal{A} \times \mathbb{R}_+ \to \mathbb{R}^3$ such that for any $a \in \mathcal{A}$ and $t_1, t_2 \in \mathbb{R}_+$ such that $t_1 \leq t_2$ we have:

$$\mathsf{Dist}(\mathsf{Loc}(a, t_1), \mathsf{Loc}(a, t_2)) \leq t_2 - t_1.$$

Example 6.2. *To illustrate the notion of mobility plan, we may consider the function Loc_0 such that $\mathsf{Loc}(\mathsf{tpm}_0, t) = (0, 0, 0)$, and $\mathsf{Loc}(\mathsf{bk}_0, t) = (100, 0, 0)$, and $\mathsf{Loc}(\mathsf{card}_0, t) = (|10 - t|, 0, 0)$, and for any $t \in \mathbb{R}_+$, and $\mathsf{Loc}(a, t) = (0, 0, 0)$ otherwise (i.e., for any $a \in \mathcal{A} \setminus \{\mathsf{tpm}_0, \mathsf{card}_0, \mathsf{bk}_0\}$). This mobility plan models a configuration in which all the agents are static except the card that is located at $(10, 0, 0)$ at time 0, then moves to the TPM's location, i.e. $(0, 0, 0)$, and finally leaves this location to come back at its original location and then keeps on moving in this direction.*

6.2.3. Semantics

Similarly to the semantics presented in Chapter 2, our new semantics is given by a labelled transition system over configurations $(\mathcal{P}; \Phi; t)$ that manipulates a multiset $\mathcal{P}$ of extended processes, an extended frame Φ (as introduced in Chapter 2), and the global time. However, to ease the formal development presented below, we immediately consider annotated labels (as introduced in Section 4.2.2 - Chapter 4). An extended process is thus denoted by $\lfloor P \rfloor_a^s$ with $a \in \mathcal{A}$ and P a process such that $fv(P) = \emptyset$, and s a unique session identifier. In addition, in order to indicate that a message is known by any agent, the elements in the extended frame can be annotated by the special symbol $\star$. This will be used to define the initial frame. A formal definition is provided below.

Definition 6.2. *A* configuration $\mathcal{K}$ *is a tuple* $(\mathcal{P}; \Phi; t)$ *where:*

- $\mathcal{P}$ *is a multiset of* extended processes $\lfloor P \rfloor_a^s$ *such that each session identifier* s *appears only once in* $\mathcal{P}$*;*
- $\Phi = \{\mathsf{w}_1 \xrightarrow{a_1,t_1} u_1, \ldots, \mathsf{w}_n \xrightarrow{a_n,t_n} u_n\}$ *is an* extended frame*, i.e., a substitution such that* $\mathsf{w}_i \in \mathcal{W}$*,* $u_i \in \mathcal{T}(\Sigma_c, \mathcal{N} \cup \Sigma_0^+ \cup \mathbb{R}_+)$*,* $a_i \in \mathcal{A} \cup \{\star\}$ *and* $t_i \in \mathbb{R}_+$ *for* $1 \leq i \leq n$*;*
- $t \in \mathbb{R}_+$ *is the global time of the system.*

Example 6.3. *A typical configuration for the PayBCR protocol is* $\mathcal{K}_0 = (\mathcal{P}_0; \Phi_0; 0)$ *with:*

$$\mathcal{P}_0 = \{ \lfloor \mathsf{TPM}(\mathsf{tpm}_0) \rfloor_{\mathsf{tpm}_0}^{s_1} ; \lfloor \mathsf{Bank}(\mathsf{bk}_0) \rfloor_{\mathsf{bk}_0}^{s_2} ; \lfloor \mathsf{Card}(\mathsf{card}_0) \rfloor_{\mathsf{card}_0}^{s_3} \}.$$

This simply models a scenario where tpm_0*,* bk_0*, and* card_0 *execute a single session of their role. Regarding the initial frame, we may assume that it contains the private key of a malicious card* att_0 *and its certificate built by the bank together with those of the honest card and the honest TPM. Formally, the initial frame is:*

$$\begin{aligned} \Phi_0 = \{ \quad & \mathsf{w}_1 \xrightarrow{\star,0} \mathsf{ssk}(\mathsf{att}_0), \\ & \mathsf{w}_2 \xrightarrow{\star,0} \mathsf{sign}(\langle \mathsf{cardCert}, \langle \mathsf{att}_0, \mathsf{spk}(\mathsf{att}_0) \rangle \rangle, \mathsf{ssk}(\mathsf{bk}_0)) \\ & \mathsf{w}_3 \xrightarrow{\star,0} \mathsf{sign}(\langle \mathsf{cardCert}, \langle \mathsf{card}_0, \mathsf{spk}(\mathsf{card}_0) \rangle \rangle, \mathsf{ssk}(\mathsf{bk}_0)) \\ & \mathsf{w}_4 \xrightarrow{\star,0} \mathsf{sign}(\langle \mathsf{tpmCert}, \langle \mathsf{tpm}_0, \mathsf{spk}(\mathsf{tpm}_0) \rangle \rangle, \mathsf{ssk}(\mathsf{bk}_0)) \}. \end{aligned}$$

The terms cardCert *and* tpmCert *are public constants from* Σ_0 *used to avoid a possible confusion between the two kinds of certificates (the one issued by the bank to certify a card, and the one used to certify a TPM). In case such a confusion is possible, we may model the two types of certificates relying on the same constant* cert*.*

Before we present our operational semantics we must define the deduction capabilities of an agent which is necessary to define the semantics of the input command. Since messages take time to travel from one location to another, messages are not immediately available to any agent through the frame. The set of terms that an agent is able to deduce depends on the extended frame but also the mobility plan. Given an extended frame Φ, and a mobility plan Loc, we say that a term u is *deducible from* Φ *by* $b \in \mathcal{A}$ *at time* t_b *using the recipe* R, denoted $\Phi \vdash^R u$ by b at time t_b w.r.t. Loc, if $R\Phi\downarrow =_{\mathrm{E}} u$, and for all $\mathsf{w} \in vars(R)$ we have that $(\mathsf{w} \xrightarrow{c,t} v) \in \Phi$ for some v, and

- either $c = \star$;
- or $t_b \geq t + \mathsf{Dist}(\mathsf{Loc}(c,t), \mathsf{Loc}(b,t_b))$.

In other words, u has to be forgeable by the agent b at time t_b and thus messages needed to forge u have to be available in due time. This definition is in line with the one introduced in Chapter 2, taking care of the time at which each message has been outputted to consider the corresponding locations.

$$\text{TIM} \quad (\mathcal{P};\Phi;t) \longrightarrow_{\mathsf{Loc}} (\mathcal{P};\Phi;t+\delta) \qquad \text{with } \delta > 0$$

$$\text{NEW} \quad (\lfloor \mathtt{new}\ n.P \rfloor_a^s \uplus \mathcal{P};\Phi;t) \xrightarrow{a,\tau,s,t,\emptyset}_{\mathsf{Loc}} (\lfloor P\{n \mapsto n'\} \rfloor_a^s \uplus \mathcal{P};\Phi;t) \qquad \text{with } n' \in \mathcal{N} \text{ fresh}$$

$$\text{OUT} \quad (\lfloor \mathtt{out}(u).P \rfloor_a^s \uplus \mathcal{P};\Phi;t) \xrightarrow{a,\mathtt{out}(u),s,t,\mathsf{w}}_{\mathsf{Loc}} (\lfloor P \rfloor_a^s \uplus \mathcal{P};\Phi \uplus \{\mathsf{w} \xrightarrow{a,t} u\};t) \qquad \text{with } \mathsf{w} \in \mathcal{W} \text{ fresh}$$

$$\text{LET} \quad (\lfloor \mathtt{let}_{\mathtt{mess}}\ x = v\ \mathtt{in}\ P \rfloor_a^s \uplus \mathcal{P};\Phi;t) \xrightarrow{a,\tau,s,t,\emptyset}_{\mathsf{Loc}} (\lfloor P\{x \mapsto v{\downarrow}\} \rfloor_a^s \uplus \mathcal{P};\Phi;t) \qquad \text{when } v{\downarrow} \in \mathcal{T}(\Sigma_c, \mathcal{N} \cup \Sigma_0^+ \cup \mathbb{R}_+)$$

$$\text{CLM} \quad (\lfloor \mathsf{claim}(u_1,u_2,u_3,u_4).P \rfloor_a^s \uplus \mathcal{P};\Phi;t) \xrightarrow{a,\mathsf{claim}(u_1,u_2,u_3,u_4),s,t,\emptyset}_{\mathsf{Loc}} (\lfloor P \rfloor_a^s \uplus \mathcal{P};\Phi;t)$$

$$\text{CHK} \quad (\lfloor \mathsf{check}(u_1,u_2,u_3).P \rfloor_a^s \uplus \mathcal{P};\Phi;t) \xrightarrow{a,\mathsf{check}(u_1,u_2,u_3),s,t,\emptyset}_{\mathsf{Loc}} (\lfloor P \rfloor_a^s \uplus \mathcal{P};\Phi;t)$$

$$\text{GTM} \quad (\lfloor \mathtt{gettime}(x).P \rfloor_a^s \uplus \mathcal{P};\Phi;t) \xrightarrow{a,\mathtt{gettime},s,t,\emptyset}_{\mathsf{Loc}} (\lfloor P\{x \mapsto t\} \rfloor_a^s \uplus \mathcal{P};\Phi;t') \qquad \text{with } t' > t$$

$$\text{IN} \quad (\lfloor \mathtt{in}(x).P \rfloor_a^s \uplus \mathcal{P};\Phi;t) \xrightarrow{a,\mathtt{in}(u),s,t,(b,t_b,R)}_{\mathsf{Loc}} (\lfloor P\{x \mapsto u\} \rfloor_a^s \uplus \mathcal{P};\Phi;t)$$

when $u \in \mathcal{T}(\Sigma_c, \mathcal{N} \cup \Sigma_0^+ \cup \mathbb{R}_+)$ and there exist $b \in \mathcal{A}$, and $t_b \in \mathbb{R}_+$ such that $t \geq t_b + \mathsf{Dist}(\mathsf{Loc}(b,t_b), \mathsf{Loc}(a,t))$ and:

- either $b \in \mathcal{A} \setminus \mathcal{M}$ and there exists $(\mathsf{w} \xrightarrow{b,t_b} u) \in \Phi$, i.e. $\Phi \vdash^R u$ with $R = \mathsf{w}$;
- or $b \in \mathcal{M}$ and $\Phi \vdash^R u$ by b at time t_b w.r.t. Loc for some recipe R.

Figure 6.2: Semantics of our calculus parametrised by the mobility plan Loc.

The semantics of processes, presented in Figure 6.2, is in line with the one presented in Chapter 2. We can nevertheless make some comments. First, one may note that the check and claim commands are executed as events, i.e. they can always be executed and do not modify the current configuration. They simply add a visible action in the trace. Then, another noticeable point is the gettime command. As expected it binds the variable x to the current global time t but it also enforces the global time to elapse, i.e.

$t' > t$. This restriction imposes that two `gettime` commands cannot be executed at the exact same time. It helps us to establish the soundness of our main result (see Section 6.3) by making it possible for us to get rid of time by replacing real numbers issuing from a `gettime` instruction by different public constants. Note that this does not seem too restrictive thanks to the high accuracy of the TPM's clock in real applications. Finally, the labels are more expressive than those presented in Chapter 2. They correspond to the annotations introduced in Chapter 4. A label is thus a tuple (a, α, s, t, r) where a is the name of the agent executing the action α from the extended process identified by the session identifier s at time t. Moreover, when α is an output then $x = \mathsf{w}$ the frame variable that is bound, and when α is an input then $r = (b, t_b, R)$ where R is the recipe used by the agent b at time t_b to forge the input message. Otherwise, $r = \emptyset$.

Example 6.4. *To illustrate the semantics, we consider the simple configuration made of a unique session of a TPM, and the initial frame as defined in Example 6.3, i.e.* $\mathcal{K}'_0 = (\lfloor \mathsf{TPM}(\mathsf{tpm}_0) \rfloor^s_{\mathsf{tpm}_0}; \Phi_0; 0)$. *We also consider the mobility plan* Loc_0 *as given in Example 6.2. We may obtain the following execution:*

$$\mathcal{K}'_0 \rightarrow (\lfloor \mathsf{TPM}(\mathsf{tpm}_0) \rfloor^s_{\mathsf{tpm}_0}; \Phi_0; 1.1)$$
$$\xrightarrow{\mathsf{tpm}_0,\mathtt{in}(\mathsf{ok}),s,1.1,(c,1.1,\mathsf{ok})} \xrightarrow{\mathsf{tpm}_0,\mathtt{gettime},s,1.1,\emptyset} (\lfloor \mathtt{out}(m) \rfloor^s_{\mathsf{tpm}_0}; \Phi_0; 2)$$
$$\xrightarrow{\mathsf{tpm}_0,\mathtt{out}(m),s,2,\mathsf{w}_5}_{\mathsf{Loc}_0} (\lfloor 0 \rfloor^s_{\mathsf{tpm}_0}; \Phi_1; 2)$$

where:

- $m = \mathsf{sign}(\langle 1.1, \mathsf{ok} \rangle, \mathsf{seck}(\mathsf{tpm}_0))$, *and*
- $\Phi_1 = \Phi_0 \cup \{\mathsf{w}_5 \xrightarrow{\mathsf{tpm}_0,2} m\}$.

The first input is possible since $\Phi_0 \vdash^{\mathsf{ok}} \mathsf{ok}$ *by any* $c \in \mathcal{M}$ *at time* 1.1. *Actually such a constant is even deducible at time* 0.

We may note that m *is deducible from* Φ_1 *by* tpm_0 *at time* 2. *Actually any agent other than* card_0 *and* bk_0 *are able to deduce* m *at time* 2. *Remember that* $\mathsf{Loc}_0(a, t) = (0, 0, 0)$ *for any* t *and any* a *different from* card_0 *and* bk_0. *The agent* bk_0 *has to wait* $t = 100$ *to be able to receive message* m, *and agent* card_0 *has to wait* $t = 6$.

6.2.4. Distance-bounding security

In order to prove secure EMV-payment protocols, we aim at defining a security property that is able to detect relay attacks/mafia frauds. Moreover, the security property must handle the use of TPMs and proximity checks performed by an agent different from the verifier.

Before we give the formal definition of our security property, we need to define the set of initial configurations with respect to which the protocols will be analysed. Such configurations should only involve an *initial frame*, i.e. a frame that contains the initial knowledge of the dishonest agents. Unlike in the previous model, we do not make any assumption on this knowledge. Therefore, an initial frame is simply an extended frame Φ_0 such that $a = \star$ and $t = 0$ for any $(\mathsf{w} \xrightarrow{a,t} u) \in \Phi_0$.

Definition 6.3. *A configuration* $\mathcal{K} = (\mathcal{P}_0; \Phi_0; 0)$ *is a* valid initial configuration *for a set of roles* $\mathcal{R}$*, if* Φ_0 *is an initial frame, and for each* $\lfloor P \rfloor_a \in \mathcal{P}_0$*, there exists* $R(\mathsf{x}_0, \mathsf{x}_1, \ldots, \mathsf{x}_k) \in \mathcal{R}$ *and* $a_1, \ldots, a_k \in \mathcal{A}$ *such that* $P = R(a, a_1, \ldots, a_k)$*.*

Roughly, we consider initial configurations made up of instances of the roles of the protocols, and we only consider roles executed by agents located at the right place, i.e., the agent a who executes the role must correspond to the first argument of the parametrised process. Note that, according to the definition above, a single agent can play different roles (e.g., the bank and the card role by a unique agent).

Example 6.5. *The frame* Φ_0 *described in Example 6.3 is an initial frame, and the configuration* $\mathcal{K}_0$*, as well as the configuration* $\mathcal{K}_0'$ *given in Example 6.4, are valid initial configurations for the protocol* PayBCR*. Although valid, these configurations are rather poor and additional scenarios will be considered when performing the security analysis. Typically, we will consider many agents, and we will assume that each agent can execute the protocol many times.*

The security property, we are going to present in Definition 6.4, relies on two events:

- $\mathsf{check}(t_1, t_2, \delta)$ means that the timing constraint $t_2 - t_1 \leq \delta$ has been verified by some honest agent;
- $\mathsf{claim}(\mathsf{tpm}_0, \mathsf{card}_0, t_1^0, t_2^0)$ means that an honest agent (typically a bank) finishes the protocol, seemingly with the two agents tpm_0 and card_0 which pretend be close between the two times t_1^0 and t_2^0.

Our result applies on all the valid initial configurations. However, to give the possibility to precisely define the set of initial configurations we want to consider during the security analysis, our definition is thus parametrised by a set $\mathcal{S}$ of valid initial configurations.

Definition 6.4. *A protocol* $\mathcal{P}_{\mathsf{db}}$ *is* DB-secure *w.r.t. a set* $\mathcal{S}$ *of valid initial configurations if for all* $\mathcal{K}_0 \in \mathcal{S}$*, for all mobility plans* Loc*, and for all executions* exec *such that:*

$$\mathcal{K}_0 \xrightarrow{(a_1,\alpha_1,s_1,t_1,r_1)\ldots(a_n,\alpha_n,s_n,t_n,r_n)\cdot(b_0,\mathsf{claim}(b_1,b_2,t_1^0,t_2^0),s,t,r)}_{Loc} \mathcal{K}$$

we have that:

- *either* $b_1 \in \mathcal{M}$*, or* $b_2 \in \mathcal{M}$*;*
- *or* $\alpha_k = \mathsf{check}(t_1^0, t_2^0, t_3^0)$ *for some* $k \leq n$ *such that:*

$$t_2^0 - t_1^0 \geq \mathit{Dist}(\mathit{Loc}(b_1, t_1^0), \mathit{Loc}(b_2, t)) + \mathit{Dist}(\mathit{Loc}(b_2, t), \mathit{Loc}(b_1, t_2^0))$$

for some $t_1^0 \leq t \leq t_2^0$*.*

Informally, this security property ensures that whenever an agent b_0 (typically a bank) ends the protocol with two agents b_1 (the TPM) and b_2 (the card) then they must have been close during the two times t_1^0 and t_2^0, and a time check must have been performed using this two times.

One may note that the security property trivially holds when agents b_1 and/or b_2 are dishonest. This would represent scenarios in which the TPM and/or the card are malicious. In the first case, the malicious TPM is able to tamper the generated timestamps. No guarantee can thus be ensured. In the second case, the card is able to collude with any other malicious agent and, in particular, with such an agent that is in the vicinity of the TPM. Hence, no guarantee can be ensured neither.

Comparison to Mauw *et al.*'s definition As previously mentioned, this security property extends Mauw *et al.*'s [MSTPTR18]. We note two differences:

- Due to mobility, we must be more precise when talking about the distance from b_1 and b_2. Therefore the security says that between the two times t_1^0 and t_2^0, the agent b_2 has been close to the location of b_1 at time t_1^0 and the location of b_1 at time t_2^0. These two locations might be different.

- In standard distance-bounding protocols, the verifier is the agent who initiates the rapid phase, performs the time checks, and makes the final claim. Unfortunately, the PayBCR and PayCCR do not follow this structure. The rapid phase is initiated by the TPM, the time check is either performed by the bank or the card, and the final claim is made by the bank. In order to model these two protocols we thus allow to consider three different agents, i.e. b_0, b_1 and b_2, to execute these actions.

6.3. Getting rid of time and locations

In this section we will define the causality-based security property against which the protocols will be analysed. This property solely relies on the order of the actions in an execution, and is out of time and location considerations. We can thus express this property in an untimed semantics to get as close as possible to the existing tools, e.g. Proverif, that will be used to analyse protocols. Once formally defined, we will prove its equivalence to the secure-DB property introduced in Definition 6.4.

6.3.1. Causality-based security

The untimed semantics operates over untimed configurations, i.e. pairs of the form $(\mathcal{P};\phi)$ where $\mathcal{P}$ is a multiset of extended processes and ϕ is a frame (i.e. an extended frame without time annotations). Since the multiset of extended processes $\mathcal{P}$ must not contain times, i.e. non negative real numbers, the `gettime` command is replaced by a $\mathsf{timestamp}(c_x)$ event where $c_x \in \Sigma_0^{\mathsf{spe}}$ a subset of Σ_0 which contains specific variables used to abstract times.

Given a configuration $\mathcal{K} = (\mathcal{P};\Phi;t)$, for sake of simplicity, we assume that variables occurring in $\mathcal{P}$ are at most bound once and we note $\overline{\cdot}$ the transformation such that $\overline{\mathcal{K}} = (\mathcal{P}_0;\phi_0)$ where:

- $\mathcal{P}_0$ is the untimed counterpart of $\mathcal{P}$, i.e. each $\texttt{gettime}(x)$ instruction occurring in $\mathcal{P}$ is replaced by $\mathsf{timestamp}(c_x)$ where $c_x \in \Sigma_0^{\mathsf{spe}}$, and the occurrences of x in the remaining process are replaced by c_x;

- ϕ_0 is the untimed counterpart of Φ, i.e. $\phi_0 = \{\mathsf{w} \to u \mid (\mathsf{w} \xrightarrow{c,t} u) \in \Phi\}$.

The untimed semantics

The untimed semantics presented in Figure 6.3 is a labelled transition system over untimed configurations. Comparing to the timed semantics, presented in Figure 6.2, the TIM rule does not exist anymore. The rules NEW, OUT and LET are straightforwardly adapted removing the global time from the configuration (and the time annotation from frame for the OUT rule). The rule GTM is modified to become a simple event instruction. Finally, the IN rule no longer cares about time.

We note that the untimed semantics embeds annotated labels of the form (a, α, s, r) where a is a name of the agent executing action α from the process identified by the session identifier s. Moreover, when the action is an output then $r = \mathsf{w}$ the frame variable which is bound, and when the action is an input then $r = (b, R)$ where b is the agent who forges the input message and R the recipe that is used. These annotated labels are the untimed counterpart of the timed labels.

Causality-based security

We are now able to define a security property that solely relies on the order of the actions during an execution. This property, inspired from [MSTPTR18], claims a protocol secure if, whenever two agents b_1 and b_2 are authenticated relying on the two abstracted times c_1 and c_2, a time check must have been performed and agent b_2 must have been active between the two events.

Definition 6.5. *A protocol $\mathcal{P}_{\mathsf{db}}$ is* causality-based secure *w.r.t. a set $\mathcal{S}$ of valid initial configurations, if for all $\mathcal{K}_0 \in \mathcal{S}$, for all execution* exec *such that:*

$$\overline{\mathcal{K}_0} \xrightarrow{(a_1,\alpha_1,s_1,r_1)\ldots(a_n,\alpha_n,s_n,r_n)\cdot(b_0,\mathsf{claim}(b_1,b_2,c_1,c_2),s,r)} (\mathcal{P}'; \phi')$$

we have that either $b_1 \in \mathcal{M}$, or $b_2 \in \mathcal{M}$, or there exist $i, j, k, k' \leq n$ with $i \leq k' \leq j$, and $u \in \mathcal{T}(\Sigma_c, \mathcal{N} \cup \Sigma_0^+)$ such that:

- $\alpha_k = \mathsf{check}(c_1, c_2, u)$;
- $(a_i, \alpha_i) = (b_1, \mathsf{timestamp}(c_1))$;
- $(a_j, \alpha_j) = (b_1, \mathsf{timestamp}(c_2))$; *and*
- $a_{k'} = b_2$.

This property is closely related to the DB-secure property presented in Definition 6.4. The difference lies in the fact that the proximity is no longer enforced by checking the

$$\text{NEW}' \quad (\lfloor \mathtt{new}\ n.P \rfloor_a^s \uplus \mathcal{P}; \phi) \overset{a,\tau,s,\emptyset}{\rightsquigarrow} (\lfloor P\{n \to n'\} \rfloor_a^s \uplus \mathcal{P}; \phi) \quad \text{with } n' \in \mathcal{N} \text{ fresh}$$

$$\text{OUT}' \quad (\lfloor \mathtt{out}(u).P \rfloor_a^s \uplus \mathcal{P}; \phi) \overset{a,\tau,s,\mathsf{w}}{\rightsquigarrow} (\lfloor P \rfloor_a^s \uplus \mathcal{P}; \phi \uplus \{\mathsf{w} \to u\}) \quad \text{with } \mathsf{w} \in \mathcal{W} \text{ fresh}$$

$$\text{LET}' \quad (\lfloor \mathtt{let}_{\mathtt{mess}}\ x = v\ \mathtt{in}\ P \rfloor_a^s \uplus \mathcal{P}; \phi) \overset{a,\tau,s,\emptyset}{\rightsquigarrow} (\lfloor P\{x \to v{\downarrow}\} \rfloor_a^s \uplus \mathcal{P}; \phi) \quad \text{when } v{\downarrow} \in \mathcal{T}(\Sigma_c, \mathcal{N} \cup \Sigma_0^+)$$

$$\text{CLM}' \quad (\lfloor \mathsf{claim}(u_1, u_2, u_3u_4).P \rfloor_a^s \uplus \mathcal{P}; \phi) \overset{a,\mathsf{claim}(u_1,u_2,u_3,u_4),s,\emptyset}{\rightsquigarrow} (\lfloor P \rfloor_a^s \uplus \mathcal{P}; \phi)$$

$$\text{CHK}' \quad (\lfloor \mathsf{check}(u_1, u_2, u_3).P \rfloor_a^s \uplus \mathcal{P}; \phi) \overset{a,\mathsf{check}(u_1,u_2,u_3),s,\emptyset}{\rightsquigarrow} (\lfloor P \rfloor_a^s \uplus \mathcal{P}; \phi)$$

$$\text{GTM}' \quad (\lfloor \mathsf{timestamp}(c_x).P \rfloor_a^s \uplus \mathcal{P}; \phi) \overset{a,\mathsf{timestamp}(c_x),s,\emptyset}{\rightsquigarrow} (\lfloor P \rfloor_a^s \uplus \mathcal{P}; \phi)$$

$$\text{IN}' \quad (\lfloor \mathtt{in}(x).P \rfloor_a^s \uplus \mathcal{P}; \phi) \overset{a,\mathtt{in}(u),s,(b,R)}{\rightsquigarrow} (\lfloor P\{x \mapsto u\} \rfloor_a^s \uplus \mathcal{P}; \phi) \quad \text{when } R\phi{\downarrow} =_{\mathtt{E}} u \text{ for some recipe } R$$

when $u \in \mathcal{T}(\Sigma_c, \mathcal{N} \cup \Sigma_0^+)S$ and there exist $b \in \mathcal{A}$, such that:

- either $b \in \mathcal{A} \setminus \mathcal{M}$ $R = \mathsf{w}$ such that $(\mathsf{w} \xrightarrow{u}) \in \phi$;
- or $b \in \mathcal{M}$ and $R\phi{\downarrow} =_{\mathtt{E}} u$ for some recipe R.

Figure 6.3: Untimed semantics

distance of the agents, but ensuring that agent b_2 has been active between the two events.

This property requires the existence of timestamp events as soon as a claim appears in an execution to identify the subpart of the execution which corresponds to the rapid phase of the protocol. We introduce such a requirement as an hypothesis. Note that we do not explicitly require the uniqueness of these events since it is enforced by the semantic rules: the global time must increase when executing a `gettime` command.

Definition 6.6. *A protocol* $\mathcal{P}_{\mathsf{db}}$ *is* well-timed *w.r.t. a set* $\mathcal{S}$ *of valid initial configurations if for all* $\mathcal{K}_0 \in \mathcal{S}$*, for all execution* exec *such that:*

$$\overline{\mathcal{K}_0} \overset{(a_1,\alpha_1,s_1,r_1)\ldots(a_n,\alpha_n,s_n,r_n)\cdot(b_0,\mathsf{claim}(b_1,b_2c_1,c_2),s,r)}{\rightsquigarrow} (\mathcal{P}'; \phi')$$

we have that there exist $i, j \leq n$ *such that:*

- $(a_i, \alpha_i) = (b_1, \mathsf{timestamp}(c_1))$*;*
- $(a_j, \alpha_j) = (b_1, \mathsf{timestamp}(c_2))$.

We are now able to state our main result that establishes the equivalence between the two properties, i.e. DB-security and causality-based security. The purpose of the next two sections is to prove this theorem.

Theorem 6.1. *Let $\mathcal{P}_{\mathsf{db}}$ be a protocol and $\mathcal{S}$ be a set of valid initial configurations. Assuming that $\mathcal{P}_{\mathsf{db}}$ is well-timed w.r.t. $\mathcal{S}$, we have that:*

$$\mathcal{P}_{\mathsf{db}} \textit{ is DB-secure w.r.t. } \mathcal{S}$$
$$\textit{if, and only if,}$$
$$\mathcal{P}_{\mathsf{db}} \textit{ is causality-based secure w.r.t. } \mathcal{S}.$$

6.3.2. From distance-bounding security to causality-based security

This section is devoted to the proof of the first implication of the main theorem.

Proposition 6.1. *Let $\mathcal{P}_{\mathsf{db}}$ be a protocol and $\mathcal{S}$ be a set of valid initial configurations. Assuming that $\mathcal{P}$ is well-timed w.r.t. $\mathcal{S}$, and $\mathcal{P}_{\mathsf{db}}$ is DB-secure, we have that $\mathcal{P}_{\mathsf{db}}$ is causality-based secure.*

Intuitively, the proof of this proposition is quite easy and consists in re-timing a witness of attack w.r.t. the causality-based security. Such a re-timing process consists in assigning a time to each action occurring in the execution. This assignment must satisfy a set of constraints that can be derived from the execution and the mobility plan in which we aim at re-timing the execution. We introduce the notion of timed formula $\mathcal{C}^{\mathsf{Loc}}_{exec}$ associated to an annotated untimed execution exec, and a mobility plan Loc.

Given a trace $\mathsf{tr}_1 \dots \mathsf{tr}_n$ we note $\mathsf{IN}(\mathsf{tr}_1 \dots \mathsf{tr}_n)$ the set of all the indices corresponding to input actions. Similarly we note $\mathsf{TS}(\mathsf{tr}_1 \dots \mathsf{tr}_n)$ the set of all the indices corresponding to timestamp events. Given a set S we note $\#S$ its size. Finally, we note $\mathtt{orig}(i)$ the index of the i$^{\text{th}}$ output in the trace.

Given an annotated execution $\mathsf{exec} = \mathcal{K}_0 = (\mathcal{P}_0; \Phi_0) \overset{\mathsf{tr}_1 \dots . \mathsf{tr}_n}{\rightsquigarrow} \mathcal{K}_n$ of size n (with $\mathsf{tr}_i = (a_i, \alpha_i, s_i, r_i)$ for $1 \leq i \leq n$ and $r_i = (b_i, R_i)$ for $i \in \mathsf{IN}(\mathsf{tr}_1, \dots, \mathsf{tr}_n)$) and a mobility plan Loc, the timed formula $\mathcal{C}^{\mathsf{Loc}}_{\mathsf{exec}}$, built upon the set of variables $Z = \{z_1, \dots, z_n\} \cup \{z_i^b \mid i \in \mathsf{IN}(\mathsf{tr}_1, \dots, \mathsf{tr}_n)\}$, is the conjunction of the following formulas:

- $0 \leq z_1 \leq z_2 \leq \dots \leq z_n$;
- $z_i < z_{i+1}$ for all $i \in \mathsf{TS}(\mathsf{tr}_1, \dots, \mathsf{tr}_{n-1})$;
- $z_i \geq z_i^b + \mathsf{Dist}(\mathsf{Loc}(a_i, z_i), \mathsf{Loc}(b_i, z_i^b))$ for all $i \in \mathsf{IN}(\mathsf{tr}_1, \dots, \mathsf{tr}_n)$;
- for all $i \in \mathsf{IN}(\mathsf{tr}_1, \dots, \mathsf{tr}_n)$, for all j such that $\mathsf{w}_j \in vars(R_i) \setminus dom(\Phi_0)$,

$$z_i^b \geq z_{\mathtt{orig}(j)} + \mathsf{Dist}(\mathsf{Loc}(b_i, z_i^b), \mathsf{Loc}(a_{\mathtt{orig}(j)}, z_{\mathtt{orig}(j)})).$$

The first item models that the global time cannot decrease along an execution. The second item ensures that the global time increases when executing `gettime` commands

as required in the semantic rule. The third and the fourth items gather the timing constraints corresponding to the input rule: the third item ensures that the input message can be forged soon enough, and the fourth item ensures that the agent who forges the input message can receive all the material on time.

Since this formula has been defined to contain all the timing constraints that an assignment must satisfy to lift a trace from the untimed to the timed semantics, the following lemma holds.

Lemma 6.1. *Let $\mathcal{P}_{\mathsf{db}}$ be a protocol and $\mathcal{K}_0 = (\mathcal{P}_0; \Phi_0; 0)$ be a valid initial configuration. Let Loc be a mobility plan.*

For any execution $\mathsf{exec} = \overline{\mathcal{K}_0} \overset{\mathsf{tr}_1 \dots \mathsf{tr}_n}{\rightsquigarrow} \mathcal{K}_n$ with $\mathsf{tr}_i = (a_i, \alpha_i, s_i, r_i)$ and function φ satisfying $\mathcal{C}^{\mathsf{Loc}}_{\mathsf{exec}}$ we have $\mathcal{K}_0 \xrightarrow{\mathsf{tr}'_1 \dots \mathsf{tr}'_n}_{\mathsf{Loc}} \mathcal{K}'_n$ with:

$$\mathsf{tr}'_i = \begin{cases} (a_i, \mathtt{gettime}, s_i, \varphi(z_i), \emptyset) & \text{if } \alpha_i = \mathsf{timestamp}(c_i) \\ (a_i, \alpha_i\varphi_c, s_i, \varphi(z_i), r_i\varphi_c) & \text{otherwise} \end{cases}$$

and $\varphi_c(c_i) = \varphi(z_i)$ for all $i \in \mathsf{TS}(\mathsf{tr}_1 \dots \mathsf{tr}_n)$. Moreover, $\overline{\mathcal{K}'_n}\varphi_c = \mathcal{K}_n$.

Proof. The proof follows the definition of $\mathcal{C}^{\mathsf{Loc}}_{\mathsf{exec}}$. Indeed this formula contains all the timing constraints required to trigger each action. Moreover, by definition of $\mathcal{C}^{\mathsf{Loc}}_{\mathsf{exec}}$ we obtain that φ_c is a bijective function as soon as variables are bound at most once in $\mathcal{K}_0$. This preserves equalities and inequalities between the untimed and the timed execution. □

Proof of Proposition 6.1

Proof. Relying on the previous lemma, we are able to prove Proposition 6.1. Assuming that the protocol $\mathcal{P}_{\mathsf{db}}$ is not causality-based secure, the proof proceeds in three steps depending on which item of the definition is falsified.

Since $\mathcal{P}_{\mathsf{db}}$ is not causality-based secure, we know that there exist a valid initial configuration $\mathcal{K}_0 = (\mathcal{P}_0; \Phi_0; 0) \in S$ and an execution exec such that:

$$\mathsf{exec} = \overline{\mathcal{K}_0} \overset{\mathsf{tr}_1 \dots \mathsf{tr}_n.(b_0, \mathsf{claim}(b_1, b_2, c_1^0, c_2^0), s, \emptyset)}{\rightsquigarrow} (\mathcal{P}'; \phi')$$

with $\mathsf{tr}_i = (a_i, \alpha_i, s_i, r_i)$ $(1 \leq i \leq n)$ and $b_1 \notin \mathcal{M}$, $b_2 \notin \mathcal{M}$ and either:

1. there is no $k \leq n$ such that $\alpha_k = \mathsf{check}(c_1^0, c_2^0, u)$ for some $u \in \mathcal{T}(\Sigma_c, \mathcal{N} \cup \Sigma_0^+)$; or
2. there is no $i \leq n$ (resp. $j \leq n$) such that $(a_i, \alpha_i) = (b_1, \mathsf{timestamp}(c_1^0))$ (resp. $(a_j, \alpha_j) = (b_1, \mathsf{timestamp}(c_2^0))$); or
3. there exist $i_0, j_0, k_0 \leq n$ and $u \in \mathcal{T}(\Sigma_c, \mathcal{N} \cup \Sigma_0^+)$ such that $\alpha_{k_0} = \mathsf{check}(c_1^0, c_2^0, u)$, $(a_{i_0}, \alpha_{i_0}) = (b_1, \mathsf{timestamp}(c_1^0))$, and $(a_{j_0}, \alpha_{j_0}) = (b_1, \mathsf{timestamp}(c_2^0))$ but there is no $i_0 \leq k' \leq j_0$ such that $a_{k'} = b_2$.

Let $\mathsf{tr}_{n+1} = (b_0, \mathsf{claim}(b_1, b_2, c_1^0, c_2^0), s, t, r)$. We consider each case separately.

Case 1: Let Loc be the mobility plan such that $\mathsf{Loc}(a,t) = (0,0,0)$ for any $a \in \mathcal{A}$ and $t \in \mathbb{R}_+$. Since the location of each agent does not depend on time, for sake of readability, we write $\mathsf{Loc}(a)$ instead of $\mathsf{Loc}(a,t)$.

Let φ be the function such that:

- $\varphi(z_i) = \#\mathsf{TS}(tr_1 \dots tr_{i-1})$ for $i \in \{1, \dots, n+1\}$,
- for all $i \in \mathsf{IN}(tr_1 \dots tr_n)$, we have that:
  - $\varphi(z_i^b) = \varphi(z_{\mathtt{orig}(j)})$ if $b_i \notin \mathcal{M}$ and $R_i = \mathsf{w}_j$,
  - $\varphi(z_i^b) = \varphi(z_i)$ otherwise.

We can show that this function φ satisfies $\mathcal{C}_{\mathsf{exec}}^{\mathsf{Loc}}$. Indeed, we have that:

- $0 \leq \varphi(z_1) \leq \varphi(z_2) \leq \ldots \leq \varphi(z_n)$;
- $\varphi(z_i) < \varphi(z_{i+1})$ for $i \in \mathsf{TS}(\mathsf{tr}_1, \dots, tr_{n-1})$;
- for all $i \in \mathsf{IN}(\mathsf{tr}_1, \dots, \mathsf{tr}_n)$, we have that either $\varphi(z_i^b) = \varphi(z_{\mathtt{orig}(j)}) \leq \varphi(z_i)$ for some j such that $\mathtt{orig}(j) \leq i$, or $\varphi(z_i^b) = \varphi(z_i)$. In both cases, we have that:

$$\varphi(z_i) \geq \varphi(z_i^b) + \mathsf{Dist}(\mathsf{Loc}(a_i), \mathsf{Loc}(b_i))$$

  since $\mathsf{Dist}(\mathsf{Loc}(a_i), \mathsf{Loc}(b_i)) = 0$. Remember that all the agents are at the same location.

- for all $i \in \mathsf{IN}(\mathsf{tr}_1, \dots, \mathsf{tr}_n)$, for all j such that $\mathsf{w}_j \in \mathit{vars}(R_i) \setminus \mathit{dom}(\Phi_0)$, we have that $\mathtt{orig}(j) < i$, and thus $\varphi(z_{\mathtt{orig}(j)}) \leq \varphi(z_i)$. Therefore, we have that:

$$\varphi(z_i^b) \geq \varphi(z_{\mathtt{orig}(j)})$$

  and this allows us to conclude since $\mathsf{Dist}(\mathsf{Loc}(b_i), \mathsf{Loc}(a_{\mathtt{orig}(j)})) = 0$. Remember that all the agents are at the same location.

Finally, applying Lemma 6.1 we obtain that the execution can be lifted into the timed semantics. This timed execution immediately falsifies the DB-secure property because there is no check event in the timed execution. Indeed the timed execution is equal to the untimed one up to the times and the bijective function φ_c defined in Lemma 6.1.

Case 2: since $\mathcal{P}_{\mathsf{db}}$ is well-timed w.r.t. $\mathcal{S}$, this case is not possible.

Case 3: Let Loc be the mobility plan such that everyone but b_2 is located at $(0,0,0)$. Agent b_2 is located at $(1,0,0)$. We formally define Loc as follows:

- $\mathsf{Loc}(a,t) = (0,0,0)$ for any $a \neq b_2$, and any $t \in \mathbb{R}_+$;

- $\mathsf{Loc}(b_2, t) = (1, 0, 0)$ for any $t \in \mathbb{R}_+$.

Since the location of each agent does not depend on time, for sake of readability, we write $\mathsf{Loc}(a)$ instead of $\mathsf{Loc}(a, t)$.

Let φ be the function such that $\varphi(z_i)$ is as follows:

- $2 \cdot \#\mathsf{IN}(\mathsf{tr}_1 \dots \mathsf{tr}_i) + \#\mathsf{TS}(\mathsf{tr}_1 \dots \mathsf{tr}_{i-1})$ for $i < i_0$;
- $\varphi(z_{i_0-1}) + 1 + \frac{1}{\mathsf{c}_0 \cdot n} \#\mathsf{TS}(\mathsf{tr}_{i_0} \dots \mathsf{tr}_{i-1})$ for $i_0 \leq i \leq j_0$;
- $\varphi(z_{j_0}) + 2 \cdot \#\mathsf{IN}(\mathsf{tr}_{j_0+1} \dots \mathsf{tr}_i) + \#\mathsf{TS}(\mathsf{tr}_{j_0+1} \dots \mathsf{tr}_{i-1})$ for $1 > j_0$.

Informally, for all actions outside the rapid phase delimited by indices i_0 and j_0, we apply a delay of 2 before each input, and a delay of 1 after each timestamp. This allows any agent to receive any message forgeable from the frame at each step of the execution. During the critical phase we do not apply delay before inputs and only apply a short delay of $1/(\mathsf{c}_0 \cdot n)$ after each timestamp to ensure that time increases as required by the semantics. The delay introduced between the two `gettime` (or timestamp) commands will thus be small.

In addition, for all $i \in \mathsf{IN}(\mathsf{tr}_1 \dots \mathsf{tr}_n)$, if $b_i \notin \mathcal{M}$ then $\varphi(z_i^b) = \varphi(z_{\mathtt{orig}(j)})$ where $R_i = \mathsf{w}_j$, otherwise:

$$\varphi(z_i^b) = \begin{cases} \varphi(z_i) - 1 & \text{if } i < i_0 \text{ or } i > j_0 \\ \varphi(z_i) & \text{if } i_0 \leq i \leq j_0. \end{cases}$$

We can show that this function φ satisfies $\mathcal{C}_{\mathsf{exec}}^{\mathsf{Loc}}$. Indeed, we have that:

- $0 \leq \varphi(z_1) \leq \varphi(z_2) \leq \ldots \leq \varphi(z_n)$;
- $\varphi(z_i) < \varphi(z_{i+1})$ for $i \in \mathsf{TS}(\mathsf{tr}_1, \ldots, tr_{n-1})$;
- Regarding the remaining constraints in $\mathcal{C}_{\mathsf{exec}}^{\mathsf{Loc}}$, we consider $i \in \mathsf{IN}(\mathsf{tr}_1, \ldots, \mathsf{tr}_n)$, and we distinguish two sub-cases:
  Case $i < i_0$ or $j_0 < i$: if the agent b_i forging the received message is honest then we have that $r_i = \mathsf{w}_j$ for some j such that $\mathtt{orig}(j) < i$ and $a_{\mathtt{orig}(j)} = b_i$. Therefore, we have that:

$$\begin{array}{rcl} \varphi(z_i) & \geq & \varphi(z_{i-1}) + 2 \\ & \geq & \varphi(z_{\mathtt{orig}(j)}) + 2 \\ & = & \varphi(z_i^b) + 2 \\ & \geq & \varphi(z_i^b) + \mathsf{Dist}(\mathsf{Loc}(a_{\mathtt{orig}(j)}), \mathsf{Loc}(b_i)) \end{array}$$

  Moreover, we have $\varphi(z_i^b) = \varphi(z_{\mathtt{orig}(j)}) + \mathsf{Dist}(\mathsf{Loc}(a_j), \mathsf{Loc}(b_i))$ since $a_{\mathtt{orig}(j)} = b_i$.

  Otherwise, we have that $b_i \in \mathcal{M}$, and we have that $\varphi(z_i) = \varphi(z_i^b) + 1$ by definition of φ, and since $\mathsf{Dist}(\mathsf{Loc}(b_i), \mathsf{Loc}(a_i)) \leq 1$, this allows us to conclude that:

$$\varphi(z_i) \geq \varphi(z_i^b) + \mathsf{Dist}(\mathsf{Loc}(b_i), \mathsf{Loc}(a_i)).$$

Now, let $\mathsf{w}_j \in vars(R_i)$. We have that:

$$\begin{array}{rcl} \varphi(z_i^b) & \geq & \varphi(z_i) - 1 \\ & \geq & \varphi(z_{i-1}) + 2 - 1 \\ & \geq & \varphi(z_{\mathtt{orig}(j)}) + 1 \\ & \geq & \varphi(z_{\mathtt{orig}(j)}) + \mathsf{Dist}(\mathsf{Loc}(b_i), \mathsf{Loc}(a_{\mathtt{orig(j)}})). \end{array}$$

Case $i_0 \leq i \leq j_0$: if the agent b_i forging the received message is b_2 (b_2 is honest) then we have that $r_i = \mathsf{w}_j$ for some j such that $\mathtt{orig}(j) < i_0$ (remember that no action of b_2 occurs between i_0 and j_0). Therefore, we have that:

$$\begin{array}{rcl} \varphi(z_i) & \geq & \varphi(z_{i_0-1}) + 1 \\ & \geq & \varphi(z_{\mathtt{orig}(j)}) + 1 \\ & = & \varphi(z_i^b) + \mathsf{Dist}(\mathsf{Loc}(b_i), \mathsf{Loc}(a_i)). \end{array}$$

By definition of φ, we have that $\varphi(z_i^b) \geq \varphi(z_{\mathtt{orig}(j)})$, and thus

$$\varphi(z_i^b) \geq \varphi(z_{\mathtt{orig}(j)}) + \mathsf{Dist}(\mathsf{Loc}(b_i), \mathsf{Loc}(a_{\mathtt{orig}(j)}))$$

since $\mathsf{Dist}(\mathsf{Loc}(b_i), \mathsf{Loc}(a_{\mathtt{orig}(j)})) = 0$ (we have that $b_i = a_{\mathtt{orig}(j)}$.)

Otherwise, if b_i is honest but different from b_2, then we have that:

$$\begin{array}{rcl} \varphi(z_i) & \geq & \varphi(z_{i-1}) \\ & \geq & \varphi(z_{\mathtt{orig}(j)}) \\ & = & \varphi(z_i^b) + 0 \\ & = & \varphi(z_i^b) + \mathsf{Dist}(\mathsf{Loc}(b_i), \mathsf{Loc}(a_i)). \end{array}$$

By definition of φ, we have that $\varphi(z_i^b) = \varphi(z_{\mathtt{orig}(j)})$, and thus

$$\varphi(z_i^b) \geq \varphi(z_{\mathtt{orig}(j)}) + \mathsf{Dist}(\mathsf{Loc}(b_i), \mathsf{Loc}(a_{\mathtt{orig}(j)}))$$

since $\mathsf{Dist}(\mathsf{Loc}(b_i), \mathsf{Loc}(a_{\mathtt{orig}(j)})) = 0$ (we have that $b_i = a_{\mathtt{orig}(j)}$.)

Now, if $b_i \in \mathcal{M}$, we have that $\mathsf{Loc}(b_i) = \mathsf{Loc}(a_i)$ (since b_2 is the only agent not located at the same place of the others, $b_2 \notin \mathcal{M}$ and we know that there is no occurrence of an action performed by b_2 between i_0 and j_0, thus $a_i \neq b_2$). We have that $\varphi(z_i) = \varphi(z_i^b)$ by definition of φ, and since $\mathsf{Dist}(\mathsf{Loc}(b_i), \mathsf{Loc}(a_i)) = 0$, this allows us to conclude that:

$$\varphi(z_i) \geq \varphi(z_i^b) + \mathsf{Dist}(\mathsf{Loc}(b_i), \mathsf{Loc}(a_i)).$$

Now, let $\mathsf{w}_j \in vars(R_i)$. In case $a_{\mathtt{orig(j)}} \neq b_2$, we have that the distance between b_i and $a_{\mathtt{orig(j)}}$ is equal to 0 and thus:

$$\varphi(z_i^b) \geq \varphi(z_{\mathtt{orig}(j)}) + \mathsf{Dist}(\mathsf{Loc}(b_i), \mathsf{Loc}(a_{\mathtt{orig}(j)}))$$

In case $a_{\mathtt{orig(j)}} = b_2$, we know that $\mathtt{orig}(j) < i_0$, and thus

$$\varphi(z_{\mathtt{orig}(j)}) \leq \varphi(z_{i_0}) - 1 \leq \varphi(z_i) - 1$$

Thus, we have that:

$$\begin{array}{rcl} \varphi(z_i^b) & = & \varphi(z_i) \\ & \geq & \varphi(z_{\mathtt{orig}(j)}) + 1 \\ & \geq & \varphi(z_{\mathtt{orig}(j)}) + \mathsf{Dist}(\mathsf{Loc}(b_i), \mathsf{Loc}(a_{\mathtt{orig(j)}})) \end{array}$$

Now, we have shown that φ satisfies $\mathcal{C}^{\mathsf{Loc}}_{\mathsf{exec}}$, we can apply Lemma 6.1 to re-time the execution. We obtain that:

$$\mathcal{K}_0 \xrightarrow{\mathsf{tr}'_1 \ldots \mathsf{tr}'_n.(b_0, \mathsf{claim}(b_1, b_2, c_1^0\varphi_c, c_2^0, \varphi_c), s, \emptyset)}_{\mathsf{Loc}} \mathcal{K}'_{n+1}$$

with:

$$\mathsf{tr}'_i = \begin{cases} (a_i, \mathtt{gettime}, s_i, \varphi(z_i), \emptyset) & \text{if } \alpha_i = \mathsf{timestamp}(c_i) \\ (a_i, \alpha_i\varphi_c, s_i, \varphi(z_i), r_i\varphi_c) & \text{otherwise} \end{cases}$$

and $\varphi_c(c_i) = \varphi(z_i)$ for all $i \in \mathsf{TS}(tr_1 \ldots tr_n)$.

By construction, we have that $\varphi_c(c_1^0) = \varphi(z_{i_0})$, $\varphi_c(c_2^0) = \varphi(z_{j_0})$, and, by definition of φ, we have that $\varphi(z_{i+1}) - \varphi(z_i) \leq 1/(\mathsf{c_0} \cdot n)$ when $i \in \{i_0, \ldots, j_0 - 1\}$. Therefore, we have that:

$$\begin{array}{rcl} \varphi(z_{j_0}) - \varphi(z_{i_0}) & \leq & (j_0 - i_0 + 1)/(\mathsf{c_0} \cdot n) \\ & \leq & 1/\mathsf{c_0} \\ & < & 2/\mathsf{c_0} \\ & \leq & 2 \times \mathsf{Dist}(\mathsf{Loc}(b_1), \mathsf{Loc}(b_2)) \end{array}$$

Hence, we have that $\mathcal{P}_{\mathsf{db}}$ is not DB-secure.

□

This proves Proposition 6.1 and thus the first direction of Theorem 6.1. In the next section we are going to focus on the proof of the other direction.

6.3.3. From causality-based security to distance-bounding security

This section is devoted to the proof of the following proposition.

Proposition 6.2. *Let $\mathcal{P}_{\mathsf{db}}$ be protocol and $\mathcal{S}$ a set of valid initial configurations. If $\mathcal{P}_{\mathsf{db}}$ is causality-based secure w.r.t. $\mathcal{S}$ then $\mathcal{P}_{\mathsf{db}}$ is DB-secure w.r.t. $\mathcal{S}$.*

This implication is more complex than the previous one. Even if the full proof is presented in Appendix C, we provide a flavour of it and recall the main intermediate lemmas.

The proof starts with an attack trace w.r.t. DB-security, i.e. such that $t_2^0 - t_1^0 < \mathsf{Dist}(\mathsf{Loc}(b_1, t_1^0), \mathsf{Loc}(b_2, t)) + \mathsf{Dist}(\mathsf{Loc}(b_2, t), \mathsf{Loc}(b_1, t_2^0))$ for any $t_1^0 \leq t \leq t_2^0$ and then follows the following steps:

1. We first weaken the trace in the untimed semantics (see Lemma 6.2).
2. Then, we clean up the trace between the two timestamp actions (see Proposition 6.3). Intuitively, we remove all the actions that do no depend on the first timestamp and on which the second timestamp does not depend on. This transformation is similar to the reordering performed in Section 4.2.2 (Chapter 4) when reducing the topologies for distance hijacking attacks.
3. We apply Lemma 6.1 to lift this execution in the timed model keeping the value $t_2^0 - t_1^0$ unchanged.
4. Assuming that the protocol is causality-based secure, there is still an action executed by the prover between the two timestamps. By construction this action depends on the first timestamp and the second timestamp depends on this action. We can thus deduce a timing constraint which contradicts the fact that the initial trace was a witness of attacks w.r.t. DB-security (see Lemma 6.3). Intuitively, if an action depends on another, then enough time must have elapsed to let a message travel between the two agents executing the actions.

The first step of the proof consists in weakening a timed trace into the untimed semantics. This is permitted thanks to Lemma 6.2. Even if this transformation can be done in a rather straightforward way, we state it with some details in order to maintain a strong link between the two executions.

To do so, we first precise the transformation $\overline{\cdot}$ presented in Section 6.3.1. When we apply this transformation, we rely on the function $\sigma_{\mathsf{spe}} : \mathcal{X} \to \Sigma_0^{\mathsf{spe}}$ which is used to replace an action of the form $\mathtt{gettime}(x)$ by the action $\mathsf{timestamp}(\sigma_{\mathsf{spe}}(x))$.

Similarly, given an execution $\mathsf{exec} = \mathcal{K}_0 \xrightarrow{tr}_{\mathsf{Loc}} \mathcal{K}_n$, we denote $\sigma_{\mathsf{time}} : \mathcal{X} \to \mathbb{R}_+$ the function that associates to each variable occurring in a `gettime` instruction, the current time at which this instruction has been executed.

Lemma 6.2. *Let $\mathcal{P}_{\mathsf{db}}$ be a protocol and $\mathcal{K}_0$ be a valid initial configuration for $\mathcal{P}_{\mathsf{db}}$. For any execution*

$$\mathsf{exec} = \mathcal{K}_0 \xrightarrow{tr_1 \ldots tr_n}_{Loc} \mathcal{K}_n$$

such that $tr_i = (a_i, \alpha_i, s_i, t_i, r_i)$ for $i \in \{1, \ldots, n\}$, we have that $\overline{\mathcal{K}_0} \overset{tr'_1 \ldots tr'_n}{\rightsquigarrow} \mathcal{K}'_n$ where $\mathcal{K}'_n = \overline{\mathcal{K}_n}\sigma$ and for any $i \in \{1, \ldots, n\}$, we have that:

$$tr'_i = \begin{cases} (a_i, \mathsf{timestamp}(t_i\sigma), s_i, \emptyset) & \text{if } \alpha_i = \mathtt{gettime} \\ (a_i, \alpha_i\sigma, s_i, (b_i, R_i\sigma)) & \text{if } r_i = (b_i, t_i^b, R_i) \\ (a_i, \alpha_i\sigma, s_i, r_i\sigma) & \text{otherwise} \end{cases}$$

where $\sigma = \sigma_{\mathsf{spe}} \circ \sigma_{\mathsf{time}}^{-1}$ assuming that σ_{spe} is the function used to transform $\mathcal{K}_0$ into $\overline{\mathcal{K}_0}$ and σ_{time} is the one associated to the execution exec.

Proof. This proof is immediate because the configurations only differ from the bijective function σ (the equalities are thus preserved) and the rules in the untimed semantics are less restrictive than the rules in the timed semantics □

The second step of the proof cleans up the trace between the two timestamps so that only actions that depend on the first timestamp event and on which the last timestamp event depends on remain. This cleaning is very similar to the one done in Section 4.2.2 (Chapter 4) when reducing the topologies for distance hijacking attacks. It relies on a notion of data/sequence dependency between actions. Since our models are similar, the proofs can be easily adapted. Hence, we only recall some of the key points. Remember the notation $\mathsf{tr}_2 \hookrightarrow^* \mathsf{tr}_1$ when the action tr_2 depends on tr_1, and $\mathsf{tr}_2 \not\hookrightarrow^* \mathsf{tr}_1$ otherwise.

Proposition 6.3. *Let $\mathcal{T}$ be a topology, and $\mathcal{K}_0 \overset{\mathsf{tr}_1 \ldots \mathsf{tr}_n}{\rightsquigarrow}_{\mathcal{T}} \mathcal{K}_n$ be an execution with $n \geq 2$. We have that there exists a bijection $\varphi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ such that:*

- *$\mathcal{K}_0 \overset{\mathsf{tr}'_1 \ldots \mathsf{tr}'_n}{\rightsquigarrow}_{\mathcal{T}} \mathcal{K}_n$ with $\mathsf{tr}_i = \mathsf{tr}'_{\varphi(i)}$ for all $i \in \{1, \ldots, n\}$; and*
- *for all j such that $\varphi(1) < j < \varphi(n)$, we have that $\mathsf{tr}'_{\varphi(n)} \hookrightarrow^* \mathsf{tr}'_j \hookrightarrow^* \mathsf{tr}'_{\varphi(1)}$.*

Moreover, for all j_1, j_2 such that $\varphi(1) \leq j_1 < j_2 \leq \varphi(n)$, we have that $\varphi^{-1}(j_1) < \varphi^{-1}(j_2)$.

6

Proof. This proposition is almost the same proposition as Proposition 4.2 that is stated and proved in Chapter 4. In this lemma we just state the new property: for all j such that $\varphi(1) < j < \varphi(n)$, we have that $\mathsf{tr}'_{\varphi(n)} \hookrightarrow^* \mathsf{tr}'_j \hookrightarrow^* \mathsf{tr}'_{\varphi(1)}$.

Thanks to the close link between our two untimed semantics (they mainly differ from the agent mobility and the $\mathsf{timestamp}(c_x)$ commands which are simple events), the same proof applies and we can easily establish this property. □

The third step of the proof lifts the trace in the timed semantics thanks to Lemma 6.1. Finally, in step 4 we establish a contradiction assuming that the protocol is causality-based secure. Indeed, regarding this assumption, there exists an action executed by agent b_2 between the two timestamp (or `gettime`) commands. Therefore, this action depends on the first timestamp, and the last timestamp depends on this action. A contradiction can thus be derived from the following lemma which provides a timing constraint that must be satisfied by any dependent actions.

Lemma 6.3. *Let Loc be a mobility plan, and $\mathsf{exec} = \mathcal{K}_0 \xrightarrow{\mathsf{tr}_1 \ldots\ldots \mathsf{tr}_n}_{\mathsf{Loc}} \mathcal{K}_1$ be an execution with $\mathsf{tr}_i = (a_i, \alpha_i, s_i, t_i, r_i)$ for $i \in \{1, \ldots, n\}$. Let $i, j \in \{1, \ldots, n\}$ such that $\mathsf{tr}_j \hookrightarrow^* \mathsf{tr}_i$. We have that:*

$$t_j \geq t_i + \mathsf{Dist}(\mathsf{Loc}(a_i, t_i), \mathsf{Loc}(a_j, t_j)).$$

Proof. A proof similar of Lemma 4.5 can be done here. Indeed, the two lemmas are the same, up to the mobility here which requires to precise the locations with the corresponding time. □

Proof of Proposition 6.2

The full proof of Proposition 6.2 is presented in Appendix C. Here we give a flavour of it.

Proof. (Sketch of proof)

Assuming that $\mathcal{P}_{\mathsf{db}}$ is not DB-secure, there exists an execution exec such that:

$$\overline{\mathcal{K}_0} \xrightarrow{(a_1,\alpha_1,s_1,r_1)\ldots(a_n,\alpha_n,s_n,r_n)\cdot(b_0,\mathsf{claim}(b_1,b_2,c_1,c_2),s,r)} (\mathcal{P}';\phi')$$

with $b_1 \in \mathcal{M}$, and $b_2 \in \mathcal{M}$, and one of the two following cases applies:

1. there is no index $k \leq n$ such that $\mathsf{tr}_k = (a_k, \mathsf{check}(t_1^0, t_2^0, t_3^0), s_k, t_k, \emptyset)$; or
2. for any t with $t_1^0 \leq t \leq t_2^0$, we have that:

$$t_2^0 - t_1^0 < \mathsf{Dist}(\mathsf{Loc}(b_1, t_1^0), \mathsf{Loc}(b_2, t)) + \mathsf{Dist}(\mathsf{Loc}(b_2, t), \mathsf{Loc}(b_1, t_2^0)).$$

Case 1: we apply Lemma 6.2 to weaken the given execution in the untimed semantics. Since this transformation only applies a bijective renaming to times, we obtain that the untimed trace does not contain a check which corresponds to the claim and thus the causality-based security is immediately falsified.

Case 2: as previously, we first apply Lemma 6.2 to weaken the given execution in the untimed semantics. Then, applying Proposition 6.3, we transform the trace so that any action executed between the two timestamps depend on them. Assuming that the two timestamps are identified by tr'_{i_0} and tr'_{j_0} in the resulting trace $\mathsf{tr}'_1 \ldots \mathsf{tr}'_n$, we have that for any $k \in \{i_0, \ldots, j_0\}$:

$$\mathsf{tr}'_{j_0} \hookrightarrow^* \mathsf{tr}'_k \hookrightarrow^* \mathsf{tr}'_{i_0}.$$

Assuming that $\mathcal{P}_{\mathsf{db}}$ is causality-based secure, we know that there is such an action executed by agent b_2, i.e. there exists $k_0 \in \{i_0, \ldots, j_0\}$ such that $\mathsf{tr}'_{k_0} = (b_2, \alpha_{k_0})$.

The last step of the proof consists in re-timing the trace. Let us distinguish three parts in the trace: before the first timestamps, after the second timestamps and between the two. To re-time before and after the timestamps, we simply let enough time elapse between two actions to ensure that all the messages can be received on time. The critical part is located between the two timestamps. To re-time this part of the execution, we apply the same delays as in the original execution exec between each action. Note that, since did not introduce new actions between the timestamps (we only removed ones), the timing constraints will be satisfied.

In addition, following the original execution to re-time the trace gives us a strong link between the times when the `gettime` commands are executed: the delay between the two timestamps remains unchanged.

Finally, in order to reach a contradiction, we apply Lemma 6.3 to $\mathsf{tr}'_{j_0} \hookrightarrow^* \mathsf{tr}'_k$ and $\mathsf{tr}'_k \hookrightarrow^* \mathsf{tr}'_{i_0}$ and obtain that:

$$t_2^0 - t_1^0 \geq \mathsf{Dist}(\mathsf{Loc}(b_1, t_1^0), \mathsf{Loc}(b_2, t_k)) + \mathsf{Dist}(\mathsf{Loc}(b_2, t_k), \mathsf{Loc}(b_1, t_2^0)).$$

where t_k is the time when the action tr'_{k_0} is executed. Contradiction. □

6.4. Case studies

Theorem 6.1, proved in the previous section, allows us to use existing tools to analyse the security of the two protocols PayBCR and PayCCR w.r.t. the DB-security property. Indeed, the causality-based security property has been proved equivalent to DB-secure and solely relies on the order of the actions in untimed traces. We decided to use the ProVerif tool [Bla01] to verify this correspondence property. All the Proverif models mentioned in this section are available at: `https://gitlab.inria.fr/adebant/verif-db/`.

6.4.1. ProVerif models and scenarios under study

We model the PayBCR protocol following the description given in Figure 6.1. Regarding PayCCR, presented in Figure 6.4, it is actually not possible to state the causality-based security property (and even more the DB-security property). The problem is that, in PayCCR, the bank never receives the reader/TPM identity nor the two timestamps. The proximity re-check is performed by the card which is assumed honest. The bank can thus trust the card and does not need to receive this information. Unfortunately, these last are needed to state the final claim. To overcome this limitation, we propose a slightly modified version of the protocol named PayCCR++ in which the TPM identity and the two timestamps are added in the AC message[1]. More formally, we have that:

$$AC_{\mathsf{PayCCR++}} = \mathtt{mac}(K_S, ATC, data, \sigma_1, \mathsf{tpm}, t_1, t_2)$$

with $K_S = \mathsf{senc}(ATC, \mathtt{shk}(\mathsf{card}, \mathsf{bk}))$ as presented in Figure 6.4.

The causality-based security property allows to analyse protocols w.r.t. an arbitrary set of valid initial configurations. We decided to consider scenarios with an arbitrary number of banks, cards, and TPMs. We do not model the readers which is assumed to be dishonest, and thus fully executable by the underlying attacker of the Proverif tool. More precisely, the role of the bank is played by many possible entities and each bank issues many cards and many TPMs. Among these cards and these TPMs, some are honest, and some are dishonest meaning that their credentials are revealed to the attacker. As usually assumed in payment protocols, we do not consider scenarios in which an honest entity acts as a card and a TPM as the same time. In order get close the the reality and thus prevent a dishonest participant from acting as a card and a TPM at the same time, it is important to differentiate the two corresponding certificates: given a bank name bk, a TPM certificate of agent a, noted $certT(a)$, is

$$\mathsf{sign}(\langle \mathsf{TPMCert}, a, \mathsf{spk}(a)\rangle, \mathsf{ssk}(\mathsf{bk}))$$

whereas a card certificate, $certC(a)$ will be

$$\mathsf{sign}(\langle \mathsf{cardCert}, a, \mathsf{spk}(a)\rangle, \mathsf{ssk}(\mathsf{bk})).$$

[1] In practice, this is feasible via optional fields inside the AC, which issuing-banks already use for further data-collection.

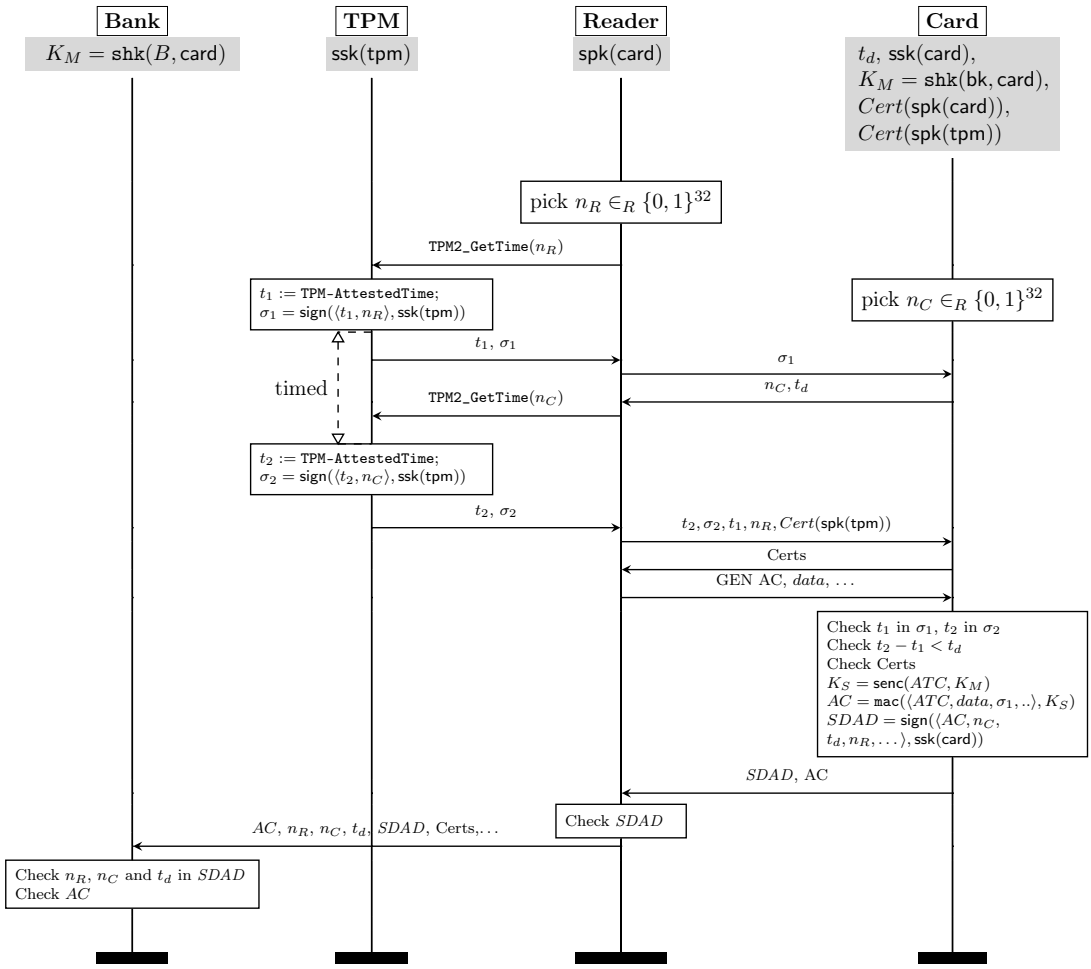


Figure 6.4: PayCCR [CBC19]: description of the protocol.

During our analyses, we considered initial frames that contain the initial knowledge of dishonest agents composed of:

- $\mathsf{spk}(a)$ for any agent a, and his associated certificate $certX(a)$;
- $\mathsf{ssk}(a)$ and $\mathtt{shk}(a, \mathsf{bk})$ when the agent a is dishonest.

6.4.2. Two extra authentication properties

Besides the causality-based security property we introduced in Definition 6.5 to verify whether a protocol ensures the physical proximity of the agents, we will analyse the protocols w.r.t. two extra authentication properties.

First, we consider a property that expresses the fact that, when the bank ends a session apparently with two agents tpm_0 and card_0, then tpm_0 is a TPM identity whereas card_0 is a card identity. The property relies on two events $\texttt{TPM}(x)$ and $\texttt{CARD}(x)$ that are triggered when an identity is created to model that x is either a TPM identity or a card identity. The security property is then as follows:

```
query tpm0:bitstring, card0:bitstring, t1:bistring, t2:bitstring;
    event(claim(tpm0,card0,t1,t2)) ==> (event(TPM(tpm0)) && CARD(card0)).
```

The second authentication property we consider deals with the time-bound t_d against which the times are checked to define the reader/TPM's proximity. This bound is specific to each card and sent by the card in the SDAD message. In this context, an attacker may try to replace it by an excessively large value to make the bank accept the transaction. To avoid this undesired behaviour, we model this bound by an uninterpreted symbol of function $\texttt{timebound}(x)$, where x is an agent name, and we add at the end of the bank process a new event $\texttt{receivedBound}(\mathsf{card}_0, \mathsf{t_d})$ and check the following extra property for any honest agent card_0:

```
query card0:bitstring, timeboundinfo:bitstring;
    event(receivedBound(card0, timeboundinfo))
        ==> timeboundinfo = timebound(card0).
```

6.4.3. Verification results

The protocols have been analysed w.r.t. the causality-based security property and the two authentication properties mentioned above. To make Proverif conclude, we added additional data in the check and timestamp events (e.g., the fresh nonce n_C generated by the card during a session). In addition, to highlight the actions performed by the card (i.e. b_2 in Definition 6.5), we added a new event `proverAction` in the role of the card. One may note that if a protocol satisfies the resulting query then it is causality-based secure. Indeed, the more precise the events are the stronger the security property is. Similarly, since only a unique action of agent b_2 is made visible through the `proverAction` event, the security property is even more difficult to satisfy. The query we are considering is:

```
query bankID:bitstring, t1:bitstring, nR:bitstring, t2:bitstring,
      nC:bitstring, timeboundinfo:bitstring;
  event(claim(tpm0, card0, t1, t2))
     ==> (event(checkTimes(t1, t2, timeboundinfo, nC, bankID))
       ==> (event(timeStamp(tpm0,t2, nC))
         ==> (event(proverAction(card0,sign((t1,nR), secKey(tpm0))))
           ==> (event(timeStamp(tpm0,t1, nR))))).
```

where `tpm0` and `card0` ar two honest identities. It relies on nested correspondences to model sequential relations between actions.

Proverif always returns in less than 1s on a standard laptop. All the results are presented in Table 6.1. As expected the two protocols are causality-based secure, i.e., the physical proximity of the two agents involved in a transaction is ensured, as soon as the

TPM is not corrupted and the card is honest. Moreover, note that, for such transactions, if time-bound authentication holds, then this inherently implies that the check of timestamps occurring in the causality-based property had been correctly performed.

However, if the PayBCR protocol satisfies role authentication, PayCCR++ does not. This means that, in PayCCR++, a bank may accept a rogue transaction, for instance one involving a card acting as a TPM (or inversely). This weakness is however not surprising. Indeed, checking the TPM certificate is part of the card's role in PayCCR++, while it is performed by the bank in PayBCR. This means that certificates may not have been checked if the card's role is executed by a malicious agent in PayCCR++, whereas in PayBCR they will be always correctly checked since the bank is assumed honest.

Protocol	Role authentication	Time-bound authentication	Causality-based security
PayCCR++	×	✓	✓
PayBCR	✓	✓	✓

Table 6.1: Results of the security analyses of PayBCR and PayCCR++.

6.5. Conclusion

In this chapter, we managed to model and prove the security of the two novel EMV-payment protocols PayBCR and PayCCR. This required first to extend an existing model and an existing security property to handle TPMs and rogue readers. In addition, we proposed a modelling for agent mobility to faithfully model real-world applications. Finally, we have proved an equivalence between our new security property, called DB-secure, and a causality-based property that can be checked by existing tools. The DB-secure and causality-based secure properties draw their inspiration from the original framework proposed by Mauw *et al.* [MSTPTR18].

Regarding the two protocols that have been analysed, one may note that the DB-security/causality-based security definitions do not apply for analysing the PayCCR protocol. Indeed, these are quite strong properties which requires to link the card's ID and the TPM's ID together with the specific timestamps, inside the claim event. Another approach would be to link the claim to the check event through a session identifier (e.g. n_C or n_R). Following this approach, the claim event would thus be $\mathsf{claim}(b_1, b_2, s_{id})$ and the check event $\mathsf{check}(s_{id}, t_1^0, t_2^0, t_3^0)$. Such an approach would be necessary to model protocols in which the card and the reader do not want to reveal any timing information to the bank: such a leakage may appear as a privacy issue. Unfortunately, following this new approach, our proof technique does not apply anymore in case several check events happen for a given session. It would thus be interesting to see how to overcome such a limitation.

7 Conclusion

This manuscript has been dedicated to the symbolic verification of distance-bounding protocols with a particular interest in contactless payment protocols. This work has proceeded in two steps: first we have addressed modelling issues to formally describe the distance-bounding protocols and the security properties that are relevant. Then, we have proposed procedures and theoretical results that allow an automatic verification.

7.1. On modelling of distance-bounding protocols

To address the analysis of these protocols we have first proposed a new symbolic model which faithfully models physical constraints, i.e. time and locations. The main novelty is the modelling of the transmission delay from one location to another. Unlike usual symbolic models, messages cannot be instantaneously sent and received over the network, enough time must have elapsed to let the message reach its destination. In addition, we have also introduced the notion of mobility, i.e. agents can move during a session of the protocol, in order to faithfully model real-world applications.

Once the model has been defined, we have proposed formal definitions of the three classes of attacks against which distance-bounding are usually analysed. In particular, our definitions encompass the infinite number of topologies/configurations that may happen in practice. A specific interest has been paid on the definition of terrorist fraud for which modelling the collusion behaviour was a challenging problem.

Open problems and future work

In this manuscript (but also in the literature) two approaches have been proposed to model distance-bounding protocols: the first one, presented in Chapter 2 and developed across Part I, relies on blocking commands, $\mathtt{if}_{\mathtt{time}}$, to represent timing constraints. It faithfully models the actual behaviour of a user who will immediately stop the protocol as soon as a constraint is not satisfied. The second approach is less restrictive about the semantics, and abstracting the timing constraints by simple events, and encodes the meaning of a constraint in the security property under study. This approach appears to be more convenient for verification.

Precisely comparing the two approaches to understand in which context they are equivalent would thus be interesting. Moreover it might allow to unify the two approaches which would be an elegant contribution.

Another interesting contribution would be to improve the modelling of the rapid phase in distance-bounding protocols. For efficiency purposes, the rapid phase often relies on bit-level operations, and the modelling of bit-messages is a well-known open problem in symbolic models. Indeed, to avoid unrealistic attacks due to the too powerful Dolev-Yao attacker model, symbolic models completely abstract such operations through names and symbols of functions. This modelling this may thus miss attacks. Providing a framework that faithfully models bit messages would thus be of great interest for distance-bounding protocols.

Recently, Chadha *et al.* [CSV17] proposed a symbolic model that builds on probabilistic commands (e.g. outputs), and could thus model bit-level operations avoiding the unrealistic attacks. However, even if this is a promising model, it also opens challenging problems. Indeed, the rapid phase, which relies on bit-level operations, is not isolated from the rest of the protocol. i.e. it may share data, keys... How to model the interactions between bit-messages and the standard term algebra remains unclear.

7.2. On verification of distance-bounding protocols

Once these models and security properties were properly defined, we proposed in this manuscript three approaches to verify them. As a first approach we extended the underlying procedure of the Akiss tool to take these new features, i.e. time and locations, into account, and be able to analyse protocols considering a bounded number of sessions. We have managed to prove the correctness (soundness and completeness) of the new procedure. We have implemented it and have managed to analyse some distance-bounding protocols.

Due to the inherent weakness of the first approach, which only applies for a bounded number of sessions, we have then proposed reduction results to make easier the verification. We have reduced the number of relevant topologies to only two, one for mafia and terrorist fraud, and one for distance hijacking. In addition, we have proved that, under few assumptions, there exists a best collusion strategy for the semi-dishonest prover, on which the analyses can focus on. Thanks to these reductions results we have been able to leverage existing tools, like Proverif or Tamarin, to perform a large case studies analysis considering an unbounded number of sessions. It required to encode the reduced topologies and it has been performed relying on the Proverif tool and its embedded notion of phases. Up to our knowledge this is the most advanced framework that allows for distance-bounding protocols analyses: it handles the unboundedness of topologies and collusion behaviours to provide a fully automated verification.

Finally, we have proved the security of two novel EMV-payment protocols that prevent relay attacks, even considering malicious readers/verifiers. Indeed, a reader may have incentives to by-pass the proximity check and accept any transaction. In order to perform this analysis w.r.t. to this strong threat model, we have extended the security properties and the theoretical result presented in [MSTPTR18]. The causality-based security property has then been checked relying on the Proverif tool.

Open problems and future work

Despite the pretty good results obtained when analysing the numerous distance-bounding protocols, each approach suffers from limitations. Each of them are precisely discussed at the end of the chapters across the manuscript. Here, we prefer to discuss two general open problems about symbolic verification of distance-bounding protocols.

First, none of the existing frameworks allow to faithfully model the exclusive-OR operator, even though it is almost always used during the rapid phase of a distance-bounding protocol. Neither our new procedure implemented on top of Akiss, nor Proverif's approaches, nor the Tamarin's ones proposed by Mauw *et al.*, allow to faithfully model this operator when verifying protocols. Regarding our new procedure (based on the Akiss tool), the limitation comes from the model itself. The Akiss procedure has recently been extended [BDGK17] to handle the exclusive-OR, and we hope that our procedure could be extended too. Nevertheless, we expect that it will strongly impact the efficiency of the procedure. Regarding the Proverif's approaches, i.e. ours and the one developed by Chothia *et al.*, the two models, and our reduction results, are generic enough to handle such an operator. The limitation comes from the tool, i.e. Proverif. Indeed, even if it has been extended to handle the exclusive-OR [KT11] operator, the new procedure has not proved its efficiency in practice. Finally, the Tamarin's framework [MSTPTR18, MSTPTR19] is certainly the most advanced framework to precisely model the exclusive-OR operator in distance-bounding protocols. However, this does not seem to be the panacea neither because of the Tamarin tool. Indeed, even if the Tamarin tool officially handles the exclusive-OR operator, it lacks efficiency when analysing protocols involving this operator. For example, we tried to perform our case studies relying on this tool but we faced up many non-termination issues. The same behaviours seem to have appeared in [MSTPTR18, MSTPTR19] since some protocol models implement a weak exclusive-OR symbol to make the analyses possible (as in our Proverif framework). Much work remains to do be done for modelling the exclusive-OR in automatic symbolic verification tools.

Finally, assuming that the open problems related to the modelling of protocols in a probabilistic model have been solved, the automatic verification in such models is another challenging problem. For example, Chadha *et al.*'s model [CSV17] completely misses automation. Again, we do think that existing procedures (especially for a bounded number of sessions) could be extended but it would strongly impact the efficiency. Proposing an efficient procedure for such models would be a giant leap for symbolic verification.

Proofs of Chapter 3 A

In this chapter we provide the omitted proofs in Chapter 3.

Completeness of the saturation step

Lemma 3.9. *Let* $\mathsf{exec} = (T_0; \emptyset; t_0) \xrightarrow{\ell_1,\ldots,\ell_p}_{\mathcal{T}} (S; \Phi; t)$ *be an execution with input recipes* $R_1, \ldots, R_k$ *forged by* $b_1, \ldots, b_k$ *and such that each* R_j *with* $j \in \{1, \ldots, k\}$ *is uniform w.r.t.* Φ*. Let* $K = \mathsf{sat}(K_{\mathsf{init}}(T_0))$*, and* $g = (H \Leftarrow B_1, \ldots, B_n) \in K$ *be such that* u_0*, the underlying world of* H*, is locally closed. Let* σ *be a grounding substitution for* g *such that* $\mathsf{skl}(g\sigma)$ *is in normal form, and* g *and* σ *match with* exec *and* $R_1, \ldots, R_k$*.*

Moreover, in case H *is of the form* $H = \mathsf{k}_{u_0}(R_H, t_H)$*, we assume that* $u_0\sigma = \ell_1, \ldots, \ell_{q-1}$ *for some* $q \in \mathsf{Rcv}(p)$ *and* $R_H\sigma$ *is asap w.r.t.* $b_{|\mathsf{Rcv}(q)|}$ *and* exec*.*

Assuming that $B_i\sigma \in \mathcal{H}(\mathsf{solved}(K))$ *with a proof tree* π_i *matching with* exec *and* $R_1, \ldots, R_k$ *for each* $i \in \{1, \ldots, n\}$*, and* $\{\pi_1, \ldots, \pi_n\}$ *is uniform, then we have that* $H\sigma \in \mathcal{H}(\mathsf{solved}(K))$ *with a proof tree* π' *matching with* exec *and* $R_1, \ldots, R_k$*, and such that* $\mathsf{nodes}(\pi') \subseteq \bigcup_{i\in\{1,\ldots,n\}} \mathsf{nodes}(\pi_i) \cup \{H\sigma\}$*.*

Proof. We prove this result by induction on the sum of the sizes of the proof trees witnessing that $B_1\sigma, \ldots, B_n\sigma \in \mathcal{H}(\mathsf{solved}(K))$. If g is solved, then since $g \in K$, we conclude by choosing π' to be $\pi_1, \ldots, \pi_n$ on which we apply Conseq with g and σ.

Otherwise, i.e. g is not solved. Let $B_j = \mathsf{sel}(g) = \mathsf{k}_{u_j}(X_j, t_j)$. By hypothesis, we have that $B_j\sigma \in \mathcal{H}(\mathsf{solved}(K))$ with a proof tree π_j matching with exec and $R_1, \ldots, R_k$ as input recipes. Therefore, π_j is ending with a statement

$$h = \mathsf{k}_{u'_0}(R_0, t_0) \Leftarrow B'_1, \ldots, B'_m \in \mathsf{solved}(K)$$

and a substitution σ' grounding for h such that $\mathsf{k}_{u'_0}(R_0, t_0)\sigma' = \mathsf{k}_{u_j}(X_j, t_j)\sigma$ and $B'_i\sigma' \in \mathcal{H}(\mathsf{solved}(K))$ for $i \in \{1, \ldots, m\}$ with a proof tree π'_i (subtree of π_j) matching with exec and $R_1, \ldots, R_k$.

Moreover, we have that the sum of the size of the proof tree witnessing that $B'_i\sigma' \in \mathcal{H}(\mathsf{solved}(K))$ for $i \in \{1, \ldots, m\}$ is smaller than the size of the proof tree π_j. Let $H_0 = \mathsf{k}_{u'_0}(R_0, t_0)$. We apply the Resolution rule between g and h. Since $\sigma \uplus \sigma'$ unifies H_0 and $\mathsf{k}_{u_j}(X_j, t_j)$, there is $\omega = \mathsf{mgu}(H_0, \mathsf{k}_{u_j}(X_j, t_j))$ and τ such that $\sigma \uplus \sigma' = \omega\tau$. Let g' be the resulting statement. We have that:

$$g' = H\omega \Leftarrow B_1\omega, \ldots, B_{j-1}\omega, B_{j+1}\omega, \ldots, B_n\omega, B'_1\omega, \ldots, B'_m\omega.$$

In order to conclude relying on our induction hypothesis, we distinguish two cases.

Case 1: $g'\Downarrow_c$ is added to the knowledge base by the update function. We conclude relying on our induction hypothesis considering $g'\Downarrow_c$ and τ. We have that $u_0\omega$ is locally closed. We have that $\mathsf{skl}(g'\Downarrow_c\tau)$ is in normal form since $\mathsf{skl}(g\sigma)$ and $\mathsf{skl}(h\sigma')$ are in normal form. The recipe occurring in $H\omega\tau = H\sigma$ (if any) is asap w.r.t. $b_{|\mathsf{Rcv}(q)|}$ and exec and all the antecedents of $g'\Downarrow_c\tau$ are in $\mathcal{H}(\mathsf{solved}(K))$ with a proof tree matching with exec and $R_1,\ldots,R_k$. Finally we have that the set made of all the proof trees $\{\pi_1,\ldots,\pi_{j-1},\pi_{j+1},\ldots,\pi_n,\pi'_1,\ldots,\pi'_m\}$ is uniform. It remains to show that $g'\Downarrow_c$ and τ match exec and $R_1,\ldots,R_k$. To do so we first show that g' and τ match exec and $R_1,\ldots,R_k$.

Given a world u, i.e. a sequence of actions possibly followed by a variable, we denote by $|u|$ the number of actions in the sequence u. By definition of the RES rule and the form of the statement, we have that :

a) either $|u_j\omega| = |u_j|$, and thus $|u_0\omega| = |u_0|$;

b) or $|u_j\omega| > |u_j|$, and in such a case, we have that $u_0\omega = u_j\omega = u'_0\omega$.

We consider these two cases separately. In the first case, we will rely on the fact that g and σ match with exec and $R_1,\ldots,R_k$, whereas in the second case, we will rely on the fact that h and σ' match with exec and $R_1,\ldots,R_k$,

Case a): $|u_0\omega| = |u_0|$. By hypothesis, we have that g and σ match with exec and $R_1,\ldots,R_k$, thus we know that there exist recipes $\hat{R}_1,\ldots,\hat{R}_{k'}$ such that:

- $\hat{R}_j(\{X_i \to t_i \mid 1 \le i \le n\} \uplus \phi(u_0))\Downarrow = v_j$ for $j \in \{1,\ldots,k'\}$;
- $\hat{R}_j\sigma = R_i$ for $j \in \{1,\ldots,k'\}$; and
- $u_0\sigma \sqsubseteq \ell_1,\ldots,\ell_p$

where $v_1,\ldots,v_{k'}$ are the terms occurring in input in u_0.

To establish that g' and τ match with exec and $R_1,\ldots,R_k$, we consider the recipes $\hat{R}_1\omega,\ldots,\hat{R}_{k'}\omega$. Let $v_1,\ldots,v_{k'}$ be the terms occurring in input in $u_0\omega$, and let us denote $B'_i = \mathsf{k}\_(X'_i, x'_i)$ for $1 \le i \le m$. The only difficult point is to show that:

$$\hat{R}_j\omega(\{X_i \to t_i\omega \mid 1 \le i \le n \text{ and } i \neq j\} \uplus \{X'_i \to x'_i\omega \mid 1 \le i \le m\} \uplus \phi(u_0\omega))\Downarrow = v'_j$$

Let $j \in \{1,\ldots,k'\}$. By hypothesis, $\hat{R}_j(\{X_i \to t_i \mid 1 \le i \le n\} \uplus \phi(u_0))\Downarrow = v_j$, and thus, since $v_j\omega$ is in normal form, we have that:

$$(\hat{R}_j\{X_j \to t_j\omega\})(\{X_i \to t_i\omega \mid 1 \le i \le n \text{ and } i \neq j\} \uplus \phi(u_0\omega))\Downarrow = v_j\omega$$

By definition of being a statement (invariant applied on h), we have that:

$$R_0(\{X'_i \to x'_i \mid 1 \le i \le m\} \uplus \phi(u'_0))\Downarrow = t_0$$

Applying ω (note that $R_0\omega = R_0$), we deduce that:

$$R_0(\{X'_i \to x'_i\omega \mid 1 \leq i \leq m\} \uplus \phi(u'_0\omega))\Downarrow = t_0\omega$$

Therefore, we have that:

$$\begin{array}{l}
\hat{R}_j\omega(\{X_i \to t_i\omega \mid 1 \leq i \leq n \text{ and } i \neq j\} \uplus \{X'_i \to x'_i\omega \mid 1 \leq i \leq m\} \uplus \phi(u_0\omega))\Downarrow \\
= (\hat{R}_j\{X_j \to R_0\}) \left(\begin{array}{l} \{X_i \to t_i\omega \; 1 \leq i \leq n \text{ and } i \neq j\} \\ \uplus \; \{X'_i \to x'_i\omega \mid 1 \leq i \leq m\} \\ \uplus \; \phi(u_0\omega) \end{array} \right)\Downarrow \\
= (\hat{R}_j\{X_j \to R_0\{X'_i \to x'_i\omega \mid 1 \leq i \leq m\}\}) \left(\begin{array}{l} \{X_i \to t_i\omega \; 1 \leq i \leq n \text{ and } i \neq j\} \\ \uplus \; \phi(u_0\omega) \end{array} \right)\Downarrow \\
= (\hat{R}_j\{X_j \to t_0\omega\})(\{X_i \to t_i\omega \mid 1 \leq i \leq n \text{ and } i \neq j\} \uplus \phi(u_0\omega))\Downarrow \\
= (\hat{R}_j\{X_j \to t_j\omega\})(\{X_i \to t_i\omega \mid 1 \leq i \leq n \text{ and } i \neq j\} \uplus \phi(u_0\omega))\Downarrow \\
= v_j\omega = v'_j
\end{array}$$

We have that g' and τ match with exec and $R_1, \ldots, R_k$.

Case b): $u_0\omega = u'_0\omega$. By hypothesis, we know that h and σ' match with exec and $R_1, \ldots, R_k$, thus we know that there exist recipes $\hat{R}'_1, \ldots, \hat{R}'_{k'}$ such that:

- $\hat{R}'_j(\{X'_i \to x'_i \mid 1 \leq i \leq m\} \uplus \phi(u'_0))\Downarrow = v_j$ for $j \in \{1, \ldots, k'\}$;
- $\hat{R}'_j\sigma' = R_i$ for $j \in \{1, \ldots, k'\}$; and
- $u'_0\sigma' \sqsubseteq \ell_1, \ldots, \ell_p$.

where $v_1, \ldots, v_{k'}$ are the terms occurring in input in u'_0 and $B'_i = \mathsf{k}\_(X'_i, x'_i)$ for $1 \leq i \leq m$. To establish that g' and τ match with exec and $R_1, \ldots, R_k$, we consider $\hat{R}'_1\omega, \ldots, \hat{R}'_{k'}\omega$. Let $v'_1, \ldots, v'_{k'}$ be the terms occurring in input in $u_0\omega$. The only difficult part is to show that:

$$\hat{R}_j\omega(\{X_i \to t_i\omega \mid 1 \leq i \leq n \text{ and } i \neq j\} \uplus \{X'_i \to x'_i\omega \mid 1 \leq i \leq m\} \uplus \phi(u_0\omega))\Downarrow = v'_j$$

By hypothesis, we have that $\hat{R}_j(\{X'_i \to x'_i \mid 1 \leq i \leq m\} \uplus \phi(u'_0))\Downarrow = v_j$, and thus, since $v_j\omega$ is in normal form, $X_i \notin vars(\hat{R}_j)$ for all $i \in \{1, \ldots, n\}$, and $dom(\omega) \cap \mathcal{Y} = \{X_j\}$, we have that:

$$\begin{array}{l}
\hat{R}_j\omega(\{X_i \to t_i\omega \mid 1 \leq i \leq n \text{ and } i \neq j\} \uplus \{X'_i \to x'_i\omega \mid 1 \leq i \leq m\} \uplus \phi(u_0\omega))\Downarrow \\
= \hat{R}_j(\{X'_i \to x'_i\omega \mid 1 \leq i \leq m\} \uplus \phi(u'_0\omega))\Downarrow = v_j\omega = v'_j
\end{array}$$

We have that g' and τ match with exec and $R_1, \ldots, R_k$.

In both cases we have that g' and τ match with exec and $R_1, \ldots, R_k$. Moreover, because $\pi'_1, \ldots, \pi'_m$ are subtrees of π_j we have that $\{\pi_1, \ldots, \pi_{j-1}, \pi_{j+1}, \ldots, \pi_n, \pi'_1, \ldots, \pi'_m\}$ is uniform.

A

Claim. *We have that $g'\Downarrow_c$ and τ match with* exec *and $R_1, \ldots, R_k$.*

Proof. This claim is proved by induction on the number p of application of the rule Remove. In case $p = 0$, we have that $g' = g'\Downarrow_c$ and the result is immediate. Now, we assume that the body of g' contains two predicates $\mathsf{k}_w(X, u)$ and $\mathsf{k}_w(Y, u)$. Since $\{\pi_1, \ldots, \pi_{j-1}, \pi_{j+1}, \ldots, \pi_n, \pi'_1, \ldots, \pi'_m\}$ is uniform we know that $X\tau = Y\tau$ and thus we can replace X by Y in a recipe without changing the term that it deduced. Hence we have that the resulting statement after the application of the Remove rule still match with exec and $R_1, \ldots, R_k$ and we conclude relying on the induction hypothesis. □

Our induction hypothesis applies and we obtain that $H\omega\tau \in \mathcal{H}(\mathsf{solved}(K))$ with a proof tree π' matching exec and $R_1, \ldots, R_k$ and such that:

$$\begin{array}{rcl} \mathsf{nodes}(\pi') & \subseteq & \bigcup_{i\in\{1,\ldots,n\}\setminus\{j\}} \mathsf{nodes}(\pi_i) \cup \bigcup_{i\in\{1,\ldots,m\}} \mathsf{nodes}(\pi'_i) \cup \{H\omega\tau\} \\ & \subseteq & \bigcup_{i\in\{1,\ldots,n\}} \mathsf{nodes}(\pi_i) \cup \{H\sigma\} \end{array}$$

because for all $i \in \{1, \ldots, m\}$ we have that π'_i is a subtree of π_j. This concludes the first case of this proof.

Case 2: $g'\Downarrow_c$ is not added to the knowledge base by the update function.

Let $H = \mathsf{k}_{u_0}(R_H, t_H)$. In such a case, we know that $g'\Downarrow_c$ is a solved deduction statement, and since such a statement has been discarded, it means that $t_H\omega$ is a variable x. Let $g'\Downarrow_c = \mathsf{k}_{u_0\omega}(R_H\omega, x) \Leftarrow \mathsf{k}\_(Z_1, z_1), \ldots, \mathsf{k}\_(Z_q, z_q)$. First, following the same reasoning as previously we have that $g'\Downarrow_c$ match with exec and $R_1, \ldots, R_k$. We note $\hat{R}_1, \ldots, \hat{R}_k$ the symbolic recipes that are used to establish this fact. Moreover, by definition of being a statement, we know that $R_H\omega(\{Z_i \to z_i \mid 1 \le i \le q\} \uplus \phi(u_0\omega))\Downarrow = x$.

Either, the variable x has been introduced by $\{Z_i \to z_i\}$ for some i such that $z_i = x$, and Z_i occurring in $R_H\omega$. In such a case, we have that Z_i is a strict subterm of $R_H\omega$ since $R_H\omega$ is not a variable, and therefore $Z_i\tau <_{\mathsf{exec}} R_H\omega\tau$. Otherwise, x is introduced by a frame element $\mathsf{w} \to t$ with $x \in vars(t)$, and $\mathsf{w} \in vars(R_H\omega)$. Therefore, we have that $\mathsf{w} \le_{\mathsf{exec}} R_H\omega\tau$. Because $u_0\omega$ is locally closed, we know that x occurs in an input in $u_0\omega$ (the action $\ell_i = (a_i, \mathtt{in}(v_i))$), and we have that $R_i <_{\mathsf{exec}} \mathsf{w}$.

Claim. *There exists $i_0 \in \{1, \ldots, q\}$ such that $x = z_{i_0}$ and $Z_{i_0}\tau \le_{\mathsf{exec}} \hat{R}_i\omega\tau = R_i <_{\mathsf{exec}} \mathsf{w} \le_{\mathsf{exec}} R_H\omega\tau$.*

Proof. We prove this claim by induction on i. If ℓ_i is the first input in $u_0\omega$. We have that $\hat{R}_i\omega\tau = R_i$ and since R_i only uses frame elements occurring before the first input, this is the same for $\hat{R}_i$ and thus $\phi(u_0\omega)$ will not introduce any variable. Therefore x is introduced by $\{Z_{i_0} \to z_{i_0}\}$ for some i_0 such that $z_{i_0} = x$ and Z_{i_0} occurs in $\hat{R}_i$ and thus $Z_{i_0}\tau \le_{\mathsf{exec}} \hat{R}_i\omega\tau = R_i <_{\mathsf{exec}} \mathsf{w} \le_{\mathsf{exec}} R_H\omega\tau$.

Otherwise, if ℓ_i is not the first input then either x has been introduced by $\{Z_{i_0} \to z_{i_0}\}$ and the same reasoning applies, or it is introduced by a frame element $w' \le_{\mathsf{exec}} \hat{R}_i$. Since $u_0\omega$ is locally closed then x occurs in another input $j < i$ and $R_j = \hat{R}_j\tau <^{\mathsf{in}}_{\mathsf{exec}} w'$. Applying our induction on $j < i$ we obtain there exists i_0 such $z_{i_0} = x$ and $Z_{i_0} \le_{\mathsf{exec}} \hat{R}_j\tau$ and we conclude by transitivity of $\le_{\mathsf{exec}}$. □

Therefore, in both cases, we have that there exists $i_0 \in \{1, \ldots, q\}$ such that $x = z_{i_0}$ and $Z_{i_0}\tau <_{\mathsf{exec}} R_H\omega\tau$. Thanks to Theorem 3.3, we know that $Z_{i_0}\tau$ is a recipe for $x\tau$. If $R_H\omega\tau \in \mathcal{W}$ we immediately contradict that $R_H\omega\tau = R_H\sigma$ is asap w.r.t. $b_{|\mathsf{Rcv}(q)|}$ and exec. Otherwise, applying Lemma 3.6 we obtain that $Z_{i_0}\tau <^{b_{|\mathsf{Rcv}(q)|}}_{\mathsf{exec}} R_H\omega\tau$ and this leads to a contradiction with the fact that $R_H\omega\tau = R_H\sigma$ is supposed to be asap w.r.t. $b_{|\mathsf{Rcv}(q)|}$ and exec. This conclude the whole proof.

□

Executions with asap recipes

Lemma 3.12. *Let* $\mathsf{exec} = K_0 \xrightarrow{\ell_1,\ldots,\ell_n}_{\mathcal{T}_0} (S; \Phi; t)$ *be an execution. We may assume w.l.o.g. that* exec *involves input recipes* $R_1, \ldots, R_k$ *forged by agents* $b_1, \ldots, b_k$ *and* R_i *is asap w.r.t.* b_i *and* exec *for each* $i \in \{1, \ldots, k\}$.

Proof. We consider that the execution exec is done with input recipes $R_1, \ldots, R_k$, forged by agents $b_1, \ldots, b_k$. We assume that for all $i \in \{1, \ldots, k\}$:

- if $b_i \in \mathcal{M}_0$ then there is no recipe R that can fill the input (i.e. satisfying the domain restrictions and the timing constraints of the input) and such that $R\Phi\Downarrow = R_i\Phi\Downarrow$ and $R <^{b_i}_{\mathsf{exec}} R_i$;
- if $b_i \notin \mathcal{M}_0$ then there is no recipe R that can fill the input and such that $R\Phi\Downarrow = R_i\Phi\Downarrow$ and $R <_{\mathsf{exec}} R_i$.

We prove that $R_1, \ldots, R_k$ are asap recipes w.r.t. $b_1, \ldots, b_k$ and exec by induction on the length n of the execution.

Base case: $n = 0$. In such a case, the result is immediate.

Induction step. In such a case, we have

$$\mathsf{exec} = (T; \emptyset; t_0) \xrightarrow{\ell_1,\ldots,\ell_{n-1}}_{\mathcal{T}_0} (S'; \Phi'; t') \xrightarrow{\ell_n}_{\mathcal{T}_0} (S; \Phi; t)$$

together with recipes $R_1, \ldots, R_k$ forged by $b_1, \ldots, b_k$ that satisfy our assumption. We note exec_0 the sub execution from $(T; \emptyset; t_0)$ to $(S'; \Phi'; t')$. We distinguish two cases depending on the action ℓ_n.

Case ℓ_n is not an input. Thanks to our induction hypothesis, we know that R_i $(1 \leq i \leq k)$ is asap w.r.t. b_i and exec_0. Then, we complete this execution exec_0 performing the action ℓ_n, and we obtain $\mathsf{exec} = (T; \emptyset; t_0) \xrightarrow{\ell_1,\ldots,\ell_n}_{\mathcal{T}_0} (S; \Phi; t)$ with input recipes $R_1, \ldots, R_k$ such that R_i is asap w.r.t. b_i and exec_0 for any $i \in \{1, \ldots, k\}$. It remains to show that $R_1, \ldots, R_k$ are still asap when considering the full execution exec. Let us distinguish two cases:

If $R_i \in \mathcal{W}$: Since the relation $<_{\mathsf{exec}_0}$ induced by exec_0 is the same as the one induced by exec on recipes built using $dom(\Phi')$, we have that R_i is still asap w.r.t. b_i and exec.

If $R_i \notin \mathcal{W}$: for all recipe R such that $R\phi\Downarrow = R_i\Phi\Downarrow$ we have that if $vars(R) \subseteq dom(\Phi')$ then $R_i \leq^{b_i}_{\mathsf{exec}_0} R$ and thus $R_i \leq^{b_i}_{\mathsf{exec}} R$. Otherwise, we have that there exists a unique

A

$\mathsf{w} \in vars(R) \cap (dom(\Phi) \setminus dom(\Phi'))$ and $\mathsf{time}(\mathsf{w}) = t'$. Such a w corresponds to the handle bound by ℓ_n when ℓ_n is an output action. For any $\mathsf{w}' \in vars(R_i)$, we have that: $\mathsf{time}(\mathsf{w}') + \mathsf{Dist}_{\mathcal{T}_0}(\mathsf{agent}(\mathsf{w}), b_i) \leq \mathsf{time}(\mathsf{w})$, and thus $\mathsf{time}(\mathsf{w}') + \mathsf{Dist}_{\mathcal{T}_0}(\mathsf{agent}(\mathsf{w}), b_i) \leq \mathsf{time}(\mathsf{w}) + \mathsf{Dist}_{\mathcal{T}_0}(\mathsf{agent}(\mathsf{w}'), b_i)$. To conclude it is sufficient to notice that either it is a strict inequality and thus we immediately have that $\mathsf{w}' <^{b_i}_{\mathsf{exec}} \mathsf{w}$ or we have an equality but since w' has been outputted before w in exec we have that $\mathsf{w}' <^{b_i}_{\mathsf{exec}} \mathsf{w}$ too. Finally we have that such a recipe R which contains w can not be smaller than R_i, i.e. $R \not<^{b_i}_{\mathsf{exec}} R_i$.

Case ℓ_n is an input, i.e. $\ell_n = (a, \mathsf{in}^z(u))$. Thanks to our induction hypothesis, we know that R_i $(1 \leq i \leq k-1)$ is asap w.r.t. b_i and exec_0. Then, we complete this execution exec_0 performing the action $\ell_n = (a, \mathsf{in}^z(u))$ with recipe R_k forged by b_k, and we obtain

$$\mathsf{exec} = (T; \emptyset; t_0) \xrightarrow{\ell_1, \ldots, \ell_n}_{\mathcal{T}_0} (S; \Phi; t)$$

with input recipes $R_1, \ldots, R_{k-1}, R_k$. First because $\Phi = \Phi'$ we have that $<_{\mathsf{exec}_0}$ and $<^a_{\mathsf{exec}_0}$ for any $a \in \mathcal{A}$, the relations induced by exec_0, are the same as the ones induced by exec. Therefore we have that R_i $(1 \leq i \leq k-1)$ is still asap w.r.t. b_i and exec. To conclude, it remains to establish that R_k is asap w.r.t. b_k and exec.

Following the semantics of the IN rule, we know that there exists $t_b \in \mathbb{R}_+$ such that $t_b \leq t - \mathsf{Dist}_{\mathcal{T}_0}(b_k, a)$ and

- if $b_k \in \mathcal{A}_0 \setminus \mathcal{M}_0$ then $R_k \in \mathcal{W} \uplus \Sigma_0 \uplus \mathbb{R}_+$. In addition, if $R_k = \mathsf{w}$ then $\mathsf{w} \in dom(\lfloor \Phi' \rfloor^{t_b}_{b_k})$;
- if $b_k \in \mathcal{M}_0$ then for all $\mathsf{w} \in vars(R_k)$, there exists $c \in \mathcal{A}_0$ such that $\mathsf{w} \in dom(\lfloor \Phi' \rfloor^{t_b - \mathsf{Dist}_{\mathcal{T}_0}(c, b_k)}_{c})$.

Let us assume that R_k is not asap w.r.t. b_k and exec.

- Case $b_k \in \mathcal{M}_0$. We have that $R_k \notin \Sigma_0 \cup \mathbb{R}_+$ and there exists R'_k such that $R'_k\Phi\Downarrow = R_k\Phi\Downarrow$ and either $R'_k <_{\mathsf{exec}} R_k$ or $R'_k <^{b_k}_{\mathsf{exec}} R_k$. Applying Lemma 3.6 we obtain that in both cases $R'_k <^{b_k}_{\mathsf{exec}} R_k$ and by consequence we have that $\mathsf{multi}_{\mathcal{W}}(R'_k) \leq^{b_k}_{\mathsf{exec}} \mathsf{multi}_{\mathcal{W}}(R_k)$. By definition of the multiset order, we deduce that for all $\mathsf{w}' \in vars(R'_k)$, there exists $\mathsf{w} \in vars(R_k)$ such that $\mathsf{w}' \leq^{b_k}_{\mathsf{exec}} \mathsf{w}$. By definition of the order over frame variables we have:

  $$\mathsf{time}(\mathsf{w}') + \mathsf{Dist}_{\mathcal{T}_0}(\mathsf{agent}(\mathsf{w}'), b_k) \leq \mathsf{time}(\mathsf{w}) + \mathsf{Dist}_{\mathcal{T}_0}(\mathsf{agent}(\mathsf{w}), b_k).$$

  Since we have that $\mathsf{w} \in dom(\lfloor \Phi' \rfloor^{t_b - \mathsf{Dist}_{\mathcal{T}_0}(\mathsf{agent}(\mathsf{w}), b_k)}_{\mathsf{agent}(\mathsf{w})})$, we know that $\mathsf{time}(\mathsf{w}) \leq t_b - \mathsf{Dist}_{\mathcal{T}_0}(\mathsf{agent}(\mathsf{w}), b_k)$. Therefore, we have that for all $\mathsf{w}' \in vars(R'_k)$, there exists $\mathsf{w} \in vars(R_k)$ such that:

  $$\begin{array}{rcl} \mathsf{time}(\mathsf{w}') + \mathsf{Dist}_{\mathcal{T}_0}(\mathsf{agent}(\mathsf{w}'), b_k) & \leq & t_b - \mathsf{Dist}_{\mathcal{T}_0}(\mathsf{agent}(\mathsf{w}), b_k) \\ & & + \mathsf{Dist}_{\mathcal{T}_0}(\mathsf{agent}(\mathsf{w}), b_k) \\ \Rightarrow \mathsf{time}(\mathsf{w}') & \leq & t_b - \mathsf{Dist}_{\mathcal{T}_0}(\mathsf{agent}(\mathsf{w}'), b_k) \end{array}$$

  and hence the IN rule can be executed using R'_k forged by b_k at time t_b. This contradicts the assumption on R_k.

- Case $b_k \in \mathcal{A}_0 \setminus \mathcal{M}_0$. In that case, $R_k \in \mathcal{W} \cup \mathbb{R}_+$ and since R_k is not asap, we know that there exists $R'_k <_{\mathsf{exec}} R_k$ such that $R'_k\Phi'\Downarrow = R_k\Phi'\Downarrow$. We consider the chain $R'_k <_{\mathsf{exec}} \ldots <_{\mathsf{exec}} R_k$ (each step corresponding to a step of $<^{\mathsf{in}}_{\mathsf{exec}}$ or $<^{\mathsf{sub}}_{\mathsf{exec}}$ under a context) witnessing the fact that $R'_k <_{\mathsf{exec}} R_k$. Let $\mathsf{w}' \in \mathcal{W}$ be the smallest variable w.r.t. $<^{\mathsf{in}}_{\mathsf{exec}}$ such that $\mathsf{w}' <^{\mathsf{in}}_{\mathsf{exec}} \ldots \mathsf{w}'' <^{\mathsf{in}}_{\mathsf{exec}} \mathsf{w}$. In case such a w' does not exist, we consider that $\mathsf{w}' = \mathsf{w}$.

  We show, by induction on the length l of $\mathsf{w}' <^{\mathsf{in}}_{\mathsf{exec}} \ldots \mathsf{w}'' <^{\mathsf{in}}_{\mathsf{exec}} \mathsf{w}$ that if $\mathsf{w} \in dom(\lfloor\Phi'\rfloor^{t_1}_{b_1})$ for some b_1 and t_1 then $\mathsf{w}' \in dom(\lfloor\Phi'\rfloor^{t_b-\mathsf{Dist}_{\mathcal{T}_0}(b_2,b_1)}_{b_2})$ for some b_2. Indeed, if $l = 0$ then choosing $b_2 = b_1$, we immediately conclude. Otherwise, since w is outputted at time t_1 (at least) by b_1 then the input recipe w'' has been built by some agent b'' at time $t''_b \leq t_1 - \mathsf{Dist}_{\mathcal{T}_0}(b'', b_1)$. We have that $\mathsf{w}'' \in dom(\lfloor\Phi'\rfloor^{t_1-\mathsf{Dist}_{\mathcal{T}_0}(b'',b_1)}_{b''})$. We apply the induction hypothesis using $t_1 - \mathsf{Dist}_{\mathcal{T}_0}(b'', b_1)$, w'' and b'' to obtain that $\mathsf{w}' \in dom(\lfloor\Phi'\rfloor^{t_1-\mathsf{Dist}_{\mathcal{T}_0}(b'',b_1)-\mathsf{Dist}_{\mathcal{T}_0}(b_2,b'')}_{b_2})$ for some b_2. Therefore we have that $\mathsf{w}' \in dom(\lfloor\Phi'\rfloor^{t_1-\mathsf{Dist}_{\mathcal{T}_0}(b_2,b_1)}_{b_2})$.

  Applying this property to $\mathsf{w}' <^{\mathsf{in}}_{\mathsf{exec}} \ldots \mathsf{w}'' <^{\mathsf{in}}_{\mathsf{exec}} \mathsf{w}$, t_b and b_k, we obtain that $\mathsf{w}' \in dom(\lfloor\Phi'\rfloor^{t_b-\mathsf{Dist}_{\mathcal{T}_0}(b',b_k)}_{b'})$ for some b'. Therefore, because the message u is received by the agent a at time t' and u is forged by b_k at time t_b, we have that:

$$\begin{array}{rrcl} & t_b & \leq & t' - \mathsf{Dist}_{\mathcal{T}_0}(b_k, a) \\ \Rightarrow & t_b - \mathsf{Dist}_{\mathcal{T}_0}(b', b_k) & \leq & t' - (\mathsf{Dist}_{\mathcal{T}_0}(b_k, a) + \mathsf{Dist}_{\mathcal{T}_0}(b', b_k)) \\ \Rightarrow & t_b - \mathsf{Dist}_{\mathcal{T}_0}(b', b_k) & \leq & t' - \mathsf{Dist}_{\mathcal{T}_0}(b', a) \end{array}$$

  If $R'_k = \mathsf{w}'$ then this last inequality give us that the rule IN can be triggered with the recipe w' considering the output is performed by b' at time $t_b - \mathsf{Dist}_{\mathcal{T}_0}(b', b_k)$. This contradicts that R_k is asap w.r.t. b_k and exec.

  Otherwise, we have that there exists $R'' \notin \mathcal{W}$ such that $R'_k <_{\mathsf{exec}} R'' <^{\mathsf{in}}_{\mathsf{exec}} \mathsf{w}'$. This input received by b' (the same agent as the one who sent w') has been built by some b'' at time t''_b. Since R'', built by b'', is received by b' before outputting w' (available in b' at time $t_b - \mathsf{Dist}_{\mathcal{T}_0}(b', b_k)$), we have that $t''_b + \mathsf{Dist}_{\mathcal{T}_0}(b'', b') \leq t_b - \mathsf{Dist}_{\mathcal{T}_0}(b', b_k)$. Moreover, we have that for all $\mathsf{w} \in vars(R'')$, there exists $c \in \mathcal{A}_0$ such that $\mathsf{w} \in dom(\lfloor\Phi'\rfloor^{t''_b-\mathsf{Dist}_{\mathcal{T}_0}(c,b'')}_{c})$. Thanks to Lemma 3.6 we have that $R'_k <^{b''}_{\mathsf{exec}} R''$ and thus $\mathsf{multi}_{\mathcal{W}}(R'_k) \leq^{b''}_{\mathsf{exec}} \mathsf{multi}_{\mathcal{W}}(R'')$. Therefore, for all $\mathsf{w}_k \in vars(R'_k)$, there exists $\mathsf{w}'' \in vars(R'')$ such that $\mathsf{w}_k \leq^{b''}_{\mathsf{exec}} \mathsf{w}''$. We thus have that $\mathsf{time}(\mathsf{w}_k) + \mathsf{Dist}_{\mathcal{T}_0}(\mathsf{agent}(\mathsf{w}_k), b'') \leq \mathsf{time}(\mathsf{w}'') + \mathsf{Dist}_{\mathcal{T}_0}(\mathsf{agent}(\mathsf{w}''), b'')$ and thus $\mathsf{w}_k \in dom(\lfloor\Phi'\rfloor^{t''_b-\mathsf{Dist}_{\mathcal{T}_0}(\mathsf{agent}(\mathsf{w}_k),b'')}_{\mathsf{agent}(\mathsf{w}_k)})$ because $\mathsf{time}(\mathsf{w}'') \leq t''_b - \mathsf{Dist}_{\mathcal{T}_0}(\mathsf{agent}(\mathsf{w}''), b'')$.

  This allows us to obtain that the rule IN can be triggered with recipe $R'_k \notin \mathcal{W}$ considering the message is built by b'' and time t''_b. Indeed, we have that:

$$\begin{array}{rrcl} & t_b & \leq & t' - \mathsf{Dist}_{\mathcal{T}_0}(b_k, a) \\ \Rightarrow & t_b - \mathsf{Dist}_{\mathcal{T}_0}(b_k, b') & \leq & t' - \mathsf{Dist}_{\mathcal{T}_0}(b', a) \text{ as before} \\ \Rightarrow & t''_b + \mathsf{Dist}_{\mathcal{T}_0}(b'', b') & \leq & t' - \mathsf{Dist}_{\mathcal{T}_0}(b', a) \\ \Rightarrow & t''_b & \leq & t' - \mathsf{Dist}_{\mathcal{T}_0}(b'', a). \end{array}$$

This contradicts the assumption on R_k because $R'_k <_{\mathsf{exec}} R_k$.

In conclusion, in all cases we obtain a contradiction with the initial assumption. Therefore we conclude that R_k is asap w.r.t. b_k and exec. □

Proofs of Chapter 4 B

In this chapter we provide the omitted proofs in Chapter 4.

B.1. Proof of Proposition 4.2

Proposition 4.2. *Let $\mathcal{T}$ be a topology, and $\mathcal{K}_0 \overset{\mathsf{tr}_1 \ldots \mathsf{tr}_n}{\rightsquigarrow}_{\mathcal{T}} \mathcal{K}_n$ be an execution with $n \geq 2$. We have that there exists a bijection $\varphi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ such that:*

- $\mathcal{K}_0 \overset{\mathsf{tr}'_1 \ldots \mathsf{tr}'_n}{\rightsquigarrow}_{\mathcal{T}} \mathcal{K}_n$ *with* $\mathsf{tr}_i = \mathsf{tr}'_{\varphi(i)}$ *for all* $i \in \{1, \ldots, n\}$*; and*
- *for all j such that $\varphi(1) < j < \varphi(n)$, we have that $\mathsf{tr}'_{\varphi(n)} \hookrightarrow^* \mathsf{tr}'_j \hookrightarrow^* \mathsf{tr}'_{\varphi(1)}$.*

Proof. We split the proof in two parts: first we prove that there exists a bijection φ_1 cleaning the trace between tr_1 and tr_n moving actions independent from tr_1 before it. Then we prove that there exists a bijection φ_2 cleaning the trace moving actions from which tr_n does not depend on after it. Considering $\varphi = \varphi_2 \circ \varphi_1$ we will be able to conclude.

Claim. *Let $\mathcal{K}_0 \overset{\mathsf{tr}_1 \ldots \mathsf{tr}_n}{\rightsquigarrow}_{\mathcal{T}} \mathcal{K}_n$ be an execution with $n \geq 1$. There exists a bijection $\varphi_1 : \{1, \ldots, n\} \to \{1, \ldots, n\}$ such that:*

- $\mathcal{K}_0 \overset{\mathsf{tr}'_1 \ldots \mathsf{tr}'_n}{\rightsquigarrow}_{\mathcal{T}} \mathcal{K}_n$ *with* $\mathsf{tr}_i = \mathsf{tr}'_{\varphi_1(i)}$ *for all* $i \in \{1, \ldots, n\}$*; and*
- *for all $j > \varphi_1(1)$, $\mathsf{tr}'_j \hookrightarrow^* \mathsf{tr}'_{\varphi_1(1)}$.*

Proof. We show this claim by induction on the length of the execution. If $n = 1$ then the results holds considering $\varphi_1 = id$ and $\mathsf{tr}'_1 = \mathsf{tr}_1$. Otherwise, we have that $\mathcal{K}_0 \overset{\mathsf{tr}_1 \ldots \mathsf{tr}_n}{\rightsquigarrow}_{\mathcal{T}} \mathcal{K}_n \overset{\mathsf{tr}_{n+1}}{\rightsquigarrow}_{\mathcal{T}} \mathcal{K}_{n+1}$, and by induction hypothesis we have that there exists a bijection $\varphi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ such that:

- $\mathcal{K}_0 \overset{\mathsf{tr}'_1 \ldots \mathsf{tr}'_n}{\rightsquigarrow}_{\mathcal{T}} \mathcal{K}_n$ with $\mathsf{tr}_i = \mathsf{tr}'_{\varphi(i)}$ for all $i \in \{1, \ldots, n\}$; and
- for all j such that $\varphi(1) < j \leq n$, we have $\mathsf{tr}'_j \hookrightarrow^* \mathsf{tr}'_{\varphi(1)}$.

If $\mathsf{tr}_{n+1} \hookrightarrow^* \mathsf{tr}'_{\varphi(1)} (= \mathsf{tr}_1)$ then we consider the bijection $\varphi_1 = \varphi \cup \{n+1 \mapsto n+1\}$ and this allows us to conclude. Otherwise, we have that $\mathsf{tr}_{n+1} \not\hookrightarrow^* \mathsf{tr}'_{\varphi(1)} (= \mathsf{tr}_1)$ and, by induction hypothesis, we have $\mathsf{tr}_{n+1} \not\hookrightarrow^* \mathsf{tr}'_j$ for any $\varphi(1) < j \leq n$. Repeatedly applying Lemma 4.6 we obtain that

$$\mathcal{K}_0 \xrightarrow{\mathsf{tr}'_1 \ldots \mathsf{tr}_{n+1} \mathsf{tr}_{\varphi(1)} \ldots \mathsf{tr}'_n}_{\mathcal{T}} \mathcal{K}_{n+1}.$$

Considering the bijection φ_1 defined as follows:

$$\varphi_1(i) = \begin{cases} \varphi(i) & \text{if } \varphi(i) < \varphi(1) \\ \varphi(i)+1 & \text{if } \varphi(i) \geq \varphi(1) \\ \varphi(1) & \text{if } i = n+1 \end{cases}$$

we prove the claim. □

Claim. *Let $\mathcal{K}_0 \xrightarrow{\mathsf{tr}_1 \ldots \mathsf{tr}_n}_{\mathcal{T}} \mathcal{K}_n$ be an execution with $n \geq 1$. There exists a bijection $\varphi_2 : \{1, \ldots, n\} \to \{1, \ldots, n\}$ such that:*

- *$\mathcal{K}_0 \xrightarrow{\mathsf{tr}'_1 \ldots \mathsf{tr}'_n}_{\mathcal{T}} \mathcal{K}_n$ with $\mathsf{tr}_i = \mathsf{tr}'_{\varphi_2(i)}$ for all $i \in \{1, \ldots, n\}$; and*
- *for all $j < \varphi_2(n)$, we have that $\mathsf{tr}'_{\varphi_2(n)} \hookrightarrow^* \mathsf{tr}'_j$.*

Proof. Similarly to the proof done for the previous claim, we first apply the induction hypothesis to $\mathcal{K}_1 \xrightarrow{\mathsf{tr}_2 \ldots \mathsf{tr}_{n+1}}_{\mathcal{T}} \mathcal{K}_{n+1}$ and we obtain $\varphi : \{2, \ldots, n\} \to \{2, \ldots, n\}$ (a shift of 1 has been applied to ease the reasoning). If $\mathsf{tr}_1 \hookrightarrow^* \mathsf{tr}'_{\varphi_2(n)} (= \mathsf{tr}_n)$ then we conclude considering $\varphi_2 = \varphi \cup \{1 \mapsto 1\}$. Otherwise, by repeatedly applying Lemma 4.6, we move tr_1 at the right place in the trace, i.e. just after $\mathsf{tr}'_{\varphi_2(n)}$. □

We are now able to prove the corollary combining these two claims. First we apply Claim 1 considering the trace $\mathcal{K}_0 \xrightarrow{\mathsf{tr}_1 \ldots \mathsf{tr}_{n-1}}_{\mathcal{T}} \mathcal{K}_{n-1}$. We obtain the existence of $\varphi_1 : \{1, \ldots, n-1\} \to \{1, \ldots, n-1\}$ and a new execution

$$\mathcal{K}_0 \xrightarrow{\mathsf{tr}'_1 \ldots \mathsf{tr}'_{n-1}}_{\mathcal{T}} \mathcal{K}_{n-1} \xrightarrow{\mathsf{tr}_n}_{\mathcal{T}} \mathcal{K}_n$$

such that:

- $\mathsf{tr}_i = \mathsf{tr}'_{\varphi_1(i)}$ for all $i \in \{1, \ldots, n-1\}$; and
- for all j such that $\varphi_1(1) < j < n$, we have that $\mathsf{tr}'_j \hookrightarrow^* \mathsf{tr}'_{\varphi_1(1)} (= \mathsf{tr}_1)$.

For sake of uniformity, let $\mathsf{tr}'_n = \mathsf{tr}_n$, and we extend φ_1 on $\{1, \ldots, n\}$ as follows: $\varphi_1(n) = n$.

Then we apply Claim 2 on the resulting execution starting at $\mathsf{tr}'_{\varphi_1(1)+1}$ to obtain that there exists a bijection $\varphi_2 : \{\varphi_1(1)+1, \ldots, n\} \to \{\varphi_1(1)+1, \ldots, n\}$ and an execution

$$\mathcal{K}_0 \overset{\mathsf{tr}'_1 \ldots \mathsf{tr}'_{\varphi_1(1)} \cdot \mathsf{tr}''_{\varphi_1(1)+1} \cdots \mathsf{tr}''_n}{\rightsquigarrow_{\mathcal{T}}} \mathcal{K}_n$$

such that:

- $\mathsf{tr}'_i = \mathsf{tr}''_{\varphi_2(i)}$ for all $i \in \{\varphi_1(1)+1, \ldots, n\}$; and
- for all j such that $\varphi_1(1)+1 \leq j < \varphi_2(n)$, we have that $\mathsf{tr}_n = \mathsf{tr}'_n = \mathsf{tr}''_{\varphi_2(n)} \hookrightarrow^* \mathsf{tr}''_j$.

For sake of uniformity, let $\mathsf{tr}''_1 \ldots \mathsf{tr}''_{\varphi(1)} = \mathsf{tr}'_1 \ldots \mathsf{tr}'_{\varphi(1)}$, and we extend φ_2 on $\{1, \ldots, n\}$ as follows: $\varphi_2(i) = i$ for all $i \in \{1, \ldots, \varphi(1)\}$.

We now show that the bijection $\varphi = \varphi_2 \circ \varphi_1$ satisfies the requirements, i.e.:

1. $\mathsf{tr}_i = \mathsf{tr}''_{\varphi(i)}$ for all $i \in \{1, \ldots, n\}$;
2. for all j such that $\varphi(1) < j < \varphi(n)$, we have that $\mathsf{tr}''_{\varphi(n)} \hookrightarrow^* \mathsf{tr}''_j \hookrightarrow^* \mathsf{tr}''_{\varphi(1)}$.

First, we note that:

- $\varphi(n) = \varphi_2(n)$; and
- $\varphi(i) = \varphi_1(i)$ when $\varphi_1(i) \leq \varphi_1(1)$.

Now, we establish that the 2 requirements are satisfied:

1. First, we have that $\mathsf{tr}_n = \mathsf{tr}'_n = \mathsf{tr}''_{\varphi_2(n)} = \mathsf{tr}''_{\varphi(n)}$. Otherwise, considering $i \in \{1, \ldots, n-1\}$, we have that $\mathsf{tr}_i = \mathsf{tr}'_{\varphi_1(i)}$. Now, we distinguish two cases:
   - $\varphi_1(i) \leq \varphi_1(1)$: we have that $\mathsf{tr}''_{\varphi_1(i)} = \mathsf{tr}'_{\varphi_1(i)}$, and thus $\mathsf{tr}_i = \mathsf{tr}''_{\varphi_1(i)} = \mathsf{tr}''_{\varphi(i)}$.
   - $\varphi_1(i) > \varphi_1(1)$: we have that $\mathsf{tr}'_{\varphi_1(i)} = \mathsf{tr}''_{\varphi_2(\varphi_1(i))} = \mathsf{tr}''_{\varphi(i)}$, and thus $\mathsf{tr}_i = \mathsf{tr}''_{\varphi(i)}$.
2. We have shown that:
   - for all j such that $\varphi_1(1) < j < n$, we have that $\mathsf{tr}'_j \hookrightarrow^* \mathsf{tr}'_{\varphi_1(1)} = \mathsf{tr}''_{\varphi_1(1)}$, and thus for all j such that $\varphi_1(1) < j < \varphi_2(n)$, we have that $\mathsf{tr}''_j \hookrightarrow^* \mathsf{tr}'_{\varphi_1(1)} = \mathsf{tr}''_{\varphi(1)}$ since $\varphi_1(1) = \varphi(1)$ and
   $$\{\mathsf{tr}'_{\varphi_1(1)+1}, \ldots, \mathsf{tr}'_{n-1}\} \supseteq \{\mathsf{tr}''_{\varphi_1(1)+1}, \ldots, \mathsf{tr}''_{\varphi_2(n)-1}\}$$
   - for all j such that $\varphi_1(1)+1 \leq j < \varphi_2(n)(= \varphi(n))$, we have that
   $$\mathsf{tr}_n = \mathsf{tr}'_n = \mathsf{tr}''_{\varphi_2(n)} \hookrightarrow^* \mathsf{tr}''_j.$$

This concludes the proof. □

B.2. Proof of Theorem 4.2

Given a configuration $\mathcal{K} = (\mathcal{P}; \Phi; t)$ (resp. $\mathcal{K} = (\mathcal{P}; \Phi)$), we note $\phi(\mathcal{K})$ its associated frame, i.e. $\phi(\mathcal{K}) = \Phi$.

Theorem 4.2. *Let $\mathcal{I}_0$ be a template, $(\mathtt{V}, \mathtt{P})$ a protocol, and $t_0 \in \mathbb{R}_+$ a threshold. If $(\mathtt{V}, \mathtt{P})$ admits a distance hijacking attack w.r.t. t_0-proximity, then there exists a valid initial configuration $\mathcal{K}_0$ for $(\mathtt{V}, \mathtt{P})$ w.r.t. $\mathcal{T}^{t_0}_{\mathsf{DH}}$ and $\Phi^{\mathcal{T}^{t_0}_{\mathsf{DH}}}_{\mathcal{I}_0}$ such that $\mathcal{K}_0 = (\{ \lfloor \mathtt{V}_{\mathsf{end}}(v_0, p_0) \rfloor^{0}_{v_0} \} \uplus \mathcal{P}_0; \Phi^{\mathcal{T}^{t_0}_{\mathsf{DH}}}_{\mathcal{I}_0}; 0)$ and*

$$(\{ \lfloor \mathtt{V}_{\mathsf{end}}(v_0, p_0) \rfloor^{0}_{v_0} \} \uplus \overline{\mathcal{P}_0}; \Phi^{\mathcal{T}^{t_0}_{\mathsf{DH}}}_{\mathcal{I}_0}; 0) \xrightarrow{\mathsf{tr}}_{\mathcal{T}^{t_0}_{\mathsf{DH}}} (\lfloor \mathsf{end}(v_0, p_0) \rfloor^{t_v}_{v_0} \uplus \mathcal{P}'; \Phi; t).$$

Proof. Let $\mathcal{T} = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}, v_0, p_0) \in \mathcal{C}_{\mathsf{DH}}$ and $\mathcal{K}_0 = (\mathcal{P}_0; \Phi^{\mathcal{T}}_{\mathcal{I}_0}; 0)$ be a valid initial configuration for $(\mathtt{V}, \mathtt{P})$ w.r.t. $\mathcal{T}$ and $\Phi^{\mathcal{T}}_{\mathcal{I}_0}$ such that

$$\mathcal{K}_0 \xrightarrow{\mathsf{tr}}_{\mathcal{T}} (\lfloor \mathsf{end}(v_0, p_0) \rfloor^{t_v}_{v_0} \cup \mathcal{P}; \Phi; t) = \mathcal{K}_{\mathsf{end}}$$

where tr is a sequence of annotated labels.

Step 1: We first remove `reset` commands and replace guarded inputs occurring in processes other than $\mathtt{V}_{\mathsf{end}}(v_0, p_0)$ by simple inputs. Denoting $\mathcal{K}^\star_0$ (resp. $\mathcal{K}^\star_{\mathsf{end}}$), the counterpart of $\mathcal{K}_0$ (resp. $\mathcal{K}_{\mathsf{end}}$) in which `reset` commands have been removed and guarded inputs have been replaced by simple inputs but the occurrences occurring in $\mathtt{V}_{\mathsf{end}}(v_0, p_0)$, following the same trace as before, we have that:

$$\mathcal{K}^\star_0 \xrightarrow{\mathsf{tr}_1 \ldots . \mathsf{tr}_n}_{\mathcal{T}} \mathcal{K}^\star_{\mathsf{end}}.$$

Indeed, all the required conditions to trigger a simple input will be satisfied since a guarded input is like a simple input with a constraint regarding time. We denote $\mathcal{K}^\star_i$ with $i \in \{0, \ldots, n\}$ the intermediate configurations, and thus we have that $\mathcal{K}^\star_{\mathsf{end}} = \mathcal{K}^\star_n$.

Step 2: By definition of the untimed semantics we have that:

$$\widetilde{\mathcal{K}}^\star_0 \xrightarrow{\widetilde{\mathsf{tr}_1}}_{\mathcal{T}} \widetilde{\mathcal{K}}^\star_1 \xrightarrow{\widetilde{\mathsf{tr}_2}}_{\mathcal{T}} \ldots \xrightarrow{\widetilde{\mathsf{tr}_n}}_{\mathcal{T}} \widetilde{\mathcal{K}}^\star_{\mathsf{end}} = \widetilde{\mathcal{K}}^\star_n$$

where $\widetilde{\mathcal{K}}^\star_{\mathsf{end}}$, $\widetilde{\mathcal{K}}^\star_i$, and $\widetilde{\mathsf{tr}}_i$ are the untimed counterparts of $\mathcal{K}^\star_{\mathsf{end}}$, $\mathcal{K}^\star_i$ and tr_i.

Due to the specific shape of the process $\mathtt{V}_{\mathsf{end}}(v_0, p_0)$, we know that $\mathsf{tr}_1 . \ldots . \mathsf{tr}_n$ contains subsequences $\mathsf{tr}_{i_0} . \ldots . \mathsf{tr}_{j_0}$ such that tr_{i_0} corresponds to a reset action and tr_{j_0} to a guarded input. Note that these two actions tr_{i_0} and tr_{j_0} are performed by v_0. Moreover we know that for all index i in between i_0 and j_0 we have that tr_i is not a guarded input. Applying Proposition 4.2 to such a subsequence we obtain that there exists a bijection $\varphi : \{i_0, \ldots, j_0\} \to \{i_0, \ldots, j_0\}$ such that:

- $\widetilde{\mathcal{K}}^\star_{i_0 - 1} \xrightarrow{\mathsf{tr}'_{i_0} . \ldots . \mathsf{tr}'_{j_0}}_{\mathcal{T}} \widetilde{\mathcal{K}}^\star_{j_0}$ with $\widetilde{\mathsf{tr}}_i = \mathsf{tr}'_{\varphi(i)}$ for all $i \in \{i_0, \ldots, j_0\}$; and

- for all j such that $\varphi(i_0) < j < \varphi(j_0)$, we have that $\mathsf{tr}'_{\varphi(j_0)} \hookrightarrow^* \mathsf{tr}'_j \hookrightarrow^* \mathsf{tr}'_{\varphi(i_0)}$.

We are now showing that any agent involved in an action between $\mathsf{tr}'_{\varphi(i_0)}$ and $\mathsf{tr}'_{\varphi(j_0)}$ in the new resulting trace is actually in the vicinity of v_0, i.e. an agent in the set:

$$\mathsf{Close}(v_0) \stackrel{\mathsf{def}}{=} \{a \in \mathcal{A}_0 \mid \mathsf{Dist}_{\mathcal{T}}(v_0, a) < t_0\}.$$

Let j be such that $\varphi(i_0) < j < \varphi(j_0)$. Since we have that $\mathsf{tr}'_{\varphi(j_0)} \hookrightarrow^* \mathsf{tr}'_j \hookrightarrow^* \mathsf{tr}'_{\varphi(i_0)}$, we deduce that $\widetilde{\mathsf{tr}}_{j_0} \hookrightarrow^* \widetilde{\mathsf{tr}}_{\varphi^{-1}(j)} \hookrightarrow^* \widetilde{\mathsf{tr}}_{i_0}$ and thus $\mathsf{tr}_{j_0} \hookrightarrow^* \mathsf{tr}_{\varphi^{-1}(j)} \hookrightarrow^* \mathsf{tr}_{i_0}$. Denoting $\mathsf{tr}_i = (a_i, \alpha_i, s_i, t_i, r_i)$ for all $i \in \{1, \ldots, n\}$, and applying Lemma 4.5 twice, we obtain that:

$$t_{j_0} \geq t_{\varphi^{-1}(j)} + \mathsf{Dist}_{\mathcal{T}}(a_{\varphi^{-1}(j)}, v_0) \text{ and } t_{\varphi^{-1}(j)} \geq t_{i_0} + \mathsf{Dist}_{\mathcal{T}}(v_0, a_{\varphi^{-1}(j)}).$$

Therefore, we have that $t_{j_0} - t_{i_0} \geq 2\mathsf{Dist}(v_0, a_{\varphi^{-1}(j)})$, and exploiting the shape of $\mathsf{V}_{\mathsf{end}}(v_0, p_0)$ we deduce that $2 \times t_0 > t_{j_0} - t_{i_0} \geq 2\mathsf{Dist}(v_0, a_{\varphi^{-1}(j)})$. In summary we have that all the actions executed between $\mathsf{tr}'_{\varphi(i_0)}$ and $\mathsf{tr}'_{\varphi(j_0)}$ are executed by agents $a \in \mathsf{Close}(v_0)$.

Similarly, for any index j such that $\varphi(i_0) < j < \varphi(j_0)$ and $\alpha_{\varphi^{-1}(j)} = \mathtt{in}(u)$ we have that either $b \in \mathsf{Close}(v_0)$ or $vars(R) \subseteq \phi(\widetilde{\mathcal{K}}^{\star}_{i_0-1})$ where $r_{\varphi^{-1}(j)} = (b, t_b, R)$. Indeed, assume that there exists $\mathsf{w} \in vars(R) \setminus dom(\phi(\widetilde{\mathcal{K}}^{\star}_{i_0-1}))$. We note i the index corresponding to this output and we have that $\varphi(i_0) < i < \varphi(j_0)$. Thus, we have that:

$$\mathsf{tr}'_{\varphi(j_0)} \hookrightarrow^* \mathsf{tr}'_j \hookrightarrow_d \mathsf{tr}'_i \hookrightarrow^* \mathsf{tr}'_{\varphi(i_0)}.$$

Lemma 4.5 and the definition of $\hookrightarrow_d$ give us the following equations:

- $t_{j_0} - t_{\varphi^{-1}(j)} \geq \mathsf{Dist}_{\mathcal{T}}(a_{\varphi^{-1}(j)}, v_0)$,
- $t_{\varphi^{-1}(j)} - t_b \geq \mathsf{Dist}_{\mathcal{T}}(b, a_{\varphi^{-1}(j)})$,
- $t_b - t_{\varphi^{-1}(i)} \geq \mathsf{Dist}_{\mathcal{T}}(a_{\varphi^{-1}(i)}, b)$, and
- $t_{\varphi^{-1}(i)} - t_{i_0} \geq \mathsf{Dist}_{\mathcal{T}}(v_0, a_{\varphi^{-1}(i)})$.

Relying on the triangle inequality, this leads to $2 \times t_0 > t_{j_0} - t_{i_0} \geq 2\mathsf{Dist}_{\mathcal{T}}(v_0, b)$, and this allows us to deduce that $b \in \mathsf{Close}(v_0)$.

Step 3: We consider the topology $\mathcal{T}' = (\mathcal{A}_0, \mathcal{M}_0, \mathsf{Loc}', v_0, p_0)$ such that $\mathsf{Loc}'(v_0) = \overline{\mathsf{Loc}(v_0)}$, and $\mathsf{Loc}'(p_0)$ is such that $\mathsf{Dist}_{\mathcal{T}'}(v_0, p_0) = t_0$ and:

$$\mathsf{Loc}'(a) = \begin{cases} \mathsf{Loc}'(v_0) & \text{if } a \in \mathsf{Close}(v_0) \\ \mathsf{Loc}'(p_0) & \text{otherwise.} \end{cases}$$

In this topology, the agents far away from v_0 are moved to p_0, and agents in the neighbourhood of v_0 are moved to v_0. We denote $\widetilde{\mathsf{tr}}_i = \mathsf{tr}'_i$ for $i \in \{1, \ldots, i_0 - 1, j_0 + 1, \ldots, n\}$, i.e. those not affected by Step 2. In this topology $\mathcal{T}'$, we only have two locations, and

we have that $\widetilde{\mathcal{K}}_0^\star \overset{\mathsf{tr}'_1.....\mathsf{tr}'_n}{\rightsquigarrow}_{\mathcal{T}'} \widetilde{\mathcal{K}}_{\mathsf{end}}^\star$ since the locations of the agents are no longer relevant in the untimed semantics.

Step 4: We now show that we can come back to a timed execution, i.e. one executable in the timed semantics by induction on the number of guarded inputs in the trace. Given a configuration $\widehat{\mathcal{K}_0}$ such that $\mathsf{untimed}(\widehat{\mathcal{K}_0}) = \widetilde{\mathcal{K}}_0^\star$, we show that there exists a configuration $\widehat{\mathcal{K}_n}$ such that $\mathsf{untimed}(\widehat{\mathcal{K}_n}) = \widetilde{\mathcal{K}}_n^\star = \widetilde{\mathcal{K}}_{\mathsf{end}}^\star$. To show this result, we split our execution trace $\widetilde{\mathcal{K}}_0^\star \overset{\mathsf{tr}'_1.....\mathsf{tr}'_n}{\rightsquigarrow}_{\mathcal{T}'} \widetilde{\mathcal{K}}_n^\star$ on several blocks of actions: a block is either a trace with no guarded input, or a sequence of actions starting with a `reset` and ending at the next occurrence of a guarded input. Note that the block of the first kind can easily be lifted to the timed semantics. Regarding the block of the second kind, we show that the lifting is possible thanks to the properties established at Step 2.

To conclude this step, we now show how to exploit properties established at Step 2 to lift a block starting with a reset instruction and ending with a guarded input.

Let $\widetilde{\mathcal{K}}_{\mathtt{reset}}^\star \overset{\mathsf{tr}'_{i_0}.....\mathsf{tr}'_{j_0}}{\rightsquigarrow}_{\mathcal{T}'} \widetilde{\mathcal{K}}_{\mathtt{in}}^\star$ be such a block, and let $\widehat{\mathcal{K}}$ be such that $\mathsf{untimed}(\widehat{\mathcal{K}}) = \widetilde{\mathcal{K}}_{\mathtt{reset}}^\star$, and let $\widehat{t}$ the global time of configuration $\widehat{\mathcal{K}}$. We have to show that there exists $\widehat{\mathcal{K}_{\mathtt{in}}}$ such that $\widehat{\mathcal{K}} \xrightarrow{\widehat{\mathsf{tr}_{i_0}}.....\widehat{\mathsf{tr}_{j_0}}}_{\mathcal{T}'} \widehat{\mathcal{K}_{\mathtt{in}}}$, with tr'_i the untimed counterpart of $\widehat{\mathsf{tr}_i}$ and $\mathsf{untimed}(\widehat{\mathcal{K}_{\mathtt{in}}}) = \widetilde{\mathcal{K}}_{\mathtt{in}}^\star$. We start by applying the rule TIM with the delay δ equals to $2 \times t_0$. Let $\widehat{\mathcal{K}_+}$ be the resulting configuration. Then we have to show that the sequence of actions $\mathsf{tr}'_{i_0}.....\mathsf{tr}'_{j_0}$ can be executed without introducing any delay. Moreover, we show that the resulting configuration $\widehat{\mathcal{K}_{\mathtt{in}}}$ is such that $\mathsf{untimed}(\widehat{\mathcal{K}_{\mathtt{in}}}) = \widetilde{\mathcal{K}}_{\mathtt{in}}^\star$. Actually, the correspondence between timed and untimed configurations is maintained along the trace. The only difficult part is when the underlying action is an input. We know that this input is performed by $a \in \mathsf{Close}(v_0)$. Let $\mathtt{in}^\star(u)$ be an input occurring in the block and let Φ' the current frame in the untimed semantics when this action occurs and $\widehat{\Phi}$ its corresponding frame in the timed trace. By definition of the untimed semantics, we know that there exists a recipe R such that $R\Phi'\downarrow = u$. Thus, we know that $R\widehat{\Phi}\downarrow = u$. To conclude, it remains to show that the timing constraints are satisfied. We distinguish two cases:

- The input has been forged by an agent $b \in \mathsf{Close}(v_0)$. Any w used in the recipe R is either in $dom(\phi(\widetilde{\mathcal{K}}_{\mathtt{reset}}^\star))$ or outputted after tr'_{i_0} by an agent located at the same place as v_0. In both cases, since the global time has elapsed of $2t_0$ between $\widehat{\mathcal{K}}$ and $\widehat{\mathcal{K}_+}$, we know that all these w will be available at time $\widehat{t} + 2t_0$ for b. Since a and b are located at the same place, we also have that this input can be done at time $\widehat{t} + 2t_0$.

- The input has been forged by an agent $b \notin \mathsf{Close}(v_0)$. In such a case, we know that $vars(R) \subseteq dom(\phi(\widetilde{\mathcal{K}}_{\mathtt{reset}}^\star))$, and thus thanks to the delay of $2t_0$ that has been applied between $\widehat{\mathcal{K}}$ and $\widehat{\mathcal{K}_+}$, we know that the input can be received at time $\widehat{t} + 2t_0$. Indeed, b can forge the message at time $\widehat{t} + t_0$ and thus it can be received at time $\widehat{t} + 2t_0$.

Note that, regarding the guarded input, the guard is trivially satisfied since no time

has elapsed since the reset has been performed. In summary we have:

$$\widehat{\mathcal{K}_0} \xrightarrow{\widehat{\mathsf{tr}_1}.....\widehat{\mathsf{tr}_n}}_{\mathcal{T}'} \widehat{\mathcal{K}_n}$$

with $\widetilde{\mathcal{K}_n^\star}$ the untimed counterpart of $\widehat{\mathcal{K}_n}$ and thus, we have that:

$$\widehat{\mathcal{K}_n} = (\lfloor \mathsf{end}(v_0, p_0) \rfloor_{v_0}^{\widehat{t_v}} \cup \widehat{\mathcal{P}}; \widehat{\Phi_n}; \widehat{t_n}) \text{ for some } \widehat{\mathcal{P}},\ \widehat{\Phi_n},\ \widehat{t_v} \text{ and } \widehat{t_n}.$$

Step 5: In order to finish the proof, it remains to reduce the topology $\mathcal{T}'$ to the topology $\mathcal{T}_{\mathsf{DH}}^{t_0} = (\mathcal{A}_{\mathsf{DH}}, \mathcal{M}_{\mathsf{DH}}, \mathsf{Loc}_{\mathsf{DH}}, v_0, p_0)$. Let us consider the renaming:

$$\rho(a) = \begin{cases} v_0 & \text{if } a \in \mathsf{Close}(v_0) \\ p_0 & \text{if } a \notin \mathsf{Close}(v_0) \text{ and } a \in \mathcal{M}_0 \\ e_0 & \text{if } a \notin \mathsf{Close}(v_0) \text{ and } a \notin \mathcal{M}_0. \end{cases}$$

Since $\mathsf{Loc}_{\mathsf{DH}}(\rho(a)) = \mathsf{Loc}'(a)$ for any $a \in \mathcal{A}_0$, and $\rho(a) \in \mathcal{M}_{\mathsf{DH}}$ if, and only if $a \in \mathcal{M}_0$, thanks to Lemma 4.4, we have that:

$$\widehat{\mathcal{K}_0}\rho \xrightarrow{\widehat{\mathsf{tr}_0}\rho.....\widehat{\mathsf{tr}_n}\rho}_{\mathcal{T}_{\mathsf{DH}}^{t_0}} \widehat{\mathcal{K}_n}\rho.$$

We can assume w.l.o.g. that $\widehat{\mathcal{K}_0}$ has global time 0, and only contain frame elements at time 0. Let $\Phi_0 = \phi(\widehat{\mathcal{K}_0})$. Thus, to conclude, it remains to show that the frame associated to $\widehat{\mathcal{K}_0}\rho$, i.e. $\phi(\widehat{\mathcal{K}_0}\rho)$, is the expected one. Thus, we have to establish that:

- $img(\lfloor \phi(\widehat{\mathcal{K}_0}\rho) \rfloor_{v_0}^0) = img(\lfloor \phi(\widehat{\mathcal{K}_0}\rho) \rfloor_{e_0}^0) = \emptyset$, and
- $img(\lfloor \phi(\widehat{\mathcal{K}_0}\rho) \rfloor_{p_0}^0) = \mathsf{Knows}(\mathcal{I}_0, p_0, \{v_0, p_0, e_0\})$.

Relying on the fact that any agent a such that $\rho(a) = v_0$ is honest, we have that:

$$\begin{array}{rcl} img(\lfloor \phi(\widehat{\mathcal{K}_0}\rho) \rfloor_{v_0}^0) & = & img(\lfloor \Phi_0\rho \rfloor_{v_0}^0) \\ & = & \bigcup_{\{a \in \mathcal{A}_0 \ \mid \ \rho(a) = v_0\}} img(\lfloor \Phi_0 \rfloor_a^0)\rho \\ & = & \bigcup_{\{a \in \mathcal{A}_0 \ \mid \ \rho(a) = v_0\}} \emptyset \\ & = & \emptyset \end{array}$$

Actually, the same reasoning applies regarding e_0. Then, we have that:

$$\begin{array}{rcl} img(\lfloor \phi(\widehat{\mathcal{K}_0}\rho) \rfloor_{p_0}^0) & = & \bigcup_{\{a \in \mathcal{A}_0 \ \mid \ \rho(a) = p_0\}} img(\lfloor \Phi_0 \rfloor_a^0)\rho \\ & = & \bigcup_{\{a \in \mathcal{A}_0 \ \mid \ \rho(a) = p_0\}} \mathsf{Knows}(\mathcal{I}_0, a, \mathcal{A}_0)\rho \\ & = & \bigcup_{\{a \in \mathcal{A}_0 \ \mid \ \rho(a) = p_0\}} \mathsf{Knows}(\mathcal{I}_0, \rho(a), \rho(\mathcal{A}_0)) \\ & = & \bigcup_{\{a \in \mathcal{A}_0 \ \mid \ \rho(a) = p_0\}} \mathsf{Knows}(\mathcal{I}_0, p_0, \{p_0, v_0, e_0\}) \end{array}$$

This allows us to conclude. □

Proofs of Chapter 6 C

In this chapter we provide the proof of the second direction of Theorem 6.1 presented in Chapter 6.

Proposition 6.2. *Let $\mathcal{P}_{\mathsf{db}}$ be protocol and $\mathcal{S}$ a set of valid initial configurations. If $\mathcal{P}_{\mathsf{db}}$ is causality-based secure w.r.t. $\mathcal{S}$ then $\mathcal{P}_{\mathsf{db}}$ is DB-secure w.r.t. $\mathcal{S}$.*

Proof. We assume that $\mathcal{P}_{\mathsf{db}}$ is not DB-secure, and thus there exist a valid initial configuration $\mathcal{K}_0 \in \mathcal{S}$ and an execution exec such that:

$$\mathsf{exec} = \mathcal{K}_0 \xrightarrow{\mathsf{tr}_1 \ldots \mathsf{tr}_n.(b_0,\mathsf{claim}(b_1,b_2,t_1^0,t_2^0),s,t,\emptyset)}_{\mathsf{Loc}} \mathcal{K}_{n+1}$$

with $b_1 \notin \mathcal{M}$ and $b_2 \notin \mathcal{M}$ and either:

1. there is no index $k \leq n$ such that $\mathsf{tr}_k = (a_k, \mathsf{check}(t_1^0, t_2^0, t_3^0), s_k, t_k, \emptyset)$; or
2. for any t with $t_1^0 \leq t \leq t_2^0$, we have that:

$$t_2^0 - t_1^0 < \mathsf{Dist}(\mathsf{Loc}(b_1, t_1^0), \mathsf{Loc}(b_2, t)) + \mathsf{Dist}(\mathsf{Loc}(b_2, t), \mathsf{Loc}(b_1, t_2^0))$$

Below, we note $\mathsf{tr}_i = (a_i, \alpha_i, s_i, t_i, r_i)$ for $i \in \{1, \ldots, n\}$. First, we apply Lemma 6.2. Therefore, we have that:

$$\mathsf{exec}' = \overline{\mathcal{K}_0} \overset{\mathsf{tr}'_1 \ldots \mathsf{tr}'_n.(b_0,\mathsf{claim}(b_1,b_2,t_1^0\sigma,t_2^0\sigma),s,\emptyset)}{\rightsquigarrow} \mathcal{K}'_{n+1}$$

where $\mathcal{K}'_{n+1} = \overline{\mathcal{K}_{n+1}}\sigma$ and for any $i \in \{1, \ldots, n+1\}$, we have that:

$$\mathsf{tr}'_i = \begin{cases} (a_i, \mathsf{timestamp}(t_i\sigma), s_i, \emptyset) & \text{if } \alpha_i = \mathtt{gettime} \\ (a_i, \alpha_i\sigma, s_i, (b_i, R_i\sigma)) & \text{if } r_i = (b_i, t_i^b, R_i) \\ (a_i, \alpha_i\sigma, s_i, r_i\sigma) & \text{otherwise} \end{cases}$$

where $\sigma = \sigma_{\mathsf{spe}} \circ \sigma_{\mathsf{time}}^{-1}$ assuming that σ_{spe} is the function used to transform $\mathcal{K}_0$ into $\overline{\mathcal{K}_0}$ and σ_{time} is the one associated to the execution exec.

We assume by contradiction that $\mathcal{P}_{\mathsf{db}}$ is causality-based secure, thus we know that there exist $i_0, j_0, k, k' \leq n$ with $i_0 \leq k' \leq j_0$, and $u \in \mathcal{T}(\Sigma_c, \mathcal{N} \cup \Sigma_0^+)$ such that:

- $\alpha_k\sigma = \mathsf{check}(t_1^0\sigma, t_2^0\sigma, u\sigma)$;
- $(a_{i_0}, \alpha_i\sigma) = (b_1, \mathsf{timestamp}(t_1^0\sigma))$;
- $(a_{j_0}, \alpha_j\sigma) = (b_1, \mathsf{timestamp}(t_2^0\sigma))$; and
- $a_{k'} = b_2$.

By definition of exec', we have that $\mathsf{tr}_k = (a_k, \mathsf{check}(t_1^0, t_2^0, u), s_k, t_k, \emptyset)$ for some time $t_k \in \mathbb{R}_+$, and this leads to a contradiction with item 1). Thus, we can assume from now that the condition stated in item 2) holds.

Now, we apply Proposition 6.3 to the sub-execution $\mathcal{K}'_{i_0-1} \xrightarrow{\mathsf{tr}'_{i_0}\ldots\mathsf{tr}'_{j_0}} \mathcal{K}'_{j_0}$ of exec', and we obtain that there exists a bijection $\varphi_\alpha : \{i_0, \ldots, j_0\} \to \{i_0, \ldots, j_0\}$ such that:

- $\mathsf{tr}'_i = \mathsf{tr}''_{\varphi_\alpha(i)}$ for all $i \in \{i_0, \ldots, j_0\}$;
- for all j such that $\varphi_\alpha(i_0) < j < \varphi_\alpha(j_0)$, we have that $\mathsf{tr}''_{\varphi_\alpha(j_0)} \hookrightarrow^* \mathsf{tr}''_j \hookrightarrow^* \mathsf{tr}''_{\varphi_\alpha(i_0)}$;
- for all j_1, j_2 such that $\varphi_\alpha(i_0) \leq j_1 < j_2 \leq \varphi_\alpha(j_0)$, we have that $\varphi_\alpha^{-1}(j_1) < \varphi_\alpha^{-1}(j_2)$; and
- $\mathcal{K}'_{i_0-1} \xrightarrow{\mathsf{tr}''_{i_0}\ldots\mathsf{tr}''_{j_0}} \mathcal{K}'_{j_0}$.

We have thus an execution exec'' such that:

$$\mathsf{exec}'' = \overline{\mathcal{K}_0} \xrightarrow{tr'_1\ldots tr'_{i_0-1}} \mathcal{K}'_{i_0-1} \xrightarrow{tr''_{i_0}\ldots tr''_{j_0}} \mathcal{K}'_{j_0} \xrightarrow{tr'_{j_0+1}\ldots tr'_n.(b_0,\mathsf{claim}(b_1,b_2,t_1^0\sigma,t_2^0\sigma),s,\emptyset)} \mathcal{K}'_{n+1}.$$

For sake of simplicity, for $i < i_0$ and $i > j_0$ we define $\mathsf{tr}''_i = \mathsf{tr}'_i$. For all $i \in \{1, \ldots, n\}$ we define a''_i the name of the agent executing tr''_i. If $i \in \mathsf{IN}(\mathsf{tr}''_1 \ldots \mathsf{tr}''_n)$, we also define R''_i (resp. b''_i) the recipe (resp. agent name) occurring in tr''_i. We have that $a''_{\varphi_\alpha(i)} = a_i$, $R''_{\varphi_\alpha(i)} = R_i\sigma$ and $b''_{\varphi_\alpha(i)} = b_i$.

In the following we aim at re-timing the trace exec'' without changing the amount of time that elapses between the two timestamps instructions. Once this is done, the dependencies

$$\mathsf{tr}''_{\varphi_\alpha(j_0)} \hookrightarrow^* \mathsf{tr}''_j \hookrightarrow^* \mathsf{tr}''_{\varphi_\alpha(i_0)}$$

for any j such that $\varphi_\alpha(i_0) < j < \varphi_\alpha(j_0)$ together with the fact that there exists such a j such that $a_j = b_2$ (since $\mathcal{P}_{\mathsf{db}}$ is assumed to be causality-based secure) will lead us to a contradiction thanks to Lemma 6.3. The remaining of the proof is there to formalise this idea.

We start by defining a function φ and a mobility plan Loc' such that φ satisfies $\mathcal{C}^{\mathsf{Loc}'}_{\mathsf{exec}''}$. Let $\mathcal{A}_0$ be the set of all the agents involved in the execution exec'' (i.e. executing an action or forging a message used to fill an input). We denote $\delta(t)$ the maximal distance w.r.t. Loc between two agents in $\mathcal{A}_0$ at time t, i.e.

$$\delta(t) = \max\{\mathsf{Dist}(\mathsf{Loc}(a,t), \mathsf{Loc}(b,t)) \mid a, b \in \mathcal{A}_0\}.$$

We define φ as follows for $i \in \{1, \ldots, n+1\}$:

$$\varphi(z_i) = \begin{cases} 2 \times \delta(t_{i_0}) \times \#\mathsf{IN}(tr''_1 \ldots tr''_i) + \#\mathsf{TS}(tr''_1 \ldots tr''_{i-1}) & \text{if } i < \varphi_\alpha(i_0) \\ t_{\varphi_\alpha^{-1}(i)} + \Delta & \text{if } \varphi_\alpha(i_0) \leq i \leq \varphi_\alpha(j_0) \\ \varphi(z_{\varphi_\alpha(j_0)}) + 1 + 2 \times \delta(t_{j_0}) \times \#\mathsf{IN}(tr''_{\varphi_\alpha(j_0)+1} \ldots tr''_i) & \\ +\#\mathsf{TS}(tr''_{\varphi_\alpha(j_0)+1} \ldots tr''_{i-1}) & \text{if } i > \varphi_\alpha(j_0) \end{cases}$$

with $\Delta = \varphi(z_{\varphi_\alpha(i_0)-1}) + 2 \times \delta(t_{i_0}) + 1$.

In addition, for all $i \in \mathsf{IN}(tr''_1 \ldots tr''_n)$, if $b''_i \in \mathcal{A} \backslash \mathcal{M}$, we define $\varphi(z_i^b) = \varphi(z_{\mathtt{orig}(j)})$ with j such that $R''_i = \mathsf{w}_j$. Otherwise, we note $\Phi''_{\varphi_\alpha(i_0)-1}$ the current frame when executing the action $tr''_{\varphi_\alpha(i_0)}$ in exec'' and we define $\varphi(z_i^b)$ as follows:

$$\varphi(z_i^b) = \begin{cases} \varphi(z_i) - \delta(t_{i_0}) & \text{if } i < \varphi_\alpha(i_0) \\ \varphi(z_{\varphi_\alpha(i_0)}) - \delta(t_{i_0}) & \text{if } \varphi_\alpha(i_0) \leq i \leq \varphi_\alpha(j_0) \text{ and } vars(R''_i) \subseteq dom(\Phi''_{\varphi_\alpha(i_0)-1}) \\ t^b_{\varphi_\alpha^{-1}(i)} + \Delta & \text{if } \varphi_\alpha(i_0) \leq i \leq \varphi_\alpha(j_0) \text{ and } vars(R''_i) \not\subseteq dom(\Phi''_{\varphi_\alpha(i_0)-1}) \\ \varphi(z_i) - \delta(t_{j_0}) & \text{if } i > \varphi_\alpha(j_0). \end{cases}$$

We consider the mobility plan Loc' such that:

$$\mathsf{Loc}'(a,t) = \begin{cases} \mathsf{Loc}(a, t_{i_0}) & \text{if } t \leq \varphi(z_{\varphi_\alpha(i_0)}) (= t_{i_0} + \Delta) \\ \mathsf{Loc}(a, t - \Delta) & \text{if } \varphi(z_{\varphi_\alpha(i_0)}) < t < \varphi(z_{\varphi_\alpha(j_0)}) \\ \mathsf{Loc}(a, t_{j_0}) & \text{if } t \geq \varphi(z_{\varphi_\alpha(j_0)}) (= t_{j_0} + \Delta). \end{cases}$$

We may note that Loc' is indeed a mobility plan. Now, we show that φ satisfies $\mathcal{C}^{\mathsf{Loc}'}_{\mathsf{exec}''}$.

We first establish that the two first items are indeed satisfied, i.e.

- $0 \leq \varphi(z_1) \leq \varphi(z_2) \leq \ldots \leq \varphi(z_n)$; and
- $\varphi(z_i) < \varphi(z_{i+1})$ for all $i \in \mathsf{TS}(\mathsf{tr}''_1, \ldots, \mathsf{tr}''_{n-1})$.

First, we note that $0 \leq \varphi(z_1)$, and $\varphi(z_i) \leq \varphi(z_{i+1})$ on the three intervals $1 \leq i < \varphi_\alpha(i_0)$, $\varphi_\alpha(i_0) \leq i \leq \varphi_\alpha(j_0)$ and $i > \varphi_\alpha(j_0)$. In addition, by definition of φ, we have that:

- $\varphi(z_{\varphi_\alpha(i_0)}) = t_{i_0} + \Delta > \varphi(z_{\varphi_\alpha(i_0)-1})$; and
- $\varphi(z_{\varphi_\alpha(j_0)+1}) > \varphi(z_{\varphi_\alpha(j_0)})$.

This concludes the proof regarding the first item. Now, we consider item 2. Let $i \in \mathsf{TS}(\mathsf{tr}''_1 \ldots \mathsf{tr}''_n)$. We distinguish several cases:

C

- Case $i < \varphi_\alpha(i_0) - 1$ or $i > \varphi_\alpha(j_0)$. In such a case, we have that:

$$\varphi(z_{i+1}) - \varphi(z_i) = \#\mathsf{TS}(\mathsf{tr}''_1 \dots \mathsf{tr}''_i) - \#\mathsf{TS}(\mathsf{tr}''_1 \dots \mathsf{tr}''_{i-1}) = 1 > 0$$

- Case $i = \varphi_\alpha(i_0) - 1$ or $i = \varphi_\alpha(j_0)$. In such a case, we have seen just before that:

$$\varphi(z_{\varphi_\alpha(i_0)}) = t_{i_0} + \Delta > \varphi(z_{\varphi_\alpha(i_0)-1}) \quad \text{and} \quad \varphi(z_{\varphi_\alpha(j_0)+1}) > \varphi(z_{\varphi_\alpha(j_0)}).$$

C

- Case $\varphi_\alpha(i_0) \leq i \leq \varphi_\alpha(j_0) - 1$. In such a case, Proposition 6.3 (item 3) tells us that $\varphi_\alpha^{-1}(i) < \varphi_\alpha^{-1}(i+1)$, and thus $\varphi_\alpha^{-1}(i)+1 \leq \varphi_\alpha^{-1}(i+1)$. From these two inequalities, and since time is non decreasing during an execution, we deduce that:

$$t_{\varphi_\alpha^{-1}(i)} < t_{\varphi_\alpha^{-1}(i)+1} \quad \text{and} \quad t_{\varphi_\alpha^{-1}(i)+1} \leq t_{\varphi_\alpha^{-1}(i+1)}.$$

  Note that the strict inequality comes from the fact that $\varphi_\alpha^{-1}(i)$ corresponds to an application of GTIM. Hence, we have that $t_{\varphi_\alpha^{-1}(i)} < t_{\varphi_\alpha^{-1}(i+1)}$. Therefore, we deduce that:

$$\varphi(z_{i+1}) - \varphi(z_i) = (t_{\varphi_\alpha^{-1}(i+1)} + \Delta) - (t_{\varphi_\alpha^{-1}(i)} + \Delta) > 0.$$

  This allows us to conclude regarding item 2.

We now establish that the last two items are also satisfied. We thus show that for any $i \in \mathsf{IN}(\mathsf{tr}''_1, \dots, \mathsf{tr}''_n)$:

- $\varphi(z_i) \geq \varphi(z_i^b) + \mathsf{Dist}(\mathsf{Loc}'(a_i'', \varphi(z_i)), \mathsf{Loc}'(b_i'', \varphi(z_i^b)))$;
- for all j such that $\mathsf{w}_j \in \mathit{vars}(R_i'') \setminus \mathit{dom}(\Phi_0)$,

$$\varphi(z_i^b) \geq \varphi(z_{\mathtt{orig}(j)}) + \mathsf{Dist}(\mathsf{Loc}'(b_i'', \varphi(z_i^b)), \mathsf{Loc}'(a''_{\mathtt{orig}(j)}, \varphi(z_{\mathtt{orig}(j)}))).$$

We distinguish several cases.
Case $i < \varphi_\alpha(i_0)$. In such a case, by definition of Loc', for any $a \in \mathcal{A}$ and $t \leq \varphi(z_{\varphi_\alpha(i_0)})$, we have that $\mathsf{Loc}'(a,t) = \mathsf{Loc}(a, t_{i_0})$. Moreover, a delay of at least $2 \times \delta(t_{i_0})$ is applied between an input and any action occurring before it. By consequence, for all $\mathsf{w}_j \in \mathit{vars}(R_i'') \setminus \mathit{dom}(\Phi_0)$ (where Φ_0 is the initial frame in $\mathcal{K}_0$), since $\mathtt{orig}(j) < i$ we have:

$$\varphi(z_i) \geq \varphi(z_{\mathtt{orig}}(j)) + 2 \times \delta(t_{i_0}).$$

In addition, since $\mathtt{orig}(j) < i < \varphi_\alpha(i_0)$, thanks to item 1, we know that:

$$\varphi(z_{\mathtt{orig}(j)}) \leq \varphi(z_i) \leq \varphi(z_{\varphi_\alpha(i_0)}).$$

Therefore, we have that:

- $\mathsf{Loc}'(a''_{\mathtt{orig}(j)}, \varphi(z_{\mathtt{orig}(j)})) = \mathsf{Loc}(a''_{\mathtt{orig}(j)}, t_{i_0})$; and

- $\mathsf{Loc}'(a''_i, \varphi(z_i)) = \mathsf{Loc}(a''_i, t_{i_0})$.

Hence we have:

$$\begin{array}{rcl} \varphi(z_i) & \geq & \varphi(z_{\mathtt{orig}(j)}) + 2 \times \delta(t_{i_0}) \\ & \geq & \varphi(z_{\mathtt{orig}(j)}) + \mathsf{Dist}(\mathsf{Loc}(a''_{\mathtt{orig}(j)}, t_{i_0}), \mathsf{Loc}(a''_i, t_{i_0}) \\ & \geq & \varphi(z_{\mathtt{orig}(j)}) + \mathsf{Dist}(\mathsf{Loc}'(a''_{\mathtt{orig}(j)}, \varphi(z_{\mathtt{orig}(j)})), \mathsf{Loc}'(a''_i, \varphi(z_i))). \end{array}$$

If $b''_i \in \mathcal{A} \setminus \mathcal{M}$, then we have $\varphi(z^b_i) = \varphi(z_{\mathtt{orig}(j)})$ and $a''_{\mathtt{orig}(j)} = b''_i$. Therefore, from the previous inequality, we deduce the result:

$$\varphi(z_i) \geq \varphi(z^b_i) + \mathsf{Dist}(\mathsf{Loc}'(b''_i, \varphi(z^b_i)), \mathsf{Loc}'(a''_i, \varphi(z_i))).$$

If $b''_i \in \mathcal{M}$, then we have $\varphi(z_i) = \varphi(z^b_i) + \delta(t_{i_0})$. For the same reasons as before we have that $\mathsf{Loc}'(b''_i, \varphi(z^b_i)) = \mathsf{Loc}(b''_i, t_{i_0})$, and $\mathsf{Loc}'(a''_i, \varphi(z_i)) = \mathsf{Loc}(a''_i, t_{i_0})$.

Therefore, we have that:

$$\begin{array}{rcl} \varphi(z_i) & \geq & \varphi(z^b_i) + \mathsf{Dist}(\mathsf{Loc}(b''_i, t_{i_0}), \mathsf{Loc}(a''_i, t_{i_0})) \\ & \geq & \varphi(z^b_i) + \mathsf{Dist}(\mathsf{Loc}'(b''_i, \varphi(z^b_i)), \mathsf{Loc}'(a''_i, \varphi(z_i))). \end{array}$$

Therefore, we are done regarding item 3 for this case.

Now, let j be such that $\mathsf{w}_j \in \mathit{vars}(R''_i) \setminus \mathit{dom}(\Phi_0)$ we have $\mathtt{orig}(j) < i$ and we have seen that:

$$\varphi(z_i) \geq \varphi(z_{\mathtt{orig}(j)}) + 2 \times \delta(t_{i_0}).$$

By definition of φ, we have that: $\varphi(z^b_i) + \delta(t_{i_0}) \geq \varphi(z_i)$. Therefore, we have that:

$$\begin{array}{rcl} \varphi(z^b_i) \geq & \varphi(z_{\mathtt{orig}(j)}) + \delta(t_{i_0}) \\ & \geq & \varphi(z_{\mathtt{orig}(j)}) + \mathsf{Dist}(\mathsf{Loc}(a''_{\mathtt{orig}(j)}, t_{i_0}), \mathsf{Loc}(b''_i, t_{i_0}) \\ & \geq & \varphi(z_{\mathtt{orig}(j)}) + \mathsf{Dist}(\mathsf{Loc}'(a''_{\mathtt{orig}(j)}, \varphi(z_{\mathtt{orig}(j)})), \mathsf{Loc}'(b''_i, \varphi(z^b_i))). \end{array}$$

This allows us to conclude for this case.

Case $j > \varphi_\alpha(j_0)$. This case is similar to the previous one. Note that we have that $\mathsf{Loc}'(a, t) = \mathsf{Loc}(a, t_{j_0})$ for any $a \in \mathcal{A}$ and any $t \geq \varphi(z_{\varphi_\alpha}(j_0)) = t_{j_0} + \Delta$.

Case $\varphi_\alpha(i_0) \leq i \leq \varphi_\alpha(j_0)$ and $\mathit{vars}(R''_i) \subseteq \mathit{dom}(\Phi''_{\varphi_\alpha(i_0)-1})$. If $b''_i \in \mathcal{A} \setminus \mathcal{M}$, then we have that $R''_i = \mathsf{w}_j$ for some j such that $\mathtt{orig}(j) < \varphi_\alpha(i_0)$, $\varphi(z^b_i) = \varphi(z_{\mathtt{orig}(j)})$, and $a''_{\mathtt{orig}(j)} = b''_i$. Therefore, item 4 is trivially satisfied.

We now consider item 3. Since $\mathtt{orig}(j) < \varphi_\alpha(i_0)$, we have that $\varphi(z_{\mathtt{orig}(j)}) < \varphi(z_{\varphi_\alpha(i_0)})$, and thus by definition of Loc', we have that:

- $\mathsf{Loc}'(a''_{\mathtt{orig}(j)}, \varphi(z_{\mathtt{orig}(j)})) = \mathsf{Loc}(a''_{\mathtt{orig}(j)}, t_{i_0})$; and
- $\mathsf{Loc}'(a''_i, \varphi(z_{\varphi_\alpha(i_0)})) = \mathsf{Loc}(a''_i, t_{i_0})$.

C

Actually, since $\texttt{orig}(j) < \varphi_\alpha(i_0)$, as before, we have that

$$\begin{array}{rcl}\varphi(z_{\phi_\alpha(i_0)}) & \geq & \varphi(z_{\texttt{orig}(j)}) + 2 \times \delta(t_0) \\ & \geq & \varphi(z_{\texttt{orig}(j)}) + \mathsf{Dist}(\mathsf{Loc}(a''_{\texttt{orig}(j)}, t_{i_0}), \mathsf{Loc}(a''_i, t_{i_0})) \\ & \geq & \varphi(z_{\texttt{orig}(j)}) + \mathsf{Dist}(\mathsf{Loc}'(a''_{\texttt{orig}(j)}, \varphi(z_{\texttt{orig}(j)}))), \mathsf{Loc}'(a''_i, \varphi(z_{\varphi_\alpha(i_0)})))) \\ & \geq & \varphi(z^b_i) + \mathsf{Dist}(\mathsf{Loc}'(b''_i, \varphi(z^b_i)), \mathsf{Loc}'(a''_i, \varphi(z_{\varphi_\alpha(i_0)}))).\end{array}$$

Finally, since Loc' is a mobility plan we have that:

$$\varphi(z_i) - \varphi(z_{\varphi_\alpha(i_0)}) \geq \qquad \mathsf{Dist}(\mathsf{Loc}'(a''_i, \varphi(z_i)), \mathsf{Loc}'(a''_i, \varphi(z_{\varphi_\alpha(i_0)}))).$$

Combining these two inequalities, and using the triangle inequality, we deduce that:

$$\varphi(z_i) \geq \varphi(z^b_i) + \mathsf{Dist}(\mathsf{Loc}'(b''_i, \varphi(z^b_i)), \mathsf{Loc}'(a''_i, \varphi(z_i))).$$

Now, if $b''_i \in \mathcal{M}$, then we have that

$$\varphi(z^b_i) = \varphi(z_{\varphi_\alpha(i_0)}) - \delta(t_{i_0}).$$

Thus we have that $\mathsf{Loc}'(b''_i, \varphi(z^b_i)) = \mathsf{Loc}(b''_i, t_{i_0})$. Regarding item 4, let j be such that $\mathsf{w}_j \in \mathit{vars}(R''_i) \setminus \mathit{dom}(\Phi_0)$. We have that $\texttt{orig}(j) < \varphi_\alpha(i_0)$ and thus we have:

$$\begin{array}{rcl}\varphi(z^b_i) & \geq & \varphi(z_{\texttt{orig}(j)}) + \delta(t_{i_0}) \\ & \geq & \varphi(z_{\texttt{orig}(j)}) + \mathsf{Dist}(\mathsf{Loc}(a''_{\texttt{orig}(j)}, t_{i_0}), \mathsf{Loc}(b''_i, t_{i_0}) \\ & \geq & \varphi(z_{\texttt{orig}(j)}) + \mathsf{Dist}(\mathsf{Loc}'(a''_{\texttt{orig}(j)}, \varphi(z_{\texttt{orig}(j)})), \mathsf{Loc}'(b''_i, \varphi(z^b_i))).\end{array}$$

Regarding item 3, we have that:

$$\begin{array}{rcl}\varphi(z_{\varphi_\alpha(i_0)}) & = & \varphi(z^b_i) + \delta(t_{i_0}) \\ & \geq & \varphi(z^b_i) + \mathsf{Dist}(\mathsf{Loc}(a''_i, t_{i_0}), \mathsf{Loc}(b''_i, t_{i_0})) \\ & \geq & \varphi(z^b_i) + \mathsf{Dist}(\mathsf{Loc}'(a''_i, \varphi(z_{\varphi_\alpha(i_0)})), \mathsf{Loc}'(b''_i, \varphi(z^b_i))).\end{array}$$

Moreover, since Loc' is a mobility plan we have:

$$\varphi(z_i) - \varphi(z_{\varphi_\alpha(i_0)}) \geq \mathsf{Dist}(\mathsf{Loc}'(a''_i, \varphi(z_i)), \mathsf{Loc}'(a''_i, \varphi(z_{\varphi_\alpha(i_0)}))).$$

Combining these two inequalities, we obtain:

$$\begin{array}{rcl}\varphi(z_i) & \geq & \varphi(z^b_i) + \mathsf{Dist}(\mathsf{Loc}'(a''_i, \varphi(z_{\varphi_\alpha(i_0)})), \mathsf{Loc}'(b''_i, \varphi(z^b_i))) \\ & & \quad + \mathsf{Dist}(\mathsf{Loc}'(a''_i, \varphi(z_i)), \mathsf{Loc}'(a''_i, \varphi(z_{\varphi_\alpha(i_0)}))) \\ & \geq & \varphi(z^b_i) + \mathsf{Dist}(\mathsf{Loc}'(a''_i, \varphi(z_i)), \mathsf{Loc}'(b''_i, \varphi(z^b_i))).\end{array}$$

This concludes this case.

Case $\varphi_\alpha(i_0) \leq i \leq \varphi_\alpha(j_0)$ and $\mathit{vars}(R''_i) \not\subseteq \mathit{dom}(\Phi''_{\varphi_\alpha(i_0)-1})$. In such a case, we have that the following inequality holds: $\varphi(z_{\varphi_\alpha(i_0)}) \leq \varphi(z_i) \leq \varphi(z_{\varphi_\alpha(j_0)})$. Therefore, we have that

$$\mathsf{Loc}'(a''_i, \varphi(z_i)) = \mathsf{Loc}'(a''_i, t_{\varphi^{-1}_\alpha(i)} + \Delta) = \mathsf{Loc}(a_{\varphi^{-1}_\alpha(i)}, t_{\varphi^{-1}_\alpha(i)}).$$

We know also that there exists k such that $\mathsf{w}_k \in \mathit{vars}(R''_i) \setminus \mathit{dom}(\Phi''_{\varphi_\alpha(i_0)-1})$. Actually, we have that:

$$\varphi(z^b_i) = t^b_{\varphi_\alpha^{-1}(i)} + \Delta.$$

This equality is trivial when $b''_i \in \mathcal{M}$ and also satisfied when $b''_i \in \mathcal{A} \setminus \mathcal{M}$. Indeed, in this last case, we have that $\varphi(z^b_i) = \varphi(z_{\mathtt{orig}(k)})$. Since $\mathtt{orig}(k) > \varphi_\alpha(i_0)$ we know that $\varphi(z_{\mathtt{orig}(k)}) = t_{\varphi_\alpha^{-1}(\mathtt{orig}(k))} + \Delta$ and since we have that $t_{\varphi_\alpha^{-1}(\mathtt{orig}(k))} = t^b_{\varphi_\alpha^{-1}(i)}$, we conclude that

$$\varphi(z^b_i) = t^b_{\varphi_\alpha^{-1}(i)} + \Delta.$$

C

Moreover, we know that $t^b_{\varphi_\alpha^{-1}(i)} \geq t_{i_0}$ since the input has been executed in the initial execution exec and the output corresponding to w_k in exec must have been executed after t_{i_0} (Remember that the trace has been clean up between i_0 and j_0 only). Thus, we have that

$$\varphi(z^b_i) = t^b_{\varphi_\alpha^{-1}(i)} + \Delta \geq t_{i_0} + \Delta = \varphi(z_{\varphi_\alpha(i_0)}).$$

Therefore, we deduce that:

$$\mathsf{Loc}'(b''_i, \varphi(z^b_i)) = \mathsf{Loc}'(b''_i, t^b_{\varphi_\alpha^{-1}(i)} + \Delta) = \mathsf{Loc}(b''_i, t^b_{\varphi_\alpha^{-1}(i)}).$$

Since the IN rule has been triggered in exec we have that:

$$t_{\varphi_\alpha^{-1}(i)} \geq t^b_{\varphi_\alpha^{-1}(i)} + \mathsf{Dist}(\mathsf{Loc}(a''_i, t_{\varphi_\alpha^{-1}(i)}), \mathsf{Loc}(b''_i, t^b_{\varphi_\alpha^{-1}(i)}))$$

Therefore, regarding item 3, we have that:

$$\begin{array}{rcl}
\varphi(z_i) & = & t_{\varphi_\alpha^{-1}(i)} + \Delta \\
& \geq & \left(t^b_{\varphi_\alpha^{-1}(i)} + \mathsf{Dist}(\mathsf{Loc}(a''_i, t_{\varphi_\alpha^{-1}(i)}), \mathsf{Loc}(b''_i, t^b_{\varphi_\alpha^{-1}(i)}))\right) + \Delta \\
& = & \varphi(z^b_i) + \mathsf{Dist}(\mathsf{Loc}(a''_i, t_{\varphi_\alpha^{-1}(i)}), \mathsf{Loc}(b''_i, t^b_{\varphi_\alpha^{-1}(i)})) \\
& = & \varphi(z^b_i) + \mathsf{Dist}(\mathsf{Loc}'(a''_i, t_{\varphi_\alpha^{-1}(i)} + \Delta), \mathsf{Loc}(b''_i, t^b_{\varphi_\alpha^{-1}(i)} + \Delta)) \\
& = & \varphi(z^b_i) + \mathsf{Dist}(\mathsf{Loc}'(a''_i, \varphi(z_i)), \mathsf{Loc}(b''_i, \varphi(z^b_i)))
\end{array}$$

This concludes the proof regarding item 3.

Regarding item 4, let $\mathsf{w} \in \mathit{vars}(R''_i)$. We know that $\mathsf{w} \in \mathit{vars}(R_{\varphi_\alpha^{-1}(i)})$, and $(\mathsf{w} \xrightarrow{c,t_c} u) \in \Phi$ (with Φ the current frame when executing the input in exec). If $c \neq \star$ then $t^b_{\varphi_\alpha^{-1}(i)} \geq t_c + \mathsf{Dist}(\mathsf{Loc}(b''_i, t^b_{\varphi_\alpha^{-1}(i)}), \mathsf{Loc}(c, t_c))$. By construction, we have that $(\mathsf{w} \xrightarrow{c,t'_c} u') \in \Phi''$ (with Φ'' the current frame when executing the input in exec'') with either $t'_c \leq \varphi(z_{\varphi_\alpha(i_0)-1})$ or $t'_c = t_c + \Delta$.

In the first case we have that $\mathsf{Loc}'(c, t'_c) = \mathsf{Loc}(c, t_{i_0})$. By definition of a mobility plan we know that

$$t_c - t_{i_0} \geq \mathsf{Dist}(\mathsf{Loc}(c, t_c), \mathsf{Loc}(c, t_{i_0})).$$

Combining this inequality with the previous one and using the triangle inequality, we obtain:

$$t^b_{\varphi_\alpha^{-1}(i)} \geq t_{i_0} + \mathsf{Dist}(\mathsf{Loc}(b''_i, t^b_{\varphi_\alpha^{-1}(i)}), \mathsf{Loc}(c, t_{i_0})).$$

Thus, we have that:

$$\begin{array}{rcl}\varphi(z_i^b) & = & t^b_{\varphi_\alpha^{-1}(i)} + \Delta \\ & \geq & t_{i_0} + \mathsf{Dist}(\mathsf{Loc}(b_i'', t^b_{\varphi_\alpha^{-1}(i)}), \mathsf{Loc}(c, t_{i_0})) + \varphi(z_{\varphi_\alpha(i_0)-1}) + 2 \times \delta(t_{i_0}) + 1 \\ & \geq & t'_c + \mathsf{Dist}(\mathsf{Loc}(b_i'', t^b_{\varphi_\alpha^{-1}(i)}), \mathsf{Loc}(c, t_{i_0})) \\ & = & t'_c + \mathsf{Dist}(\mathsf{Loc}'(b_i'', t^b_{\varphi_\alpha^{-1}(i)} + \Delta), \mathsf{Loc}'(c, t'_c)) \\ & = & t'_c + \mathsf{Dist}(\mathsf{Loc}'(b_i'', \varphi(z_i^b)), \mathsf{Loc}'(c, t'_c))\end{array}$$

C

In the second case, we have that $t'_c = t_c + \Delta$ and thus $\mathsf{Loc}'(c, t'_c) = \mathsf{Loc}(c, t_c)$. Therefore, we have that:

$$\begin{array}{rcl}\varphi(z_i^b) & = & t^b_{\varphi_\alpha^{-1}(i)} + \Delta \\ & \geq & t_c + \mathsf{Dist}(\mathsf{Loc}(b_i'', t^b_{\varphi_\alpha^{-1}(i)}), \mathsf{Loc}(c, t_c)) + \Delta \\ & = & t'_c + \mathsf{Dist}(\mathsf{Loc}'(b_i'', \varphi(z_i^b)), \mathsf{Loc}'(c, t'_c))\end{array}$$

This allows us to conclude regarding item 4.

In conclusion we have that Lemma 6.1 applies with exec'' and φ. We have thus a timed execution $\mathsf{exec}_{\mathsf{time}}$ in the mobility plan Loc' such that

$$\begin{array}{rl}\mathsf{exec}_{\mathsf{time}} = & \mathcal{K}_0 \xrightarrow{\mathsf{tr}_1'''\ldots\mathsf{tr}_n'''}_{\mathsf{Loc}} \mathcal{K}_n'' \\ & \xrightarrow{(b_0,\mathsf{claim}(b_1,b_2,t_1^0\sigma\varphi_c,t_2^0\sigma\varphi_c),s,\varphi(z_{n+1}),\emptyset)}_{\mathsf{Loc}'} \mathcal{K}_{n+1}''\end{array}$$

with $\varphi_c(c_i) = \varphi(z_i)$ for all $i \in TS(\mathsf{tr}_1'' \ldots \mathsf{tr}_n'')$.

Since the causality-based secure property holds, we know, by unicity of the timestamps, that there exists an index k' such that $\varphi(i_0) \leq k' \leq \varphi(j_0)$ and $a_{k'} = b_2$. Moreover, from the cleaning performed relying on Corollary 6.3 we have that $\mathsf{tr}''_{\varphi_\alpha(j_0)} \hookrightarrow^* \mathsf{tr}''_{k'} \hookrightarrow^* \mathsf{tr}''_{\varphi_\alpha(i_0)}$ which is equivalent to $\mathsf{tr}'''_{\varphi_\alpha(j_0)} \hookrightarrow^* \mathsf{tr}'''_{k'} \hookrightarrow^* \mathsf{tr}'''_{\varphi_\alpha(i_0)}$. Applying Lemma 4.5 we obtain that:

$$\begin{array}{rcl}\varphi(z_{\varphi_\alpha(j_0)}) - \varphi(z_{k'}) & \geq & \mathsf{Dist}(\mathsf{Loc}'(a_{k'}, \varphi(z_{k'})), \mathsf{Loc}'(b_1, \varphi(z_{\varphi_\alpha(j_0)}))) \\ & \geq & \mathsf{Dist}(\mathsf{Loc}'(a_{k'}, t_{\varphi_\alpha^{-1}(k')} + \Delta), \mathsf{Loc}'(b_1, t_{j_0} + \Delta)) \\ & \geq & \mathsf{Dist}(\mathsf{Loc}(a_{k'}, t_{\varphi_\alpha^{-1}(k')}), \mathsf{Loc}(b_1, t_{j_0})).\end{array}$$

Similarly we have that:

$$\varphi(z_{k'}) - \varphi(z_{\varphi_\alpha(i_0)}) \geq \mathsf{Dist}(\mathsf{Loc}(a_{k'}, t_{\varphi_\alpha^{-1}(k')}), \mathsf{Loc}(b_1, t_{i_0})).$$

Combining these two inequalities we obtain:

$$\begin{array}{rcl}\varphi(z_{\varphi_\alpha(j_0)}) - \varphi(z_{\varphi_\alpha(i_0)}) & \geq & \mathsf{Dist}(\mathsf{Loc}(a_{k'}, t_{\varphi_\alpha^{-1}(k')}), \mathsf{Loc}(b_1, t_{j_0})) \\ & & +\mathsf{Dist}(\mathsf{Loc}(a_{k'}, t_{\varphi_\alpha^{-1}(k')}), \mathsf{Loc}(b_1, t_{i_0}))\end{array}$$

and thus:

$$\begin{array}{rcl}t_{j_0} - t_{i_0} & \geq & \mathsf{Dist}(\mathsf{Loc}(a_{b_2}, t_{\varphi_\alpha^{-1}(k')}), \mathsf{Loc}(b_1, t_{j_0})) \\ & & +\mathsf{Dist}(\mathsf{Loc}(a_{b_2}, t_{\varphi_\alpha^{-1}(k')}), \mathsf{Loc}(b_1, t_{i_0}))\end{array}$$

This leads to a contradiction. Hence the results. □

Bibliography

[ABB+18] Gildas Avoine, Muhammed Ali Bingöl, Ioana Boureanu, Srdjan čapkun, Gerhard Hancke, Süleyman Kardaş, Chong Hee Kim, Cédric Lauradoux, Benjamin Martin, Jorge Munilla, Alberto Peinado, Kasper Bonne Rasmussen, Dave Singelée, Aslan Tchamkerten, Rolando Trujillo-Rasua, and Serge Vaudenay. Security of distance-bounding: A survey. *ACM Comput. Surv.*, 51(5), September 2018.

[ABG+17] Gildas Avoine, Xavier Bultel, Sébastien Gambs, David Gerault, Pascal Lafourcade, Cristina Onete, and Jean-Marc Robert. A terrorist-fraud resistant and extractor-free anonymous distance-bounding protocol. In *Proc. 12th ACM Asia Conference on Computer and Communications Security*. ACM Press, 2017.

[ABK+11] Gildas Avoine, Muhammed Ali Bingöl, Süleyman Kardaş, Cédric Lauradoux, and Benjamin Martin. A framework for analyzing RFID distance bounding protocols. *Journal of Computer Security*, 19(2):289–317, 2011.

[ACC+08] A. Armando, R. Carbone, L. Compagna, J. Cuéllar, and M. L. Tobarra. Formal analysis of SAML 2.0 web browser single sign-on: breaking the SAML-based single sign-on for Google apps. In *Proc. 6th ACM Workshop on Formal Methods in Security Engineering (FMSE'08)*, pages 1–10. ACM, 2008.

[AF01] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. 28th Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115. ACM Press, 2001.

[AT09] Gildas Avoine and Aslan Tchamkerten. An efficient distance bounding rfid authentication protocol: balancing false-acceptance rate and memory requirement. In *International Conference on Information Security*, pages 250–261. Springer, 2009.

[BBK17] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. Verified models and reference implementations for the tls 1.3 standard candidate. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 483–502. IEEE, 2017.

[BC93] Stefan Brands and David Chaum. Distance-bounding protocols. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 344–359. Springer, 1993.

[BCDD20] Ioana Boureanu, Tom Chothia, Alexandre Debant, and Stéphanie Delaune. Security Analysis and Implementation of Relay-Resistant Contactless Payments. In *(To appear in) Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2020.

[BCSS11] David Basin, Srdjan Capkun, Patrick Schaller, and Benedikt Schmidt. Formal reasoning about physical properties of security protocols. *ACM Transactions on Information and System Security (TISSEC)*, 14(2):16, 2011.

[BDGK17] David Baelde, Stéphanie Delaune, Ivan Gazeau, and Steve Kremer. Symbolic verification of privacy-type properties for security protocols with xor. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 234–248. IEEE, 2017.

[BDH17] David Baelde, Stéphanie Delaune, and Lucca Hirschi. A reduced semantics for deciding trace equivalence. *Logical Methods in Computer Science*, 13(2), 2017.

[BGG+16] Xavier Bultel, Sébastien Gambs, David Gerault, Pascal Lafourcade, Cristina Onete, and Jean-Marc Robert. A prover-anonymous and terrorist-fraud resistant distance-bounding protocol. In *Proc. 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks, (WISEC'16)*, pages 121–133. ACM Press, 2016.

[BGL15] Agnès Brelurut, David Gerault, and Pascal Lafourcade. Survey of distance bounding protocols and threats. In *International Symposium on Foundations and Practice of Security*, pages 29–49. Springer, 2015.

[Bla01] B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *Proc. 14th Computer Security Foundations Workshop (CSFW'01)*, pages 82–96. IEEE Computer Society Press, 2001.

[BMV13] Ioana Boureanu, Aikaterini Mitrokotsa, and Serge Vaudenay. Secure and lightweight distance-bounding. In *International Workshop on Lightweight Cryptography for Security and Privacy*, pages 97–113. Springer, 2013.

[CBC19] Tom Chothia, Ioana Boureanu, and Liqun Chen. Making contactless emv robust against rogue readers colluding with relay attackers. In *Proceedings of the 23rd International Conference on Financial Cryptography and Data Security (FC 19)*. International Financial Cryptography Association, 2019.

[ČBH03] Srdjan Čapkun, Levente Buttyán, and Jean-Pierre Hubaux. Sector: secure tracking of node encounters in multi-hop wireless networks. In *Proc. 1st ACM workshop on Security of ad hoc and sensor networks*, pages 21–32. ACM, 2003.

[CCCK16] Rohit Chadha, Vincent Cheval, Ştefan Ciobâcă, and Steve Kremer. Automated verification of equivalence properties of cryptographic protocol. *ACM Transactions on Computational Logic*, 23(4), 2016.

[CDD12] Véronique Cortier, Jan Degrieck, and Stéphanie Delaune. Analysing routing protocols: four nodes topologies are sufficient. In *International Conference on Principles of Security and Trust*, pages 30–50. Springer, 2012.

[CdRS18] Tom Chothia, Joeri de Ruiter, and Ben Smyth. Modelling and analysis of a hierarchy of distance bounding attacks. In *Proc. 27th USENIX Security Symposium, USENIX Security 2018*, 2018.

[CGdR+15] Tom Chothia, Flavio D. Garcia, Joeri de Ruiter, Jordi van den Breekel, and Matthew Thompson. Relay cost bounding for contactless EMV payments. In *Proc. 19th International Conference on Financial Cryptography and Data Security (FC'15)*, volume 8975 of *LNCS*. Springer, 2015.

[CGG19] Véronique Cortier, Pierrick Gaudry, and Stephane Glondu. Belenios: a simple private and verifiable electronic voting system. In *Foundations of Security, Protocols, and Equational Reasoning*, pages 214–238. Springer, 2019.

[CKR18] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. Deepsec: deciding equivalence properties in security protocols theory and practice. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 529–546. IEEE, 2018.

[CLC03] Hubert Comon-Lundh and Véronique Cortier. Security properties: two agents are sufficient. *Programming Languages and Systems*, pages 99–113, 2003.

[CLD05] Hubert Comon-Lundh and Stéphanie Delaune. The finite variant property: How to get rid of some algebraic properties. In *International Conference on Rewriting Techniques and Applications*, pages 294–307. Springer, 2005.

[Con76] John H Conway. On numbers and games, acad. *Press, London, New York, San Francisco*, 1976.

[CRSC12] Cas Cremers, Kasper B Rasmussen, Benedikt Schmidt, and Srdjan Capkun. Distance hijacking attacks on distance bounding protocols. In *Proc. IEEE Symposium on Security and Privacy (S&P'12)*, pages 113–127. IEEE, 2012.

[CSV17] Rohit Chadha, A Prasad Sistla, and Mahesh Viswanathan. Verification of randomized security protocols. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12. IEEE, 2017.

[Dan63] GB Dantzig. Linear programming and extensions, princeton, univ. *Press, Princeton, NJ*, 1963.

[DD19] Alexandre Debant and Stéphanie Delaune. Symbolic verification of distance bounding protocols. In *Proc. 8th International Conference on Principles of Security and Trust (POST'19)*, LNCS. Springer, 2019.

[DDW18] Alexandre Debant, Stéphanie Delaune, and Cyrille Wiedling. A symbolic framework to analyse physical proximity in security protocols. In *Proc. 38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, (FSTTCS'18)*, volume 122 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

[DDW19] Alexandre Debant, Stéphanie Delaune, and Cyrille Wiedling. Symbolic analysis of terrorist fraud resistance. In *European Symposium on Research in Computer Security*, pages 383–403. Springer, 2019.

[Des88] Yvo Desmedt. Major security problems with the 'unforgeable'(feige)-fiat-shamir proofs of identity and how to overcome them. In *Proceedings of SECURICOM*, volume 88, pages 15–17, 1988.

[DFKO11] Ulrich Dürholz, Marc Fischlin, Michael Kasper, and Cristina Onete. A formal approach to distance-bounding RFID protocols. In *Proc. 14th International Conference on Information Security (ISC'11)*, volume 7001 of *LNCS*. Springer, 2011.

[DGB87] Yvo Desmedt, Claude Goutier, and Samy Bengio. Special uses and abuses of the Fiat-Shamir passport protocol. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 21–39. Springer, 1987.

[DHRS18] Jannik Dreier, Lucca Hirschi, Sasa Radomirovic, and Ralf Sasse. Automated Unbounded Verification of Stateful Cryptographic Protocols with Exclusive OR. In *CSF'2018 - 31st IEEE Computer Security Foundations Symposium*, Oxford, United Kingdom, July 2018.

[Dil89] David L Dill. Timing assumptions and verification of finite-state concurrent systems. In *International Conference on Computer Aided Verification*, pages 197–212. Springer, 1989.

[DY83] Danny Dolev and Andrew Yao. On the security of public key protocols. *IEEE Transactions on information theory*, 29(2):198–208, 1983.

[EMV16] EMVCo. EMV contactless specifications for payment systems, version 2.6, 2016.

[FDC11] Aurélien Francillon, Boris Danev, and Srdjan Capkun. Relay attacks on passive keyless entry and start systems in modern cars. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Eidgenössische Technische Hochschule Zürich, Department of Computer Science, 2011.

[FHMM10] Lishoy Francis, Gerhard Hancke, Keith Mayes, and Konstantinos Markantonakis. Practical nfc peer-to-peer relay attack using mobile phones. In *International Workshop on Radio Frequency Identification: Security and Privacy Issues*, pages 35–49. Springer, 2010.

[FO13] Marc Fischlin and Cristina Onete. Subtle kinks in distance-bounding: an analysis of prominent protocols. In *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks*, pages 195–206, 2013.

[Ger18] David Gerault. *Security Analysis of Contactless Communication Protocols*. PhD thesis, Université Clermont Auvergne, 2018.

[Gro16] Trusted Computing Group. Trusted platform module library family 2.0, specification - part 1: Architecture, revision 1.38 and part 3: Commands, revision 1.38. Technical report, 2016.

[Han05] Gerhard P Hancke. A practical relay attack on iso 14443 proximity cards. *Technical report, University of Cambridge Computer Laboratory*, 59:382–385, 2005.

[HK05] Gerhard P Hancke and Markus G Kuhn. An RFID distance bounding protocol. In *Proc. 1st International Conference on Security and Privacy for Emerging Areas in Communications Networks (SECURECOMM'05)*, pages 67–73. IEEE, 2005.

[Jan17] Pieter Janssens. Proximity check for communication devices, October 31 2017. US Patent 9,805,228.

[KAK+08] Chong Hee Kim, Gildas Avoine, François Koeune, François-Xavier Standaert, and Olivier Pereira. The Swiss-Knife RFID distance bounding protocol. In *Proc. 11th International Conference on Information Security and Cryptology (ICISC'08)*, volume 5461 of *LNCS*. Springer, 2008.

[KT11] Ralf Küsters and Tomasz Truderung. Reducing protocol analysis with xor to the xor-free case in the horn theory based approach. *Journal of Automated Reasoning*, 46(3-4):325–352, 2011.

[KV16] Handan Kilinç and Serge Vaudenay. Efficient public-key distance bounding protocol. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and*

Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II, volume 10032 of *Lecture Notes in Computer Science*, pages 873–901, 2016.

[LSLD15] Li Li, Jun Sun, Yang Liu, and Jin Song Dong. Verifying parameterized timed security protocols. In *International Symposium on Formal Methods*, pages 342–359. Springer, 2015.

[MBK10] Sreekanth Malladi, Bezawada Bruhadeshwar, and Kishore Kothapalli. Automatic analysis of distance bounding protocols. *arXiv preprint arXiv:1003.5383*, 2010.

[MP08] Jorge Munilla and Alberto Peinado. Distance bounding protocols for rfid enhanced by using void-challenges and analysis in noisy channels. *Wireless communications and mobile computing*, 8(9):1227–1232, 2008.

[MPP+07] Catherine Meadows, Radha Poovendran, Dusko Pavlovic, LiWu Chang, and Paul Syverson. Distance bounding protocols: Authentication logic analysis and collusion attacks. In *Secure localization and time synchronization for wireless sensor and ad hoc networks*, pages 279–298. Springer, 2007.

[MSCB13] S. Meier, B. Schmidt, C. Cremers, and D. Basin. The Tamarin Prover for the Symbolic Analysis of Security Protocols. In *Proc. 25th International Conference on Computer Aided Verification (CAV'13)*, volume 8044 of *LNCS*, pages 696–701. Springer, 2013.

[MSDL99] J Mitchell, A Scedrov, N Durgin, and P Lincoln. Undecidability of bounded security protocols. In *Workshop on Formal Methods and Security Protocols*, 1999.

[MSTPTR18] S. Mauw, Z. Smith, J. Toro-Pozo, and R. Trujillo-Rasua. Distance-bounding protocols: Verification without time and location. In *Proc. 39th IEEE Symposium on Security and Privacy (S&P'18)*, pages 152–169, 2018.

[MSTPTR19] Sjouke Mauw, Zach Smith, Jorge Toro-Pozo, and Rolando Trujillo-Rasua. Post-Collusion Security and Distance Bounding. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2019.

[MTPTR16] Sjouke Mauw, Jorge Toro-Pozo, and Rolando Trujillo-Rasua. A class of precomputation-based distance-bounding protocols. In *Proc. 1st IEEE European Symposium on Security and Privacy (EuroS&P'16)*. IEEE, 2016.

[MVB10] Sebastian Mödersheim, Luca Viganò, and David A. Basin. Constraint differentiation: Search-space reduction for the constraint-based analysis of security protocols. *Journal of Computer Security*, 18(4):575–618, 2010.

[NPW02] Tobias Nipkow, Lawrence C Paulson, and Markus Wenzel. *Isabelle/HOL: a proof assistant for higher-order logic*, volume 2283. Springer Science & Business Media, 2002.

[NTU16] Vivek Nigam, Carolyn Talcott, and Abraão Aires Urquiza. Towards the automated verification of cyber-physical security protocols: Bounding the number of timed intruders. In *Proc. 21st European Symposium on Research in Computer Security (ESORICS'16)*, pages 450–470. Springer, 2016.

[RC10] Kasper Bonne Rasmussen and Srdjan Capkun. Realization of rf distance bounding. In *USENIX Security Symposium*, pages 389–402, 2010.

[RT03] Michaël Rusinowitch and Mathieu Turuani. Protocol insecurity with a finite number of sessions and composed keys is np-complete. *Theoretical Computer Science*, 299(1-3):451–475, 2003.

[SC13] Luigi Sportiello and Andrea Ciardulli. Long distance relay attack. In *International Workshop on Radio Frequency Identification: Security and Privacy Issues*, pages 69–85. Springer, 2013.

[TRMA10] Rolando Trujillo-Rasua, Benjamin Martin, and Gildas Avoine. The poulidor distance-bounding protocol. In *International Workshop on Radio Frequency Identification: Security and Privacy Issues*, pages 239–257. Springer, 2010.

UNIVERSITÉ DE RENNES 1

Titre : Vérification formelle des protocoles délimiteurs de distance - Application aux protocoles de paiement

Mot clés : méthode formelle, modèle symbolique, protocole cryptographique, protocole de paiement

Resumé :

L'essor des nouvelles technologies, et en particulier la Communication en Champ Proche (NFC), a permis l'apparition de nouvelles applications. Á ce titre, nous pouvons mentionner le paiement sans contact, les clefs mains libres ou encore les carte d'abonnement dans les transports en commun. Afin de sécuriser l'ensemble de ces applications, des protocoles de sécurité, appelés protocoles délimiteurs de distance on été développés. Ces protocoles ont pour objectif d'assurer la proximité physique des appareils mis en jeu afin de limiter le risque d'attaque. Dans ce manuscrit, nous présentons diverses approches permettant une analyse formelle de ces protocoles. Dans ce but, nous proposons un modèle symbolique permettant une modélisation précise du temps ainsi que des positions dans l'espace de chaque participant. Nous proposons ensuite deux approches : la première développant une nouvelle procédure de vérification, la seconde permettant la ré-utilisation d'outils existants tels que Proverif. Tout au long de ce manuscrit, nous porterons une attention particulières aux protocoles de paiement sans contact.

Title : Symbolic verification of distance-bounding protocols - Application to payment protocols

Keywords : formal method, symbolic model, cryptographic protocol, payment protocol

Abstract :

The rise of new technologies, and in particular Near Field Communication (NFC) tags, offers new applications such as contactless payments, key-less entry systems, transport ticketing... Due to their security concerns, new security protocols, called distance-bounding protocols, have been developed to ensure the physical proximity of the devices during a session. In order to prevent flaws and attacks, these protocols require formal verification. In this manuscript, we present several techniques that allow for an automatic verification of such protocols. To this aim, we first present a symbolic model which faithfully models time and locations. Then we develop two approaches : either based on a new verification procedure, or leveraging existing tools like Proverif. Along this manuscript, we pay a particular attention to apply our results to contactless payment protocols.